Inaugural-Dissertation

zur Erlangung der Doktorwürde der

Naturwissenschaftlich-Mathematischen Gesamtfakultät

der

Ruprecht-Karls-Universität Heidelberg

vorgelegt von
Diplom-Mathematiker

Lutz Büch

aus Arnsberg (Westfalen)

Tag der mündlichen Prüfung:

Metric Selection and Metric Learning for Matching Tasks

Betreuer, Erstgutachter:	Prof. Dr. Artur Andrzejak
Zweitgutachter:	Prof. Dr. Holger Fröning

Zusammenfassung

Ein Vierteljahrhundert nachdem das World-Wide Web eingeführt wurde, haben wir uns sehr daran gewöhnt, einfachen Zugang zu einer riesigen Menge an Daten und Open-Source Software zu haben. Der Wert dieser Resourcen ist allerdings dadurch bedingt, dass sie ordentlich integriert und gewartet werden. Ein Großteil solcher Arbeit läuft auf Matching hinaus: Das Auffinden von existierenden Datensätzen, um sie mit weiterer Information aus neuen Datensätzen anzureichern; das Integrieren von Code in eine bestehende Code-Basis, ohne gleichzeitig Duplikation einzuführen.

In dieser Arbeit gehen wir zwei verschiedene solcher Matching-Probleme an. Erstens machen wir Gebrauch von der vielfältigen und ausgereiften Menge an String-Ähnlichkeitsmaßen um in einem iterativen, halb-überwachten Lernansatz das String-Matching-Problem zu lösen. Er ist so angelegt, dass der User eine Sequenz an Einzelfällen des String-Matchings entscheiden muss. Wir zeigen dass mit einer nur sehr kleinen Menge an Entscheidungen fast optimale Lösungen gefunden werden können. Der geringe Annotationsaufwand unseres Algorithmus kommt daher, dass wir das Cold-Start-Problem, das dem Active Learning innewohnt, auf zweierlei Arten behandeln. Einerseits durch das Ordnen der Instanzen nach ihrer Rang-Varianz, solange noch nicht genug überwachte Information vorliegt, und andererseits durch einen selbstregulierenden Mechanismus, der anfänglichen Verzerrungen des Komitees entgegenwirkt.

Zweitens widmen wir uns dem Matching von Code-Fragmenten für die Deduplikation. Programmiercode ist nicht nur ein Werkzeug, sondern stellt selbst eine Resource dar, die der Wartung bedarf. Code-Duplikation ist ein häufig auftretendes Problem, das besonders im Zusammenhang moderner Entwicklungspraxis entsteht. Es gibt viele Gründe, Code-Duplikate aufzudecken und zu beheben; zum Beispiel das Bewahren einer sauberen und wartbaren Code-Basis. Für solche komplexeren Datenstrukturen wie Code sind String-Ähnlichkeitsmaße inadäquat. Stattdessen untersuchen wir einen modernen Ansatz des überwachten Metric-Learning, um Code-Ähnlichkeit mit Neuronalen Netzen zu modellieren. Ein Ergebnis ist, dass das Repräsentieren der elementaren Code-Tokens durch vortrainierte Embeddings die wichtigste Zutat in einem solchen Modell ist. Unsere Auswertung ergibt sowohl qualitativ, durch Visualisierung, dass thematische Verbundenheit gut durch diese Embeddings modelliert wird, und quantitativ, durch Ablation, dass die kodierte Information nützlich für das nachgelagerte Matching ist.

Als nicht-technischen Beitrag geben wir einen einheitlichen Zugang zu gemeinsamen Herausforderungen, die beim überwachten Lernen von Record Matching, Code Clone Detection und allgemeinen Metric-Learning-Anwendungen auftreten. Wir geben einen neuen Zugang zu String-Ähnlichkeitsmaßen vom Standpunkt der Wahrnehmungspsychologie, zeigen einen lange bestehenden Namenskonflikt von String-Ähnlichkeitsmaßen auf und dokumentieren ihn. Schließlich geben wir einen Überblick über die Schnittmenge der neuesten Forschung in Code Clone Detection mit dem Gebiet des Natural Language Processing.

Abstract

A quarter of a century after the world-wide web was born, we have grown accustomed to having easy access to a wealth of data sets and open-source software. The value of these resources is restricted if they are not properly integrated and maintained. A lot of this work boils down to matching; finding existing records about entities and enriching them with information from a new data source. In the realm of code this means integrating new code snippets into a code base while avoiding duplication.

In this thesis, we address two different such matching problems. First, we leverage the diverse and mature set of string similarity measures in an iterative semisupervised learning approach to string matching. It is designed to query a user to make a sequence of decisions on specific cases of string matching. We show that we can find almost optimal solutions after only a small amount of such input. The low labelling complexity of our algorithm is due to addressing the cold start problem that is inherent to Active Learning; by ranking queries by variance before the arrival of enough supervision information, and by a self-regulating mechanism that counteracts initial biases.

Second, we address the matching of code fragments for deduplication. Programming code is not only a tool, but also a resource that itself demands maintenance. Code duplication is a frequent problem arising especially from modern development practice. There are many reasons to detect and address code duplicates, for example to keep a clean and maintainable codebase. In such more complex data structures, string similarity measures are inadequate. In their stead, we study a modern supervised Metric Learning approach to model code similarity with Neural Networks. We find that in such a model representing the elementary tokens with a pretrained word embedding is the most important ingredient. Our results show both qualitatively (by visualization) that relatedness is modelled well by the embeddings and quantitatively (by ablation) that the encoded information is useful for the downstream matching task.

As a non-technical contribution, we unify the common challenges arising in supervised learning approaches to Record Matching, Code Clone Detection and generic Metric Learning tasks. We give a novel account to string similarity measures from a psychological standpoint and point out and document one longstanding naming conflict in string similarity measures. Finally, we point out the overlap of latest research in Code Clone Detection with the field of Natural Language Processing.

Acknowledgements

First and foremost, I would like to thank my advisor, Prof. Dr. Artur Andrzejak, of the Faculty of Mathematics and Computer Science at Heidelberg University, for his trust and support over the whole duration of my PhD studies. He encouraged me to explore and experiment and I learned a lot from him; not least about writing, teaching and organization. I will keep fond memories of the battle experience of last minute paper submissions and our fight for rooms, tutors and copies for the lectures and exams. I am also grateful to Prof. Dr. Holger Fröning, Prof. Dr. Michael Gertz and PD Dr. Wolfgang Merkle for serving on my thesis committee.

I thank the Heidelberg Graduate School of Mathematical and Computational Methods for the Sciences (HGS MathComp) for the financial support for travelling and the fun and educational annual colloquia. I am grateful to Prof. Dr. Filip Sadlo, who secured an extension of my funding. I would like to thank Anke Sopka and Catherine Proux-Wieland for their support in organizing my PhD studies and managing the teaching activities. Thanks also to all the competent tutors who I worked with over the years as a teaching assistant. And thanks to Rolf Bogus for the three years of tours through the Universitätsrechenzentrum with wonderful anecdotes for our lecture 'Betriebssysteme und Netzwerke'.

I had the pleasure of working alongside lovely colleagues in the *Parallel and Distributed Systems* group. Thanks to Felix Langner, Mohammadreza Ghanavati, Zhen Dong, Diego Elias Damasceno Costa, Kai Chen and Thomas Bach for making my time in the group enjoyable and for that sense of community spirit. I remember our discussions (Software Engineering and otherwise), being humbled in chess by Mohammad and barely learning the Chinese chess rules from Zhen, celebrating Christmas with Diego and Priscilla, fixing *hyper* and *turbo* together, and hanging out on the Neckarwiese.

The joint lunch break with the optimization and the databases groups, that survived the move to Mathematikon, was a constant source of joy, and replenishment of energy and motivation. Thanks for that to all these countless "Botanik philosopher's" over the years, from Christian Sengstock, Jannik Strötgen, Stefan Wiesberg, Achim Hildenbrandt, Hui Li, until Andreas Spitz, Erich Schubert and Sebastian Lackner, among others. The "Happy Hour" group is near to my heart and overlaps with this lunch group. It includes Victoria Ponce, Asha Roberts, Julia Jäger, Artsiom Sanakoyeu, Fereydoon Taheri. I am glad to call you my friends and hope we can maintain this friendship, even when some of us have gone or will go elsewhere in the world.

Bastian Rieck and Julia Portl, who I studied with, started their PhD studies before me and were always in a higher floor. That floor was always a good stop when I needed diversion, encouragement or concrete bureaucratic advice. I will always aspire to Bastian's discipline and technical literacy. Julia and the others of my oldest friend circle in Heidelberg helped me persist through the tougher times of my PhD studies. I was delighted when Martin Monath and my old work group moved right next door in Mathematikon. That allowed me to drop by more often, share a laugh, and get a glimpse of the cutting-edge of the research I got to know a little in my Diplom studies.

I want to thank my wonderful parents, Lieselotte and Thomas, and my brother Felix, for their support, help and advice over all these years.

Finally, I especially thank my wife Mangayarkarasi for her patience and loving support[30]. She does not know me without my PhD studies (and still stuck by me). I cannot imagine a life without her anymore, and I am looking forward to our life after the thesis.

[30]நான் உன்னை காதலிக்கிறேன்

Contents

1 Introduction

In this thesis, we study two different kinds of matching problems. For each, very different solution spaces are appropriate and we address the problems accordingly. In the first case we can make use of existing proven solutions and minimize the human input needed to pick the best one. This favours efficiency and simplicity. In the other, we use machine learning to fit a Neural Network model to the data. This takes care of the more complex data type.

1.1 Motivation

The advent of the internet and world-wide web marked the beginning of a new economy of data. Software and data sets could be exchanged without having to create and ship any physical copies. Open data, social media and the internet of things are sources for an ever-increasing torrent of data sets and free software.

Variety is one of the 'V's that characterize the concept of *Big Data*[1]. This does not only refer to the positive meaning (multi-modal, rich and diverse data) but also points to the fact that data is inconsistent, duplicated, non-documented, or incompatible in terms of schemas. Modern software development involves community-driven support and resources (help forums like StackOverflow and open-source software repositories like GitHub). This fuels fast-paced development and cooperation but arguably also worsens the phenomenon of code duplication.

So to reap the benefits of the modern connected world, data has to be connected and cleaned. Tasks like Record Matching, Similarity Join and Deduplication are therefore frequent and ubiquitous technical problems. They have yet to be solved by off-the-shelf tools and require bespoke solutions despite being mundane.

First, we address the most generic of such problems which appears as a subproblem in many others, String Matching. We do this by iteratively zeroing in on one of many existing similarity measures and simultaneously finding a good threshold. Both choices are hard to make even for an expert while a domain expert can easily label the few example cases we require. Second, we address Code Deduplication which represents the opposite case. A new model is required here because there is no out-of-the-box solution that reflects the data. We explore both problems and highlight the commonalities as well as the unique challenges and opportunities.

[1]Also regularly included are *Volume*, *Velocity*, *Veracity* and others.

1.2 Contributions

This thesis makes the following contributions:

String Matching without expert knowledge and requiring little effort

- We propose an Active Learning algorithm for picking the best string similarity measure for a String Matching task which also tunes the according decision threshold at the same time (Chapter 4). A committee of hypotheses queries the user to label instances of string pairs. A self-regulating mechanism counteracts the bias of the current committee. Finally, a simple stopping criterion terminates the loop. We evaluate this algorithm on 13 widely used string matching data sets and on 4 additional ones that we introduce. The results show that the algorithm terminates after only a small number of queries and finds settings close to optimal in the hypothesis space. This work is published in [BA15].
- We give a detailed account of an existing failure of nomenclature in the Computer Science community, where the same name 'Monge-Elkan' is used for two very different string similarity measures (Section 2.2.3). After outlining the history and status quo of this unfortunate circumstance, we offer some ways to deal with it in the future. We also categorize string similarity measures along psychological notions of similarity (Section 2.2.1).

Metric Learning for Code Clone Detection

- We use a Twin Neural Network to cast Code Clone Detection as a supervised Metric Learning problem (Chapter 5). We employ a Recursive Neural Network with a Long-Short-Term-Memory (LSTM) unit to aggregate the code fragments represented by Abstract Syntax Trees. We use self-supervised training to pretrain embeddings for node types and contents and show that these on their own represent a strong baseline when simply averaged. This work is published in [BA19].
- We present a discussion of Record Linking and Code Clone Detection in the context of supervised Metric Learning (Section 2.3). The type of binary relation associated with matching tasks yields a special label distribution. We discuss and address different problems arising from this.

1.3 Overview

In Chapter 2 we give an introduction to important concepts and existing work that pertain to both main chapters. We discuss our central concept of similarity in Section 2.1 where we take the psychological point of view because ultimately, humans define what they deem similar.

In Section 2.2 we give an overview over the topic of similarity measures and for which domains and data types they have been devised. Specifically, Section 2.2.1 presents the most important string similarity measures and organizes these along the psychological models of similarity we introduced in Section 2.1. Finally, we make a small detour to resolve a long-standing confusion about the two different similarity measures that go with the moniker 'Monge-Elkan' (Section 2.2.3).

Both main chapters present supervised learning approaches to arriving at a good similarity measure. Only the latter would be properly named *Metric Learning* but there are many insights that are important for both. We collect these in Section 2.3. All of the discussed issues there are related to the specific fact that Metric Learning uses *pairs* of instances and what that entails.

The Related Work in Section 3 is organized along the subsequent main chapters. The first three Subsections 3.1–3.3 are related to Chapter 4, while the last Subsection 3.4 relates mainly to Chapter 5.

Chapter 4 lays out our study of using Active Learning to pick a similarity measure and threshold for String Matching tasks. It follows the simple structure of providing some background (about Active Learning; Section 4.1), description of the method (Section 4.2), experimental setup and evaluation (Section 4.3) and discussion (Section 4.4).

Chapter 5 is about Metric Learning for Code Clone Detection and is structured very similarly. The background (about Code Clone Detection and representation of tokens; Section 5.1) is followed by the description of the approach (Section 5.2), experimental setup and evaluation (Section 5.3), discussion (Section 5.4). It also features an outlook (Section 5.5) outlining some directions in this quite active field of research.

We conclude this thesis in Chapter 6.

2 Background

In this chapter, we will introduce similarity as a fundamentally psychological notion, what problems this entails and how psychology has tried to address them. Further, we consider the problem of representation of abstract concepts by means of finite strings of discrete symbols. A special focus will be on how to capture their similarity through string similarity measures. And finally, we discuss common problems arising in supervised learning of binary relations.

2.1 Similarity

The notion of similarity is not easily defined. It is a subjective and psychological notion and depends on context. Similarity is a relationship between mental representations of objects, not the objects themselves per se. Humans do not find it difficult to judge (word) similarity without an external definition and show remarkably high reliability when doing so [MC91]. Psychology has tried for decades to model similarity relations and similarity scores reported by human subjects [Hah14].

One of the earliest models is the **spatial model**, which tries to capture the similarity based on human reporting of their perception with an implicit space of attributes of objects. The metric distance between vectors of attribute values of the respective objects corresponds to their perceived (dis-)similarity.

The spatial model interfaces very neatly with the theory of vector spaces and nice properties hold for the induced similarity relation. Many algorithms happily work with this tidy definition. Hence, this concept is easy to use by computer scientists and other quantitative scientists.

While the spatial model is nice and simple, it does not reflect some of the reality of similarity as perceived by humans. Tversky [Tve77] observed that when measured in human trials, similarity turns out to not be symmetric, and not obey transitivity.

Humans will more readily agree with the statement “a is like b” than with “b is like a”, when the latter object is more prototypical. For example. “An ellipse is like a circle” is more acceptable than “A circle is like an ellipse”. And “103 is virtually 100” will be more likely subscribed to than “100 is virtually 103”.

Further, transitivity breaks down since similarity depends on context. For example, a human might agree to “Jamaica is similar to Cuba” and “Cuba is similar to Russia” (the study was conducted in Soviet time) while denying that Jamaica was similar to

Russia. The reason is that the selection of the objects frames the comparison and makes different features salient. In the first case, the geographical proximity drives the similarity, in the latter it is about political similarity. But the independence of both allows the transitivity to break down in this case.

In the same study, the **contrast model** is proposed. It is defined via (binary) features of the objects under consideration. To determine similarity, the overlap of features, and the unique features to each object are combined. Each feature is weighted relative to its salience or importance. And the three feature sets (overlap, two sets of unique features) are further combined linearly. Since the unique feature sets can be differently weighted, the resulting relation $s(a, b)$ can be asymmetric. The so-called *focusing hypothesis* states that the perceived similarity is higher if the second argument's features are more salient (and the weight coefficients need to be adjusted accordingly). A human trial with country names was able to confirm this [Tve77].

The same paper also investigates the difference between asking for similarity vs asking for difference. To do this, the author sets asymmetry aside by tying the weights of the distinctive features to make the overall relation symmetric. These two modes are virtually equivalent in terms of correlation but differ in how the weights have to be set. When asked for similarity humans put more emphasis on common features while they focus more on distinctive features when asked for difference.

These two models—the spatial and the contrast model—represent objects merely by real or binary features. Much of the semantics of an object is lost if its features are fixed but all relations between them are discarded or shuffled. Much like how a face drawn by Picasso in his later works might have all facial features but barely resemble a real human face, because the spatial relations between the features have been scrambled. A more expressive representation of objects not only records their features, but also relations between them. The model of **structural alignment** consequently measures the degree to which these structural representations of two objects can be aligned. A perfect alignment is essentially an isomorphism, while for less similar pairs of objects not all features or relations between features can be aligned perfectly.

A newer model for the human notion of similarity is centered on **transformations**. Its central claim is that similarity is perceived relative to how many steps of basic transformations lie between two object representations. For example, a horse (visually) becomes a unicorn if you add a horn. This model of course relies on a set of permissible transformations. The shortest sequence of transformations (possibly with different weights) capable of transforming one object into the other then defines the distance between those two objects. In computer science, this framework is well known and the

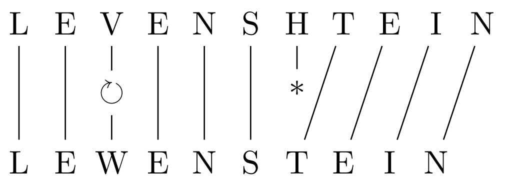


Figure 2.1: Levenshtein edit distance

edit distance (Levenshtein) measures similarity of sequences of symbols. It considers as permissible (local) transformations the deletion (or insertion) of one character, the exchange of one character for another, and the swapping of neighboring characters.

The transformation model can be seen as generalization of the feature-based models (spatial and contrast), as one can transform the values of each feature in one step. It also generalizes the structural alignment model as it can transform corresponding features into one another, and delete or create features without correspondence.

2.1.1 Representation

In the above discussion, all objects were stimuli in human studies. These include sentences, words, numbers, diagrams, pictures or names of entities, such as countries. In each case, they are either everyday entities (since every participant has to know them) or abstract symbols or stimuli that elicit a certain psychological reaction.

In more technical domains, where the help of computers is desired because the size, number or abstractness of data calls for machine support, the objects under consideration are often of a different nature. They may be biological sequences like DNA or RNA – too long for a human to handle, yet simple and unambiguous. Or they are sound files: divorced from any human interpretation, they become mathematical objects – waves that have been sampled at regular intervals. Also big relational data, like social graphs, are in principle of interest for a human. But they break any individual human's capacity to understand them very fast, or even hold and manipulate them in their mind. But to a computer, they are a simple and unambiguous data structure – a graph. Similarly, strings of tokens in a formal language are simple, but more accessible to a computer than a human.

Of course unambiguous data like this can still be ambiguous for a human. A sound wave famously can be understood as both "Laurel" or "Yanny" by English speakers.

But while this data at least has an unambiguous abstract meaning, its representation is seldomly unique. It is often hard to come up with a scheme to encode abstract objects of a certain class into strings of tokens such that each object can be represented by exactly one string.

Real numbers, for example, escape any attempt of representation by finite strings, since their cardinality is greater than the set of finite strings over a finite alphabet. Conversely, if one picks a base b, the positional notation of real numbers (e.g., decimal expansion for $b = 10$) will yield a periodic expansion as well as a simple terminating expansion (e.g., $0.\overline{9} = 1.0$). For rational numbers, if constructed as fractions of integers, the problem of representations is even more prevalent: Every rational number has infinitely many equivalent representations, e.g., $\frac{4}{5} = \frac{8}{10} = \frac{-44}{-55} = \frac{52}{65} = \ldots$.

These representations form equivalence classes: Each representation uniquely refers to one abstract object and every abstract object may have a (finite or infinite) set of representations referring to it.

For many cases, these representations can be normalized by algorithmically determining a unique canonical representative for each equivalence class. For example, one can define a unique canonical representation for each real number with a finitely describable (periodic or finite) decimal expansion. One can algorithmically determine the unique representation for a rational number such that the denominator of that representation is the smallest positive integer among all representations for that number.

In other cases, there is no canonical representation that can be arrived at algorithmically (or none has been found yet), such as for 2D projections of (mathematical) knots. However, it has been proven that all representations of any given knot are in the same orbit with respect to a small set of transformations (Reidemeister moves).

And then there are practical obstacles to arriving at canonical representations, even if they exist. For example, propositional logic formulae can be used to represent Boolean functions. They can be effectively mapped to a canonical form (e.g., full Disjunctive Normal Form), but it is an NP-hard problem. Under some conditions, there are obvious “brute force” ways of arriving at a normal form. For example, a graph as represented by its adjacency matrix can be easily serialized. The column and row indices correspond to an (arbitrary) numbering of its nodes. This means that equivalent representations must be matrices of equal size. A canonical form may then be defined by the lexicographically smallest matrix that represents a graph that is isomorphic to the original. However, the graph canonization problem is known to be NP-hard as well. And that fact does not depend on whether one defines the canonical form in any other way.

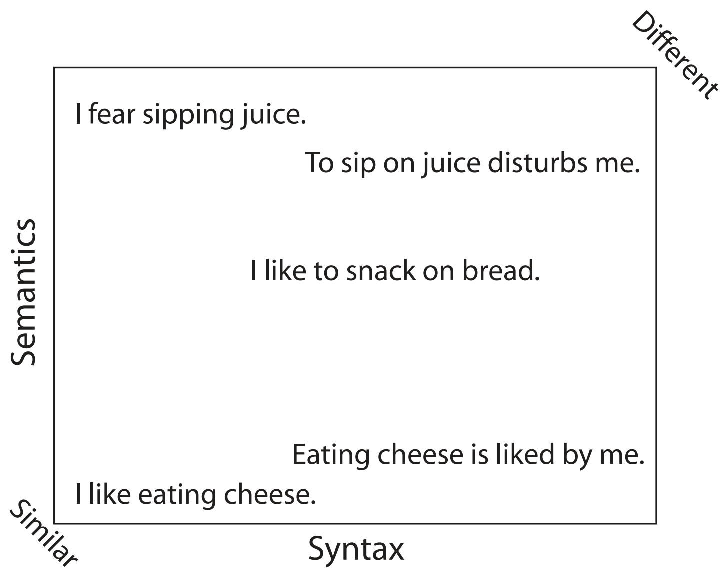

Different
I fear sipping juice.
To sip on juice disturbs me.
I like to snack on bread.
Eating cheese is liked by me.
I like eating cheese.
Semantics
Similar
Syntax

Figure 2.2: Syntactical vs semantic similarity; figure from [EJ07]

2.1.2 Syntax vs semantics

Simple objects such as geometric shapes or images depicting simple scenes are used in psychological research about similarity as simple stimuli that are largely unambiguous. Language, on the other hand, is inherently complex. It links arbitrary sounds or symbols to abstract concepts or concrete things[1]. Its design makes it possible to flip meaning by just interjecting one syllable, or a few characters (e.g., “not”), into a long sentence. So the elements of a sentence interact to create meaning. But elements, like words, do not always come with a fixed isolated meaning. The task of determining the sense that a given word has in a given context is called *word sense disambiguation*. Further, natural language really only exists in its speakers’ minds and evades formal definition. This is a fact that fuels a whole industry of researchers and engineers in the field of Natural Language Processing/Understanding.

Language artifacts can have superficial similarity (syntax), whereas their semantics are very dissimilar (and the other way around). Figure 2.2 illustrates this. Words that share a lot of their contextual usage are likely to be semantically similar [MC91]. This fact has been used in modern efforts mapping natural language words to vectors that reflect their similarity (see Section 5.1.4). However, if words have polar opposite

[1] Chinese characters may sometimes resemble concrete things in the world, and words may sometimes sound like the concrete things they denote (*onomatopoeia*). But such close relation between a symbol and what it signifies are the exception in languages, not the rule.

meaning, it is likely that they share a lot of their contexts, too (antonyms, e.g., “weak” and “strong”) [MC91].

2.2 Similarity measures

Similarity measures are concrete implementations reflecting certain kinds of variation that (in a given domain) reflect change in semantics via change in syntax. Many different fields have come up with their own similarity measures.

Similarity measures serve to solve concrete practical applications. For example, suppose one wants to determine the identity of entities by some set of observable features like faces, fingerprints or irises. In other circumstances, one is not interested in identity, but similarity measures are used to characterize groups of individuals. It enables clustering instances of observations, or matching instances against prototypical specimen (e.g., with k-Nearest Neighbours classification). Downstream applications can be exploratory studies of the emergent clusters or removal of the variation, in other words, deduplication.

The concept of similarity measures comes with a few axioms that generally but not always hold. A similarity measure is supposed to be symmetric in its two arguments, and take the maximal value only for identical arguments. Often, values are normalized to the interval $[0, 1]$. Distance metrics are similarly axiomatically defined (*non-negativity*, *identity of indiscernibles*, *symmetry*, and *triangle inequality*). And by mapping the values of a distance metric in certain ways, one can get a similarity measure. The details of this mapping are not as important as the fact that it should invert the resulting ordering of pairs. Because of this duality, whenever we talk about a distance in the context of similarity measures, it is understood that we mean the associated similarity measure to that distance.

2.2.1 String similarity measures

Different kinds of data types have their own kinds of similarity measures. One of the simplest non-trivial data types pervasive in almost every field are strings of discrete symbols. Apart from strings from the everyday human experience, such as names, dates, addresses and text documents, there are more technical kinds of strings. Computational Biology has emerged as a big field that deals with naturally occurring strings of discrete symbols, like the base pairs in strings (or loops) of long organic molecules like DNA. Richard Dawkins [Daw07] noted:

> Since Watson and Crick in 1953, biology has become a sort of branch of computer science. I mean, genes are just long computer tapes, and they use a code which is just another kind of computer code. It's quaternary rather than binary, but it's read in a sequential way just like a computer tape. It's transcribed. It's copied and pasted. All the familiar metaphors from computer science fit.

Since strings are such a general concept, many different fields encountered the need to measure similarity between instances of domain-specific strings.

A broad historic survey focusing on edit distances can be found in [Nav01]. An empirical study evaluating a big set of string similarity measures in name matching tasks has been conducted in [CRF03]. And [Chr12] introduces string similarity measures in the context of general Data Matching pipelines.

Here, we want to give an account of the most prominent metrics in terms of the models of similarity motivated by psychology.

Spatial model

The spatial model supposes that objects or stimuli (here: strings) are first mapped into a numeric feature space, before being compared via Euclidean or cosine similarity. The most prominent traditional example is called TF-IDF. This approach reserves one dimension of the representation vector for each token of a fixed vocabulary. The given string of tokens is then represented by one real value per token of this vocabulary. This value is the relative frequency in the token string (term frequency, TF), normalized by the base frequency of this token in a background corpus (inverse document frequency, IDF). In [CRF03], the authors introduce a relaxation of TF-IDF, where tokens are matched not by identity but by a secondary similarity measure for the tokens (Soft-TF-IDF).

A newer development leading to a numeric vector representation is where (Recurrent) Neural Networks read input strings of tokens, which then can be represented by the final hidden state of the network, which is trained on a (supervised or unsupervised) objective. These vector representations are often called *embeddings* (see Section 5.1.4). By comparing these vectors with Euclidean or cosine distance, one effectively has a (domain-specific, because trained) string similarity measure. Related work about other learnable string similarity measures is presented in Section 3.1.

Contrast model

The contrast model relies on the definition of binary features and a way of quantifying overlap between the feature sets of two inputs. For string similarity measures, a very useful feature is the presence or frequency of n-grams. These sets of features for both inputs are then compared by Overlap, Jaccard or Dice coefficient. N-grams are sometimes generalized to n-grams of strings with padding, n-grams encoding also the position, or skip-grams, that effectively relax identity. Again, like in Soft-TF-IDF, a relaxation of identity of tokens to mere similarity as measured by a secondary similarity measure, can be applied (e.g., Extended Jaccard [NH10]).

Structural alignment

The structural alignment model assumes that similarity is perceived as a measure of how well parts of both instances can be matched and compared. These following similarity measures work well with strings that naturally come as concatenation of substrings that can occur in arbitrary ordering, for example, identifiers that are composed of names and other characteristic features (like addresses).

The *Longest Common Substring* similarity measure [FS92] creates a matching by repeatedly identifying the longest common substring. This process ends once there is no common substring of a given minimal length. The lengths of these substrings are added up and normalized by the lengths of the two string instances. Because the iterative identification of substrings is greedy, it might not find a globally optimal matching, but can be implemented by a dynamic programming paradigm. Furthermore, this measure is only symmetric, if one imposes an additional regularization (like requiring the matching to be globally optimal).

Another string similarity measure based on structural alignment is the hybrid metric proposed by Monge and Elkan [ME95]. It assumes that strings are tokenized into substrings and that there is a secondary similarity measure that measures similarity of these tokens. It then looks at the bipartite weighted graph made up by the two sets of tokens from the two input strings. The weights on the edges are the similarity value as measured by the secondary similarity measure. It then finds for each token in the first token set the most similar token in the opposite token set. The Monge-Elkan similarity is then defined as the average over these similarities. Since this definition hinges on which token set is the first one, this measure is not symmetric. However, this could be amended if one instead averages similarities over the optimal *matching* in this weighted bipartite graph. It has found wide-spread adoption specifically in the iteration ‘level 2’.

For more on this metric and how its name is mistakenly used for another metric, see Section 2.2.3.

TagLink [CS06] is also a hybrid metric, that is, it aggregates the similarity of tokens of two tokenized strings. While the hybrid metric of Monge and Elkan (often simply dubbed ‘Level2’) focuses on one string and finds matches for its strings, TagLink defines the matching of tokens as a general matching (where the tokens are nodes of a bipartite weighted graph).

Translation metrics The idea of matching tokens and aligning n-grams to measure string similarity has special importance in the field of machine translation. It is hard to automatically compare the output of a machine translation system against reference translations. The NLP community has come up with different metrics to do this. BLEU [PRWZ02] measures a modified n-gram precision which caps the repeated matching of tokens to the reference tokens to a maximum. There is also a penalty on short outputs which otherwise easily reach high precision by guessing the single-most obvious token in the target translation.

ROUGE [LO04] denotes a family of similar metrics that try to address shortcomings of BLEU. Instead of penalizing short output, they measure recall of matches. By either requiring or favouring in-order and consecutive matches, they also naturally favour closer alignments of syntactic structures between output and reference strings. These notions of precision and recall are combined as in F-score. METEOR [LSJ04] is defined very similarly, but aggregates to a F_3 score which is a variant of F_1 with 3 times as much weight to precision over recall (see Section 2.3.3). METEOR further drops the requirement of exact token matches. It instead allows equivalent tokens after stemming and synonym matches. Stemming is the operation that maps words to their word stem which gets rid of inflections. For example, “walking” and “walked” would both map to the stem “walk”.

This last innovation is taken to a higher level by Bertscore [ZKW+19]. It also calculates the F_1 score of token matches but the matches have certain weights. First of all, tokens are weighted by an IDF-weight, which makes it less important to find an equivalent for words like “the” but infrequent words, especially content words like nouns and adjectives, are important to match with a good equivalent. Another weight expresses the similarity of the token in the context of their sentence. This is achieved by feeding each sentence into BERT which is a general language model. It has been trained to predict missing pieces of text and thereby has learned to represent tokens with highly informative vectors. In this sense, these language models are an extension of the idea of (non-contextual) word embeddings (see Section 5.1.4). The contextual nature of the BERT vectors can also capture aspects like the sense of an ambiguous term or the meaning of a turn of phrase.

All these automatic metrics cannot replace human judgement of a good translation or summary. However they achieve high correlation with the ground-truth and can be evaluated cheaply.

Transformation model

This model defines a set of transformation rules along with a weight that determines how costly each transformation is. The lowest cost that allows for a transformation of the first input into the second input then defines the distance. This is called *edit distance* in the context of strings. The Levenshtein distance is defined by allowing deletion, insertion and replacing of single tokens, where each such transformation step has uniform cost [Lev66].

Other distances are variations of this. The Damerau-Levenshtein distance additionally allows and accounts for swapping of adjacent tokens [Dam64]. The Needleman-Wunsch distance [NW70], which is motivated by DNA and protein sequences, allows for gaps. The Smith-Waterman distance [SW81] is a variant of Needleman-Wunsch, which first aligns substrings according to an alignment score, before measuring the edit distance.

Another variation on the edit distance concept, Editex, introduces the organization of letters of the English language to equivalence classes. For example, *v* and *f* may be considered equivalent. Replacing letters across equivalence classes then incurs higher cost as compared to replacements within a class.

A hybrid model

Based on the above ideas, the US census bureau devised a domain-specific string similarity measure for the special application of matching personal names. It is called Jaro-Winkler similarity measure and combines n-gram based similarity with edit distance in an elaborate way optimized for its purpose. The basic Jaro algorithm [Jar89] aggregates both the number of common characters within the first half of the longer string with the number of transpositions to map sets of common substrings.

Several modifications by Winkler account for experiences gained in the work at the census bureau [Win90]. They discount differences according to how long the common prefix is relative to the total amount of common characters and treat small equivalence classes of similar-sounding characters.

***The* Similarity Metric**

It has been shown that there exists a string similarity measure that in a sense generalizes any practical string similarity measure. It is derived from what is called the *normalized information distance* [LCL$^+$04]. This distance is normalized for length differences of its two arguments and is uniformly smaller or equal to any effectively implementable distance (up to a constant). Because of this generality the associated similarity measure is simply called 'The Similarity Metric'. It is not a computable function but simply a theoretic concept based on Kolmogorov complexity. If, however, one replaces the Kolmogorov complexity by any computable compression function, one gets a practical so-called *compression distance*.

2.2.2 Other similarity measures

Another very versatile data type are graphs and networks. Trees, as a special case of graphs, can represent hierarchies, which, since it is a very general concept, are ubiquitous. Graphs and networks can represent social relations or chemical reactions. Any binary relation will map to a graph, which is why these structures are found in every field.

Probability distributions can be compared by Kullback-Leibler divergence. This is not a real distance since it is not a symmetric relation. Furthermore, there is the Wasserstein metric (with the so-called Earth mover's distance as a special case) to compare probability distributions.

Another frequent data type are scorings and rankings, because these functions operate on any given data instance. As scorings they are commonly compared by Pearson's correlation correlating the scores themselves. When just correlating the rank indices, Spearman's correlation or Kendall's Tau can be used. Similarly, clusterings are just discrete-valued functions operating on data instances implying equivalence classes. These can be compared by the Rand index or mutual information.

An important family of similarity measures are quality metrics (Section 2.3.3). They measure the similarity of a target function which one aims to model and the function instantiated by a model. They also appear in a more instrumental way as so-called *loss functions*. In that context, they are semantically distance metrics. It is usually important that the loss function is differentiable with respect to the model parameters, to enable optimization strategies like gradient descent. The family of loss functions includes cross entropy and hinge loss.

Another application around Neural Networks is to measure similarity of activations of sets of neurons (e.g., a specific hidden layer) in two Neural Networks on a given set of inputs. The Neural Networks do not have to be the same. These activations can be compared, even absent a direct mapping of neurons, by Canonical Correlation Analysis (CCA, [MRB18]). This maximizes correlation scores of affine linear mappings from one set of neurons onto the other. Centered Kernel Alignment (CKA, [KNLH19]) is a related statistic about network activation patterns that has been shown to even better recover existing correspondences.

Apart from these data types that are functions operating on data types, there are many domain-specific data types where research communities come up with specialized similarity measures to compare instances by. These include trajectories, spatio-temporal data (events). And finally, some similarity measures cross distributions or even modalities, for example, comparing queries to documents or persons to organizations.

2.2.3 Confusion around 'Monge-Elkan'

In the following, we will discuss the confusion around the term 'Monge-Elkan metric/distance'. It is used in two broad but clearly different senses and to the best of our knowledge this fact has not been addressed in the literature. Finally, we illustrate how this has led to undesirable outcomes and provide suggestions how to resolve the problem. This is a conclusive account of a failure of the organic way of naming concepts in academia. It is unclear, if (and to what degree) anyone is to be blamed, or if this bad outcome can emerge despite reasonable levels of care. We want to include it as a general cautionary tale and as an attempt to disentangle the concrete confusion around the particular term.

Historic background

In 1995, Monge and Elkan [ME95] study matching strings representing entire database records. One central idea was to break up these strings into their fields, delimited by specific characters. Another idea was accounting for the frequent phenomenon of abbreviating parts of names. These two ideas would be developed in several publications in the subsequent years.

In [ME95], the first idea was realized by a recursive matching algorithm. It defines a matching score of whole text fields as the average of maximal matchings of subfields. Their match score is determined by recursively breaking the subfields up into subsubfields and again averaging the maximal matchings of subsubfields. Each input string comes with an additional input list of delimiters that defines the sub- and subsubfields. The base case deals with primitive values that will not be broken up further. Their match scores are binary, 1 or 0. A match score of 1 is given when string A (without punctuation) is a prefix of string B. They outlined three other, more complex rules that matched abbreviations (like “Dept.” with “Department” or “Caltech” with “California Institute of Technology”), but they were not yet implemented. This was the first attempt at addressing the second idea of dealing with abbreviations. The entire algorithm is concretely given in pseudocode in this paper. It provided an overall scheme that was capable of matching whole records, serialized as strings, with the capability of accounting for out-of-order fields and patterns of abbreviations.

In [ME96], Monge and Elkan study this recursive algorithm alongside two other algorithms for field matching. Here, they only give an abridged description of the recursive matching algorithm which unfortunately does not specify how the hierarchy of tokenization is defined. Furthermore, they introduce Smith-Waterman as a field matching algorithm. The authors optimize the weights and character equivalence classes to work better with text-based data, rather than biological sequences. They point out that this algorithm does address variations like abbreviations (idea 2), but not out-of-order subfields. In addition, a “base field matching algorithm” which measures overlap of tokens in tokenized strings (without stop words) with the dice coefficient is defined. Later, Monge and Elkan [ME97] study the potential of their variant of Smith-Waterman for capturing abbreviations with local edits instead of complex formalized rules addressing specific abbreviation patterns in a paper of its own.

In his dissertation thesis [Mon97], Monge devotes one chapter to domain-independent record matching where for the first time the recursive algorithm and the Smith-Waterman edit distance are studied combined in an explicit hybrid version (Smith-Waterman fulfilling the role of scoring the base case matches).

Some years later, the first publication uses the term “Monger-Elkan distance function” (sic!) [CRF03]. The authors refer with this term to the variant of the Smith-Waterman edit distance, citing [ME96]. At the same time, they address the “recursive matching scheme” as introduced in [ME96], and state; “following Monge and Elkan, we call this a level two distance function”. Of course, it had already been introduced and explained in much greater detail in [ME95]. Moreover, the expression “level 2” cannot be found in [ME96] (it only mentions “lowest level”).

This terminology is instead introduced in [Mon97] which explicitly talks about “nesting levels”. Here, level “L=0” means applying Smith-Waterman directly to the inputs, level “L=1” means breaking up the record into fields and matching the fields by their edit distance scores (“subrecord-level Smith-Waterman”) and level “L=2” means breaking the fields further into words before matching those with (“word-level Smith-Waterman”).

The authors implemented a software library [CRFR03] for the evaluation in [CRF03].

A more appropriate reference in [CRF03] would have been [ME95], which details the recursive algorithm with pseudocode. Instead, [ME96] is given as a reference, which itself does not even reference [ME95]. Despite being the original and more extensive publication of this algorithm, there are only a dozen citations for [ME95][2]. The implementation [CRFR03] gives an unambiguous definition of "Monge-Elkan distance" as the modified Smith-Waterman edit distance by naming it in the source code, while calling the Level 2 instantiation of the recursive algorithm simply "Level2".

However, the vague reference in [CRF03] to [ME96] which shows both the recursive algorithm and the Smith-Waterman variant side-by-side might have caused some confusion down the line. The fact that the distance coined as 'Monge-Elkan' is a variant of the already named 'Smith-Waterman' might have masked its role in [ME96], as well.

Subsequently, in 2004, a study about Information Integration for the Semantic Web [CNC05] came with an open-source implementation of similarity measures [CSC04]. Here, unfortunately, 'Monge-Elkan' was the name given to the recursive algorithm, instead of the variant of the Smith-Waterman edit distance. The base case is handled here per default by Smith-Waterman-Gotoh, which is a computationally more efficient approximation of Smith-Waterman, but can also be overridden with another metric.

Later, there were two influential publications using the moniker 'Monge-Elkan' in the original sense of the edit distance [EIV07, KR10], but also some (comparatively less-cited) publications used it in the recursive sense [PS07, MYC08, JBGG09]. In both [PS07, JBGG09], several different secondary similarity measures were compared, that calculated the base case of the recursive algorithm. One widely-cited publication [BE08] did not specify the operative meaning of 'Monge-Elkan' at all.

Then, between 2010 and 2012, three text books about duplicate detection [NH10], data matching [Chr12] and data integration [DHI12] were published. They all use the non-original meaning of 'Monge-Elkan' (recursive function).

Status quo

Overall, the edit distance sense has more de facto relevance, having over 6300 citations in influential publications[3] that refer to this sense by some proper noun (e.g., 'Monge-Elkan similarity measure', 'Monge-Elkan metric', or simply 'Monge-Elkan'/'MongeElkan'): [CRF03, BMC$^+$03, SSK05, EIV07, BE08, KR10, RRV13, CH13]. It is implemented in two important academic software implementations – SecondString [CRFR03] and

[2]as of February 2020

[3]more than 100 citations as per Google Scholar in January 2020

DKPro [BZG13]. The term ‘Monge-Elkan distance function’ was coined in [CRF03] and unambiguously named and implemented in [CRFR03]. The follow-up paper [ME97] to [ME96] exclusively studies the Smith-Waterman variant as the only similarity measure.

On the other hand, the hybrid metric sense is referred to with ‘Monge-Elkan’ in a proper name in widely-cited publications that accumulate to 2200 citations in total ([MYC08, NH10, DAC10, Chr12, DHI12, GDD$^+$14]). Among these are three text books, which is the most important source of authority in this camp. Furthermore, apart from [CSC04] there are at least four other open-source software implementations[4].

Despite the imbalance in terms of reach by citations, the confusion is far from settled. There are still papers being published referring to ‘Monge-Elkan’ distance or (similarity) metric/measure in either sense (Smith-Waterman variant [ZGH$^+$18] or the hybrid metric [PWH18, SMFM18]). Which of the two broad meanings are used is sometimes implicit (e.g., “character-based”, “edit distance” [ZGH$^+$18]). A very good way of delineating both variants carefully is exhibited in [SMW15]. Maybe it is because the authors use the SecondString implementation [CRFR03] that clearly documents the differences in programming code.

Bad consequences

Inconsistent naming like this inevitably leads to bad outcomes. The worst consequence is that the body of research does not chrystalize to knowledge, because we are comparing “apples with oranges”. The ambiguity is likely not even discovered by most researchers, who may assume there was only one definition and they would likely go with the first or most authoritative definition they encounter. This is exemplified by [JBGG09][5], which introduces ‘Monge-Elkan’ in the sense of the hybrid metric and goes on to generalize it. At the same time, it refers to four existing papers [BM03a, CRF03, Chr06, PS07] as references for ‘Monge-Elkan’, only one of which [PS07] uses it in the same sense of the hybrid/recursive algorithm.

We fell into the trap ourselves in [BA15], assuming there was only one meaning (Level2), while using the implementation from [CRFR03] as a black box (which is Smith-Waterman). In [AX06] the exact opposite happened: The authors refer to Monge-Elkan distance in the sense of Smith-Waterman while giving [CSC04] as the only reference which implements Monge-Elkan in the Level2 hybrid metric sense. And there

[4]`https://github.com/chrislit/abydos`
`https://github.com/mpkorstanje/simmetrics`
`https://github.com/anhaidgroup/py_stringmatching` (project Magellan [KDSG$^+$16])
`https://github.com/life4/textdistance`

[5]that has ‘Mongue-Elkan’ right in the title – including that typo

is similar confusion in the paper [dPAEG15], calling Monge-Elkan an edit distance (implying the Smith-Waterman variant), while also explicitly and formally defining it as the Level2 hybrid method.

Conclusion

Usually, when a method, algorithm or such is named after its author(s), it is done by a third party. The third party usually informally refers to it with such a name, and this nomenclature picks up traction. In particular, this is outside the responsibility of the original authors of the artifact themselves. In our particular circumstances, the original sources for the algorithms, and the references to them, were sufficiently vague and ambiguous that the mapping of name to algorithm was not unequivocal. The paper [CRF03] and the accompanying implementation [CRFR03] that named the two algorithms actually did a good job on that but later publications did not follow suit.

Clearly, both similarity measures have been valuable innovations and continue to play important roles. Going forward, it would be good to not add to the existing confusion between them. To this end, we want to discuss the different options.

For backwards-compatibility, names used in the future should ideally call attention to the fact that *there has been* confusion around ‘Monge-Elkan algorithm/similarity measure/metric’. This would be ideally engineered when readers familiar with any one of the *new* terms would have to stop and wonder when encountering one of the existing ambiguous terms like ‘Monge-Elkan metric’, or vice versa. To give an analogy: when specifying either one of the two meanings of the word “billion”, one can use an additional attribute, or the scientific notation: short-scale (10^9) or long-scale (10^{12}). The notation with the attribute is preferable, because it calls attention to the existence of a difference in the first place.

The variant of the Smith-Waterman algorithm could be referred to as ‘Monge-Elkan edit distance’. That would always clearly associate it with Smith-Waterman and call attention in situations a reader has only encountered the recursive algorithm in the context of ‘Monge-Elkan’. Another good title would be “Smith-Waterman with Monge-Elkan weights”[6]. This also clearly refers to one rather than the other algorithm while also distinguishing the original algorithm from the innovation of natural language specific weights.

The recursive, hybrid algorithm should ideally always go with the “hybrid” attribute, to make it understood that it is only fully qualified with a secondary similarity measure.

[6]This is similar to a formulation used in [Chr12] in the section about Smith-Waterman.

A reader would stumble over this fact when they were understanding 'Monge-Elkan' to be the Smith-Waterman algorithm, which does not require a secondary similarity measure. A good, albeit long name could be, e.g., "Monge-Elkan hybrid metric with Levenshtein as secondary similarity measure". Alternatively, "Monge-Elkan hybrid metric with word-level Levenshtein". Two good examples are [SMFM18] ("the Monge-Elkan 2-level algorithm", and later "leveraging Jaro-Winkler as an internal measure") and [PWH18] ("Level2 method proposed by Monge and Elkan").

Authors that are aware of the confusion could also point it out directly, for example, "by 'Monge-Elkan' metric, we specifically refer to the hybrid algorithm, as opposed to the variant of Smith-Waterman".

Computer science as a field has problems with reproducibility, even if all experiments live in silico [Pen11]. The problem may not be as big as the replication crisis in psychology [OSC15], but more avoidable. Peng [Pen11] states: "Perhaps the biggest barrier to reproducible research is the lack of a deeply-ingrained culture that simply requires reproducibility for all scientific claims." Ironically, the terminology around this very topic is fraught with conflict in nomenclature. In [Ple18], the various definitions around 'reproducibility', 'replicability' and 'repeatability' and their histories are outlined.

2.3 Metric Learning

The goal of Metric Learning is to approximate a function that measures similarity (or distance) according to an underlying notion of similarity.

A basic form of Metric Learning involves learning a linear transformation on the instances and comparing the representation vectors by the Euclidean distance. The result is a generalized Mahalanobis distance. The survey [Kul13] gives a good overview of this framework. Metric Learning can be done with different amounts of supervision. Good overviews of unsupervised, semi-supervised and supervised learning approaches with linear or non-linear transformation functions are given in [BHS13, WS15]. There are different paradigms to define the loss based on the supervision signal in supervised learning. The overarching idea is always to make matching pairs of instances more similar over time while making non-matching pairs less similar. A comprehensive survey can be found in [KB19].

The concepts discussed above and the pointers are mostly relevant only for Chapter 5 which is about learning similarity of code fragments. But there are several important aspects that equally apply to Chapter 4 which is about Active Learning of string similarity for String Matching.

These common challenges arise from a crucial difference to standard forms of supervised machine learning. The most common case of supervised machine learning is to approximate or learn a target function $f : D \to C = \{0, \dots, n\}$ (i.e., classification) or $f : D \to \mathbb{R}$ (i.e., regression). In our setting, the domain of the target function is a Cartesian product $D \times E$. A target matching function would have the signature $f : D \times E \to C = \{\text{match}, \text{non-match}\}$, with deduplication as the special case of $D = E$. This kind of supervision signal is also used in so-called *Constrained Clustering* [BDW08], which uses weak supervision in form of *must-link* and *cannot-link* pairs. The associated regression problem $f : D \times E \to \mathbb{R}$ is the target similarity metric. The central consequence is that supervised data is not arbitrary, but necessarily inter-dependent. In the following sections, we will look at resulting challenges.

2.3.1 Label distribution

For the following consideration, we lay down the following definitions: $n := |D|$ and $m := |E|$. Without loss of generality, $n > m$.

In matching, the positive class of matches can maximally be of size m, whereas the negative class contains at least $(n-1) \cdot m$ tuples. So, the matches can only grow linearly while the non-matches will grow quadratically. In clustering, the positive class is determined by the biggest cluster. If its size is c, the positive class can be maximally $\frac{n}{c} \cdot \frac{c \cdot (c-1)}{2} = \frac{n \cdot (c-1)}{2}$ while the negative class has at least $\frac{n \cdot (n-1)}{2} - \frac{n \cdot (c-1)}{2} = \frac{n \cdot (n-c)}{2}$ many elements. Since usually $c \ll n$, the label distribution for matching is similarly skewed, even if generally less so. More specifically, with the assumption that cluster sizes will not pass a certain size threshold, the asymptotic growth is linear versus quadratical, just as in matching. Tables 4.3 and 5.2 exemplify this with the data sets of our experiments.

This simple combinatoric fact has several important implications that we are going to address in the following. Learning under heavily-skewed label distributions is generally referred to as the *class imbalance problem*. Good references for general supervised learning with imbalanced data are [BTR16, Wei04]. For the specific challenges that arise for Active Learning, refer to Chapter 7.4 in [Set12] or Section 4 in [AP11].

Blocking / Indexing

First of all, this class imbalance makes the application of any comparison function on all pairs inefficient. It is clear that most pairs will not be matches, but finding out which ones are (naïvely) requires quadratically many evaluations. Depending on the domain, one can find simple necessary conditions for a pair to be a match. For example, in the matching of records representing persons, it might hold true that both data sources have an up-to-date and complete field about the ZIP code the person is living in. The records can be *indexed*/sorted by this key and comparisons are needed only within the *blocks* of records with identical ZIP codes.

This idea can be relaxed to any preliminary classification of matches with (near) perfect recall and with a not too terrible level of precision. This classification is not perfect, but because of its perfect recall it can be used as a necessary condition (or filter) for further comparisons. See [Chr12] for an overview of blocking methods in Record Matching. These more general methods are usually still referred to as *blocking*, even if they do not arrange instances in blocked lists.

Blocking schemes can be devised statically by a domain expert, but can also be learned [MK06]. In [SW18], a blocking scheme is learned via Active Learning.

Blocking does not only increase efficiency during inference time of a similarity model. It also affects all parts of the learning pipeline. It can make the task of labelling less cumbersome, models will learn more effectively and it makes the evaluation of the query strategy in Active Learning less expensive.

Sampling

Unlike in the evaluation of a model to identify all matches or clusters, the training of such a model does not rely on the full data set – not even all of the positive pairs. Because good models are capable of generalization, sampling from the data may be appropriate. Note that depending on the particular loss the model is trained with, the amount of supervised data relative to the number of instances can be even more extreme (e.g., cubic for triplet loss). Informative sample selection is important in metric learning [KB19], because it can speed up the learning and ensure better generalization.

Class imbalance is a problem that often arises, also outside of Metric Learning [BTR16, KKP06, Wei04]. Applications like anomaly detection, intrusion detection or medical screening often deal with small positive classes. Undersampling is the technique of suppressing the prevalence of the majority class. It can be done entirely randomly or with a specific strategy. Oversampling, on the other hand, artificially increases the minority class, for example by synthesizing artificial instances (SMOTE [CBHK02]). For more involved sampling strategies, refer to [Wei04, BTR16].

Not only the ratio of positive versus negative labels matters, but also their distribution. Too easily classified samples eventually do not contribute to the learning progress in Metric Learning [KB19]. Techniques like negative sample mining specifically seek out samples for which the current prediction is poor. At the extreme end of curating the training data is Active Learning where one incrementally builds up the training data to avoid the vast amount of labelling. In [EHBG07], Active Learning is used to drive sampling in imbalanced data. In Chapter 4, we use Active Learning for string matching. Refer to Section 3.3 for related work on Active Learning for Record Matching.

Finally, the way the samples are used in the learning can be adjusted to account for the class imbalance. It is long known that classification runs into problems if the involved classes are either not in a balanced distribution or not equally important (in terms of error cost) [BFSO84]. In *cost-sensitive learning*, classes are weighted according to the cost a specific error related to this class would incur. In Section 5.3.2, we employ such a technique under the name 'error scaling'. Changing the label distribution during training through sampling is in theory equivalent to adjusting the cost for different errors [BFSO84] albeit with some practical caveats [Wei04]. Also, sampling decreases the quality of the approximation, either by loss of information (undersampling) or distortion (oversampling) which may cause overfitting [CBHK02]. Cost-sensitive learning does not change the selection of data points themselves but only how they are used.

Logic

The interdependence of labels does not only cause the problematic prevalence of negative pairs. It also contains some structure that can be taken advantage of. One obvious structural aspect of the label distribution is that it is symmetric. That is, (a, b) is positive if and only if (b, a) is positive. This property is usually reflected in the model and it is made sure that only one of both pairs are used to avoid inefficiencies.

There is another property that can be exploited. In the case of a matching problem, it might be the case that both D and E are duplicate-free. Then, each represented entity can only be represented at most once in each set D and E. Therefore, there can be at most one match per element in both D and E. This has the following two implications.

The first implication affects the learning phase, at least in Active Learning. If one knows that elements d and e can have at most one matching element in the opposite set, and (d, e) has been revealed to be positive, then all tuples in $\{(\tilde{d}, e) | \tilde{d} \in D \setminus \{d\}\}$ and $\{(d, \tilde{e}) | \tilde{e} \in E \setminus \{e\}\}$ are necessarily negative. This significantly increases the pool of labeled data without requiring additional labelling.

The other implication affects the inference phase: If a model produces similarity values for all pairs, and each instance in one set can match at most one in the other, one can solve this as a weighted bipartite graph matching problem. If one expects that there are non-matching elements in both sets, one should first prune the bipartite graph by those edges below a minimum similarity value. In [Chr12], it is suggested to solve this by maximizing the weights in the matching. However, [GRC11] finds that non-optimal heuristics are more robust and give better results.

In clustering, this logical way of extending the negative data is not possible, because we do not deal with two distinct duplicate-free sources. Instead, the very task is to deduplicate or infer clusters of similar instances. A natural idea might be to instead extend the positive data by the rule of transitivity: If a is a duplicate of b, and b is a duplicate of c, can we not infer that a is a duplicate of c? In the ideal case, clusters form equivalence classes and this logic reasoning works.

However, relations implied by similarity do not usually yield equivalence classes. For example, string similarity defined by being closer than t in edit distance cannot be transitive. Equally, the definition of type-3 code clones (Section 5.1.1) cannot result in a transitive relation. And as seen in Section 2.1, human judgment of similarity behaves similarly. In more well-defined scenarios like in the deduplication of databases, it might make sense to assume transitivity of the duplicate relation [Mon00]. When many records are very similar (e.g., because their number is very high relative to how big the records are), this can still lead to incorrect inference through transitivity, but this happens rarely in real-world applications (see Section 6.8 in [Chr12]).

2.3.2 Generalization

That fact that we deal with tuples of instances in Metric Learning also requires some special attention in the evaluation of learned models.

Imagine that a system has been trained to recognize people's identity from a photograph. It has been exposed to pairs of photos from different people to learn to tell them apart. And it has been shown pairs of different photos of the same person to learn to identify the depicted person. Now, its creators want to measure the performance of this system to get an idea how well it will work in the real-world application, in order that they can be confident in its performance.

If they were to naïvely split the supervised data into training and test sets, without any regard to which tuples were selected, they would commit the following error: There would almost certainly be many pairs of photos in the test set that represent people that the system saw on photos in the training phase. In fact, it is also very likely that it will be evaluated on some of the same photos that it was exposed to then. The only thing this naïve split guarantees, is that no exact same tuple will appear in the test set.

Consequently, one cannot be terribly confident in the performance numbers that drop out of this evaluation. It might simply be the case that the system got very good at recognizing the particular people it saw during its training and excels at identifying them (or conversely, telling them apart). A realistic evaluation must make sure that the system generalizes to unseen people.

For example, in Code Clone Detection (CCD) there is the benchmark dataset BigCloneBench [SR15b, SR16]. It contains verified clone pairs and non-clone pairs and is therefore sometimes used for training and testing of supervised Code Clone Detection approaches (as is done in Chapter 5)[7]. BigCloneBench contains clusters of Java methods that have been collected by heuristic searches. Each cluster only contains methods for one specific functionality. So it would be advised to separate the *clusters* used in training from those used in testing. We call this 'cluster-aware data splitting' (Section 5.3.2).

However, this care does not seem to be universally taken. We are aware of one work [LFZ$^+$17] that practices cluster-aware data splitting by explicitly isolating cluster #4, as the biggest cluster, for training. There are no non-clones from pairs between separate clusters this way, but there are false positives in populating the cluster by the search heuristic. This way, the singular cluster contains verified non-clones that are used during training. Another work [ZH18] repeats its own evaluation with this exact data split (cluster #4 for training, and the rest for testing). However, it is claimed in [ZH18] that the performance measured is almost the same as with the splitting that is not cluster-aware. We run a comparison like that in Section 5.3.6, but find dramatic differences in performance between the two.

Other related works in supervised learning of CCD [SK16a, WL17, TWB$^+$18, ZWZ$^+$19] do not provide sufficiently explicit descriptions of how training and testing data are split. It is hence unclear if care is taken to split along cluster lines or to make sure that at least the same code fragments do not appear in both training and testing.

A similar case is reported in [VDK$^+$20], where many studies make a similar mistake in the data split. There, the data instances are not pairs but because of the common practice of oversampling, they can be relevant in Metric Learning as well. Since one class is in the overwhelming majority, these studies oversample from the other class by synthesizing artificial instances. However, they only split the data into training and test sets after this synthesizing step. By doing so, they leak information into the test set that can no longer be considered fully unseen. The authors of [VDK$^+$20] show that correct splitting leads to sizable performance losses compared to the replicated approaches, in most of the cases.

[7]The authors of [SFL$^+$18] find that it is already bad practice to use a benchmarking dataset for that.

2.3.3 Quality metrics

If we want to evaluate a binary classification model, we want to measure its ability to predict the correct one of two outcomes. In a case like ours (matching/deduplication), we pay special attention to one of the two. In a more general setting, this could be a diseased person (as opposed to a healthy person), or a fraudulent transaction (among normal transactions), or a security breach (among normal system logins), or an oil reservoir (within other positions in the landscape without oil). This more interesting class of the two is often called *positive* (through medical language it has found its way into everyday language; e.g., *HIV-positive*) and comprises the noteworthy kind of instances or events that a model is supposed to detect or retrieve. It is often the minority class because of its exceptional character.

In our application, the general population are *pairs* of texts or code fragments. The positive class are the matches, clones or duplicates. Using quality metrics that are appropriate in other contexts can lead to sub-optimal classification models in the context of imbalanced data sets [BTR16]. We will therefore put special focus on how the class imbalance in this specific setting affects the quality metrics.

Types of errors

It is important to know how good a classification model is at predicting whether an instance belongs to the positive class or negative class. Our model can make two kinds of errors. One is to predict an exceptional event, when there is none (type I error, false positive or error of commission). Colloquially, this is called a *false alarm*. Or it can fail to predict such an event, when there really was one (type II error, false negative or error of omission). We could also refer to this as a *miss*.

A model might make very cautious, specific predictions for the positive class. It therefore may have a relatively low number of false positives (type I error, see Table 2.1). At the same time, it may have many false negatives (type II errors). If a model does generous predictions of the positive class, these characteristics may be exactly reversed.

Which scenario is more desirable depends on the application. This will determine what costs are associated with what kind of error. In a security system that is supposed to catch breaches or attacks, it is worse to miss an event than to err on the side of safety and make more false alarms. Of course there is always a point of too many alarms, because investigating reported events by some downstream process (possibly involving human labour) may add up to too much wasted effort through the many false alarms by the "hair-trigger" security system. In the worst case, human users notified by too

	Ground truth		
	Positive	**Negative**	
Predicted positive	True positive	False positive (Type I error)	Precision: $\frac{\sum TP}{\sum PP}$
Predicted negative	False negative (Type II error)	True negative	
	Recall: $\frac{\sum TP}{\sum P}$		F_1: $\frac{2 \cdot \sum TP}{2 \cdot \sum TP + \sum FN + \sum FP}$
		Specificity: $\frac{\sum TN}{\sum N}$	Accuracy: $\frac{\sum TP + \sum TN}{\sum P + \sum N}$

Table 2.1: Confusion matrix and derived metrics

many false positives get tired, complacent with the inconsequential alerts, or actively dismiss incoming alarms (as in "the boy who cried wolf").

On the other end of the spectrum are systems that are detrimental if they produce (too many) false positives. For example, a cleaning robot that is supposed to detect junk and remove it. It is nice whenever it finds genuine junk and hence cleans the area it is responsible for some more. But a false positive might mean that it identifies a invaluable piece of modern art as junk and destroys it. And again, the less costly false negatives can still add up to prohibitive cost. If the cleaning robot makes so many type II errors that it effectively does not clean anything, it is not worth the cost of acquiring and running it. Additionally, it might give a false sense of the problem (of cleaning) having taken care of, while effectively, it has not.

Precision & Recall

There are two metrics capturing these two competing goals in avoiding each type of error. *Recall* is the number of true positives over the number of all positives (see Table 2.1). It measures the ratio of all positives that the model was able to retrieve. *Precision* is the number of true positives relative to all instances that were predicted as positives. That is, how much of the time the model was correct, when it claimed to have found a positive. Unfortunately, both metrics offer a very poor goal for predictive performance because they each can be trivially maximized. Any model that predicts all instances to be positive will have perfect recall and any model that predicts all instances to be negative will have perfect precision (or at least *one* instance that is very obviously positive in order to have a non-zero denominator).

Accuracy

A straightforward way to address this is to count the relative amount of avoiding both types of errors a model does in its prediction. This is called *accuracy* and is defined as the number of all correct predictions relative to the number of all instances (see Table 2.1). Accuracy conflates the two types of errors and as such, does not make a distinction between the two classes. It is symmetric in both classes and hence there is no concept of a *positive* class that applies here. That makes it easily generalizable to multi-class prediction. Accuracy can be a poor metric, however, when the task at hand has a very skewed class distribution. If that is the case in a binary classification task, the instances of one class will dominate the overall set of instances. Because of this, a trivial model that always predicts that a given instance belongs to the majority class, can achieve accuracy as high as the prevalence of the dominant class. In our specific application of deduplication, we deal with matches among the set of pairs of documents. Because of the combinatorics involved (see Section 2.3.1), we naturally deal with a highly skewed class distribution. That is why accuracy is not a good metric in our case.

F_1 score

The F_1 score (or F_1 measure) also addresses both kinds of errors. It is defined as the harmonic mean of precision and recall. These two metrics clearly refer to one of the classes as the *positive* one. This metric cannot be maximized by any trivial classifier as it combines the competing metrics precision and recall in a way that it is not enough to maximize one at the expense of the other. Hence, the errors are not interchangeable like they were in the case of accuracy.

The F_1 score can be generalized to the F_β score, where a higher value for the parameter β gives more importance to false negatives and a lower β emphasizes false positives. It has been shown that optimal thresholds for F_β are lower than optimal thresholds for accuracy [FK15]. Averaging of F_1 scores is misleading [FK15]. Instead, one can consider the F-Gain score that allows proper averaging and takes into account the always-positive baseline [FK15]. The F-Gain score can be translated into the corresponding F-Score.

The F_β is employed in several metrics used for the evaluation of machine translation (see Section 2.2.1). These are string similarity measures, but treat the aligning and matching of tokens as a small retrieval problem. F_β requires a good precision of token matchings while punishing short translations by also requiring good recall.

MCC

Matthew's Correlation Coefficient (MCC) is the correlation between the prediction of the model and the actual labels of instances (after identifying each of both classes with an arbitrary real number). The MCC is a function of all four cells of the contingency table. And it is symmetric with respect to the two classes, so there really is no concept of a *positive* class. The MCC was originally conceived of in biochemistry [Mat75] and has not reached much popularity in Computer Science literature.

Scatter plots

Computational models often do not just classify all samples or event candidates into positives and negatives. More often, they provide a ranking of candidates, where an instance is deemed more likely than another to be a positive if it is ranked higher by the model. Or the model gives a concrete (probability) score for each candidate which of course implies such a ranking. This allows the user of a system to choose an appropriate cut-off point (or score threshold, in the case of scoring) that defines the decision by the model underlying the system. In these cases, it is more informative to consider such a ranking model as a family of classification models.

A ranking implies a maximum of $n + 1$ many non-equivalent models because there are at most this many thresholds with respect to a given test set of n instances. This number can be less if the model is scoring instead of ranking and if some scores coincide.

Now, if one is not just optimizing for one metric but considering trade-offs between different objectives, one can do a scatter plot of the performance of this family of models with respect to any two quality metrics. Models without a threshold parameter that simply classify instances into the two classes will appear as one single point in such plots. For example, human experts have a hard time coming up with a score or a (consistent) ranking for instances like that. They rather use their expertise, i.e., knowledge and experience, to come to a definite answer, maybe with an additional rough score of how confident they feel about a particular decision.

These plots are a useful tool in visualizing characteristics of a model and its behaviour with different thresholds. One can pay attention to specific regions of interest [Faw06], for example, if one has constraints or costs for specific kinds of errors. Or one might have the requirement that one quality metric be higher than some lower bound after which one is free to optimize for the other. Thus, it serves as a good tool to make an informed decision about a decision threshold to turn the scoring/ranking model into a classification model.

In the following, we look at concrete examples of pairs of metrics that are in frequent use in several different fields.

ROC curve

The so-called Receiver Operating Characteristic (ROC) plots Recall (Sensitivity, or true positive rate, TPR) against Fall-out (or false positive rate, FPR). Sometimes, it is equivalently defined as plotting Recall against Specificity (or true negative rate, TNR).

The term *receiver operating characteristic* was coined during the Second World War when radar was used to pick up the presence of ships and planes. The sensitivity of the radar set (the receiver) could increase the sensitivity and pick up more faint signals with the side effect of picking up on the amplified noise which led to false positives. The points in ROC space are also called *operating points*. These characterize the behaviour of a specific setting an operator might use.

The first ROC plot was done in a technical report [PBF54] seminal for the field of Signal Theory [Swe73]. The researcher Lee Lusted worked at the Radio Research Laboratory at Harvard during World War II on the development of electronic radar countermeasures equipment and was exposed to the ROC curve as an instrument to analyze the quality of radar signal detection [Lus84]. In his later career as a medical doctor, he introduced this tool into the field of medical decision-making [Lus71]. Later, it was proposed to use Receiver Operating Characteristic curves to evaluate machine learning classifiers [Bra97].

ROC plots have some nicely interpretable properties. The major diagonal line indicates the performance of a random classifier where each point on that diagonal has a distinct rate at which it guesses the positive class. The top-right operating point represents the always-positive classifier and the bottom-left the always-negative classifier. Recall is normalized by the prevalence of the *positive* class whereas Fallout is normalized by the prevalence of the *negative* class. Therefore, the whole ROC plot is independent of the class distribution and specific error costs. The angle between the x-axis and the line connecting the point with the origin is proportional to precision [FK15]. An introduction to ROC curve analysis is given in [Faw06].

To go from some measured points to the whole ROC curve, one has to interpolate, as mentioned above. In the case of the ROC curve, linearly interpolating between the few well-defined points makes sense: The classifiers corresponding to points on the line between any two operating points can be achieved by interpolating the neighboring classifiers via *fractional sampling* [Faw06]. Especially, points on the convex hull can be realized by such an ensemble model. So, in addition to the area under the ROC curve (ROC-AUC), one can define the Area Under the ROC Convex Hull (ROCH-AUC).

The ROC-AUC of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance [Faw06]. It is worth noting that it is possible for a classifier to perform worse in a specific region of ROC space than a classifier with a lower ROC-AUC [Faw06]. Furthermore, ROC analysis overemphasizes the region with highest recall [FK15].

PR curve

The Precision-Recall-Curve (PR curve) plots Precision against Recall. It was conceived only after the ROC curve and was possibly inspired by it [FK15]. Random classifiers do not show up in the same way in PR curves. Their performance is found on the horizontal line at the height of the prevalence of the positive class.

A lot of favourable properties of the ROC curve do not hold for the PR curve. Interpolating linearly between the measured points is not meaningful and the set of operating points in PR space that are pareto-optimal are not convex or easily determined [FK15]. The ROC curve and the PR curve relate to each other in that a curve dominates in ROC space if and only if it does in PR space [DG06]. However, the PR curve is a better tool than ROC when evaluating on imbalanced data [SR15a].

PRG curve

The Precision-Recall-Gain-Curve (PRG curve) was conceptualized to improve on the PR curve. The central insight leading to the modification is that in Precision-Recall analysis, the best trivial baseline is the always-positive classifier. It has perfect recall and its precision takes the value of the prevalence π. So any classifier with precision or recall below π would not be worth considering. Consequently, it makes sense to rescale precision and recall to reflect this. The *precision gain* and *recall gain* are defined as precision (or recall, respectively), rescaled with a harmonic scale:

$$precG = \frac{prec - \pi}{(1-\pi) \cdot prec} = 1 - \frac{\pi \cdot FP}{(1-\pi) \cdot TP},$$

$$recG = \frac{rec - \pi}{(1-\pi) \cdot rec} = 1 - \frac{\pi \cdot FN}{(1-\pi) \cdot TP}$$

Points on the line between two classifiers in PRG space can be achieved by interpolating via fractional sampling, just like in ROC space (but not in PR space) [FK15].

In the PRG plot, isometric curves of F1 scores are straight lines parallel to the minor diagonal. More generally, isometric curves of F_β scores have slope $-\beta^2$ [FK15].

'Area under the curve'

The aforementioned scatter plots are often non-chalantly extended to a single metric that gives a summary for the whole family of models – the so-called *area under the curve*. It is worth unpacking what goes into the construction of this aggregated metric derived from the set of 2D points.

Firstly, the set of 2-dimensional points is discrete whereas a *curve* is the image of an interval mapped by a continuous function into a topological space. Secondly, the *area under a function graph* (of an integrable function) can be well-defined as an integral, whereas *the area under a curve* is not well-defined. These two gaps can be bridged under certain conditions.

The points *are* the image of a function, specifically the function that maps a threshold $t \in \mathbb{R}$ to the 2D point $(q_1(m,t), q_2(m,t))$ of the quality of model m with decision threshold t with respect to the quality metrics q_1, q_2. But this function takes only a discrete set of values as discussed above. If we could find a meaningful way of extending the points to a curve that happened to coincide with the graph of an integrable function, we could relate the points to the integral under that function and call it "area under the curve". There are three things that in principle can stand in the way of this endeavour.

First, for the curve to coincide with a functional graph and the integral to relate to the curve, the curve has got to have values in the first dimension that are monotone in the decision threshold t. In particular, $q_1(m,t)$ must be monotone as a function of t. This is not true for every quality metric, so we must keep this constraint in mind. Recall, for example, *is* monotonely decreasing. In other words, as you increase the decision threshold, fewer instances will be classified as *positive*.

Second, the way to connect subsequent points with curve segments has to relate in a meaningful way to the family of models. Otherwise, it is unclear what exactly the *area under the curve* would measure. We go into this aspect in the below examples for the concrete cases.

Third, there is a merely technical point. There might be non-unique values of $q_2(m,t)$ for a given value of $q_1(m,t)$. In these cases $q_2(m,t)$ is not a function of $q_1(m,t)$. For example, q_1 might be Recall and a negative instance might be the only one scoring in the interval $[t_1, t_2]$. Therefore, $q_1(m,t_1) = q_1(m,t_2)$, but $q_2(m,t_1) < q_2(m,t_2)$ (e.g., if q_2 is Precision). This breaks any attempt of representing the curve connecting all points with a functional graph. We can define that the interpolation will only be done on intervals $(q_1(m, t_1), q_1(m, t_2))$ where $\forall t > t_1 : q_1(m, t_1) > q_1(m, t) \geq q_1(m, t_2)$. That is, t_1 is the maximal threshold with that value of q_1 and t_2 is *one of the* thresholds with *the next* value of q_1. The value of the function is not well-defined *at* the points $(q_1(m, t), q_2(m, t))$ themselves but that is not a problem since it is a discrete set which has measure zero. How to meaningful interpolate in between the points depends on the pair of quality metrics.

Area under ROC, PR, PRG

When measuring the area under the curves described above, one gets a measure that is independent of the threshold and that aggregates the performance of the ranking/scoring model. As noted above, the ROC curve is entirely independent of the prevalence of the positive class and hence the AUROC may be viewed as only measuring the capability of the ranking/scoring model. However, it is worth noting that differences in the prevalence can make huge differences in the outcome. So tests should reflect the prevalence likely encountered in the real application.

The AUROC of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance [Faw06]. It is also equivalent to the Wilcoxon statistic [HM82].

Maximizing AUROC is not equivalent to maximizing AUPR [DG06]. AUPR indeed does not have an interpretable meaning (other than the expected precision when uniformly varying the recall, which does not make a lot of sense) and it can favour models with lower expected F_1 score [FK15]. The AUPRG, on the other hand, coincides with the expected F-Gain score [FK15].

Conclusion

Popular choices to measure the performance of matching systems with a single number are F_1 score and AUROC. Good alternatives seem to be AUPRG, F-Gain and MCC, but these seem to be relatively unknown in Computer Science. In [BTR16, MWGM10] one finds some more obscure quality metrics, especially for the use with imbalanced data sets.

3 Related Work

3.1 Optimal similarity measures

In [VN11], a system is proposed to match attributes to a predefined set of semantic classes (like phone number, last name, ZIP code, city) by comparing their values to reference data. The system provides a fixed similarity measure that is preselected to deal best with the variations usually encountered in an attribute of that class. The study [DSSOH07] is centered on estimating optimal thresholds for similarity measures. It shows that this consideration can be turned around and used to measure the quality of similarity measures themselves. Our approach to selecting similarity measures, by contrast, does not depend on specification of a domain but learns directly from sparse user feedback.

For several domains, tasks and even languages (or combinations thereof) similarity measures have been experimentally compared with respect to their usefulness in that specific setting. This includes generic domains and tasks like (personal) names [CRF03, Chr06, PS07, MYC08, GMIF16], ontology alignment [CH13, SMW15], and even obscure niches like obscenities in Russian [Che17]. Our approach does not require a corpus of such background knowledge or the specification of an existing target domain or task. The information which similarity measure works best is given implicitly by the feedback of the user, instead.

Systems for record matching[1] require the selection of string similarity measures for given attributes. FEVER [KTR09, KTR10, Köp14], a framework for evaluation and comparison of record matching algorithms, offers different kinds of hyperparameter search; random, grid search, and gradient descent. FEVER measures not only the performance in terms of the contingency table but also the labelling effort for tuning the parameters and training the matchers. In our approach these two stages are combined. The string similarity measure and threshold are not hyperparameters but rather the output of the learning loop. Our approach reaches near optimal results with up to around 20 labels whereas the labelling effort reported in the FEVER study [KTR10] ranges from 20 to 500 labels. Record matching and string matching are only somewhat comparable. Three of the 17 datasets we evaluate on (**fodorZagrat**, **census** and **coraATDV**) are matching records that are serialized in single strings.

[1] Ironically, many other synonymous terms for this exist, e.g., entity resolution/linking, record linkage, reference reconciliation, to name a few

There is some research on Metric Learning of string similarity measures, e.g., [ACGK08, BM03a, TKM02]. Some is based on the rigorous theory of kernel functions [BBS08, KJ12]. One specific variation is the learning of string edit distances. In the survey [BHS13], one section gives an overview over this specific area. Our approach cannot be categorized the same way in Metric Learning, since it merely selects an existing similarity measure and provides a corresponding threshold.

3.2 Optimal similarity thresholds

The distribution of similarity values of matching and non-matching attributes in practice always overlap. Oftentimes, applications require setting a definite decision threshold for classification[2]. This implies a trade-off between false positive and false negative errors. In other circumstances no single threshold defines this relation, but selections of thresholds have to be made at various points (e.g., in decision trees). Probabilistic record matching introduced by Fellegi and Sunter [FS69] works with a low and a high threshold classifying all pairs as links, *possible* links, and non-links. Here, the thresholds are chosen by estimating the errors from samples.

Two works from the field of information retrieval [AvH01, DSSOH07] deal with the optimal choice of a similarity threshold. Their applications can be seen as (one-to-many) matching of strings, namely queries to *relevant* documents. *Score-distributional threshold optimization* [AvH01] uses a statistical model to estimate an optimal threshold for this task. In this work, the authors define a model for the distribution of similarity scores and a measure for probability of relevance. The scores for matching pairs is modelled with a Gaussian distribution, while the scores for non-matching pairs is modelled with an exponential distribution. They introduce an effective way of approximating the score densities in this theoretical model without loss of too much accuracy. In [AKR09], it is observed that the statistical model in [AvH01] suffers from two shortcomings. One is the support incompatibility between the two used distributions; the exponential distribution is not defined everywhere. And the second is that the exponential dominates the Gaussian in the long tail of high scores; which is exactly the opposite of the reality of relevance of documents. The authors address this by modifying the distributions by (two different) truncated versions and accordingly modifying the estimation of the optimal threshold. In [KPDA09] it is shown that a mixture of Gaussians (matches) and a sum of independent exponential distributions adds complexity but is a better fit. The paper [ACG02] assumes a Gaussian distribution but applies outlier detection methods

[2]This simple process sometimes is referred to as the ‘threshold method’ [KKP06, BTR16].

to set good thresholds. Our method incrementally samples interesting samples that lie in the intermediate area of similarity values for many similarity measures. This sample is necessarily highly biased and does not allow for the estimation of such distributions.

Our method of picking a threshold does not assume and estimate any types of distributions. Instead, it empirically optimizes a quality metric on a supervised sample. This method is described in [AvH01] as the *straight-forward empirical method.* In [DSSOH07], this method is compared to a score-distributional model with two Gaussian distributions for relevant and non-relevant scores to estimate the optimal threshold. Again, our query strategy does not follow this theoretic model. In [DSSOH07], accuracy is the target metric, and the experiment is done on a small artificial dataset of 18 paper titles (i.e., the database) and 150 arbitrary manually created variations (i.e., possible queries) of these. The result of the straight-forward empirical method is also evaluated as a function of number of samples. This is equivalent to an Active Learning experiment with random sampling. In the one experiment, the threshold jumps to a close proximity to the optimum at around 40 samples, and makes another jump at around 75. In terms of labelling effort, each query entails 18 (interdependent) labels (1 relevant, 17 irrelevant). The *straight-forward empirical method* with respect to accuracy is also used in several studies comparing biological string data (DNA and rRNA sequences) [KOPC14, BDD15, BDBU$^+$18].

Decision trees can be seen as cascades of splits by (sometimes) continuous attributes. These splits are usually based on purity measures (like Gini coefficient or entropy) and only apply to the remaining population of samples that are relevant in the given node. Random Forests (see Section 3.3 below) further aggregate the decisions of many (shallow) decision trees. The selection of thresholds in decision trees is not comparable (because these have only local relevance) whereas our approach considers single-dimensional matchers given by similarity measures where the threshold applies globally.

As mentioned in the previous section, the system FEVER [Köp14] for evaluating record matching algorithms allows for hyperparameter search for individual matchers which includes similarity thresholds. These can be optimized via random search, grid search, or gradient descent. The search will be counted towards the overall effort of an evaluated matching system. In our approach, the tuning of the similarity threshold is integrated in the Active Learning loop and part of the output matcher.

Instead of overlap of score distributions, [GWP$^+$17] suggests to define user preferences for logical properties of the resulting matching of two sources viewed as a bipartite graph. The authors define two user preferences; *MaxGroups* chooses a threshold such that the number of non-trivial connected components is maximal. *MinOutJoin*, in contrast, selects a threshold such that the size of the full outer join is minimal. That objective balances between an empty matching and a matching of all pairs, both of which have a big outer join.

3.3 Active Learning for matching

There are many publications that address the Record Matching problem with Active Learning techniques. All of this related work differs in at least two important aspects from our work. Our approach yields the simplest possible model, namely just an existing string similarity measure with a decision threshold and we address String Matching, which is a special case and subproblem of Record Matching. Three of our datasets (**fodorZagrat**, **census** and **coraATDV**) are actually comprised of records that are represented by single strings.

Active Atlas [Tej02, TKM01, TKM02] is the first work to apply Active Learning to Record Matching. The fundamental idea of Active Atlas is to create many hypotheses about how the relationship between attributes from the two input sources can relate by transformations. These include many ideas built into string similarity measures (Section 2.2.1), like substring relationship, stemming and Soundex. Tokens occurrences are compared by the cosine similarity of their frequency vectors weighted with TF-IDF weights. Decision trees are built on the known examples (each from a random subset of attribute similarities) to form a committee. The query strategy is to maximize the disagreement in the committee as measured by the absolute difference in match and non-match votes.

There are fundamental differences in how Active Atlas is evaluated compared to our approach. Firstly, the whole committee is used to predict the matches whereas we only evaluate the single best hypothesis. Secondly, Active Atlas applies logical inference to improve the output matching. It uses the Hungarian method of determining a maximal bipartite matching with the constraint that there can be at most one match per instance per source. This can reduce the amount of false positives because some of the lower similarity matches will be discarded. Active Atlas also uses the other type of logical inference (Section 2.3.1) for Record Matching to increase the amount of labelled data which we also do not apply.

This system is evaluated on three data sets among which are two that we use as string matching problems in our evaluation (**fodorZagrat** alias **restaurant** and **business** alias **company**). There are two important differences. Active Atlas uses the *restaurant* data as split into three attributes, while we use the concatenated version. And crucially, these data sets included one important attribute that was used as the key attribute for creating the ground-truth in [CRF03, CRFR03]; the website URL in **business**, and the phone number in **restaurant**. These two reasons (and because Active Atlas uses a much richer model) lead to the almost perfect prediction on these data sets. Still, this level of accuracy is only reached after around 50 and 100 queries respectively.

Other works study variations of the concept of committees of decision trees (in principle a random decision forest [Ho95]). ALIAS [SB02] compares this approach to Active Learning with SVMs and Bayes classifiers. In [AGK10], active learning is used to arrive at decision trees that maximize recall while exceeding a user-defined precision level based on the assumption of a monotonicity property. This is compared to passive learning and to a linear classifier. A similar importance to precision is given in the active learning of blocking schemes [SW18]. RAVEN [NLAH11] uses stable matchings among attributes to define features and as well as among instances to define matchings. As for models, this paper compares decision trees to linear classifiers. In [XAF13], logic inference for deduplication (outlined in Section 2.3.1) is used to increase the amount of labeled data from queries by the decision trees committee. This kind of inference is used similarly in [CVW15]. Corleone [GDD+14] answers the queries from the random decision forest by crowdsourcing. DIAL [Dou17, DSLW17] proposes the initialization of the committee of decision trees by an unsupervised density-driven algorithm. Smurf [SGADA18] derives fields from strings and learns decision trees with active learning loops driving both the blocking and matching phases.

Another batch of work uses genetic programming. Since this approach works by randomly mutating and recombining solutions that are kept in a candidate pool of models, it naturally has a committee of hypotheses. EAGLE [NL12] defines a population of logical formulas over attribute similarity comparisons which are essentially equivalent to decision trees. In [IJB12], the population is made up of models that aggregate weighted comparisons of transformed attributes in several different ways.

A newer development is to learn a similarity or distance metric in the Active Learning loop. This is essentially Metric Learning which we will discuss outside of the Active Learning context in the following section. The first work in this direction is Bayesian Active Distance Metric Learning (BADML, [YJS07]). It is a general Bayesian approach that works with real-valued vector data. In [YJS07], it is evaluated by clustering an image and a sound data set. ACIDS [SN12] learns a generalized edit distance for Record Matching. And the first Neural Network based approach is proposed in [HGD19]. Here, bidirectional RNNs with gated recurrent units (GRU) are pretrained to yield embedding vector to represent the attribute values. The absolute difference of two attribute values for two records is then stored in vectors. The vectors are all added up across attributes. A shallow Feedforward Neural Network is then trained to predict matches from this 300-dimensional aggregate vector.

In [KQG+19], the Active Learning task is not a matching task itself, but tagging of sentences. It employs Metric Learning by a Siamese Network of bidirectional RNNs with LSTM units, which is done with a fully-supervised sentence similarity data set. The learned similarity metric is then used to filter redundant queries from a batch of queries generated in the Active Learning loop.

Over the years there have been some solutions that do not fit into the aforementioned model families. In [BM03b] an SVM is used in a weaker form of Active Learning called 'static-active selection'. Instead of using features solely based on string similarity and document frequency, [AJW+14] uses other features related to the represented entities, like their social network links. The paper [FCW16] proposes the use of Markov logic networks. LUSTRE [BQL+18] learns from (essentially) partial records a model made up of a set of mapping rules. The query strategy is a combination of uncertainty sampling and density-weighted sampling.

Recently, a benchmark framework for Active Learning of Record Matching has been published [MPSS20]. It allows the comparison of different models and query strategies on many data sets with several evaluation metrics.

3.4 Metric Learning for matching

Metric Learning is mainly applied for sensory recognition tasks like person identification by face or voice [Kul13, BHS13, KB19]. We will highlight some work that focusses on unstructured and token-based data like text or code, in order to be comparable to our own work.

The closest in methods to our work is [TSM15]. In fact, we built our model in [BA19] on their implementation. It is maybe the first paper applying nonlinear Metric Learning to tree-structured data, after existing work only applied linear methods or learned tree edit distances [BHS13].

Shortly after [BHS13], the idea of learning sentence similarity with Siamese Neural Network was picked up [NVR16, MT16] but instead using Recurrent Neural Networks which treat the data as strings. In [NVR16], a character-based bidirectional Recurrent Neural Networks with LSTM was trained to match similar job titles. And [MT16] applied the same type of network to short sentences and reportedly outperformed the Recursive model of [TSM15]. They also found that the Manhattan distance performs better than cosine distance. Besides a boost in performance, it also naturally delineates the meaning of sentences along neurons; Some hidden units were found to represent negation while others reflected categories of subjects, or of direct objects.

In Section 5.2.1, we observe that the negative sampling approximation of skipgram word2vec is equivalent to training a Siamese Network with contrastive loss. The only difference is that Siamese Networks are trained with supervised data, whereas word2vec has only the implicit ground-truth of co-occurrence. The paper [KBdR16] similarly extends the word2vec algorithm. First, words are represented by pretrained word2vec vectors (they compare both word2vec variants). They average the word vectors to obtain sentence representations because that is a strong model. These are then used in the CBOW fashion to predict a focus sentence from its context sentences.

As further noted in Section 5.1.4, embeddings pretrained on the co-occurrence task oftentimes result in similar vectors for antonyms. Siamese Networks, however, have a built-in symmetry for the relation they are representing. In order to leverage direct supervised learning of the antonym relation, [EW19] proposes a *Parasiamese Network* which is asymmetric. One side has a single copy of the base network whereas the other side applies the same base network twice in sequence.

Metric Learning has been also conducted with weak supervision by data sampled from Active Learning algorithms (see Section 3.3 above). Apart from that, there is a fully supervised linear approach using Largest-Margin Nearest Neighbors [LSLH18]. A supervised Metric Learning approach for Record Matching is presented in [YHMH19]. First, names, addresses, coordinates, and categories representing businesses and other venues are extracted from web pages. These are then represented by embeddings learned in an unsupervised way. A Siamese Feedforward Neural Network is trained to match the records. It is shown that hard sampling, attention mechanism in the Neural Network, and label denoising improve the performance. By contrast, we use a Recursive Neural Network to aggregate and compare unstructured data (programming code).

3.4.1 Code Clone Detection

In this section we only include such Code Clone Detection methods that are using some kind of representation learning. That means that we also exclude learning-based approaches that rely on handwritten or mined features like [LFZ$^+$17, SFL$^+$18, ZH18] here. We give a small overview over general Code Clone Detection methods in Section 5.1.2.

The first work learning a representation for the application of Code Clone Detection is presented in [WTVP16]. A representation of the terminal nodes in an Abstract Syntax

Tree (AST) is learned by training a Recurrent Neural Network to predict the next token in the sequence of terminal nodes. Then, an autoencoder is trained to merge the sequence of terminal nodes. Its objective is to reconstruct the original two child nodes from the parent node representation. In this way, the embeddings for the terminal nodes (learned in an unsupervised fashion) get extended to all internal nodes by another step of unsupervised learning. Finally, the representation of whole fragments are compared by Euclidean distance. Thus, similar to our approach, binarized AST representations are used to aggregate pretrained node vectors that are finally compared to reflect similarity. The main difference to our approach is that we only pre-train the node embeddings and use a supervision signal to finetune the aggregated representation. Later, some of the authors extended this work to types of sequential input (sequences of identifiers, AST node types or bytecode) learned and aggregated in a very similar way [TWB+18]. They compare it to a learned graph embedding of CFGs and combinations with (supervised) ensemble models on top of the unsupervised pretrained representations.

CDLH [WL17] is the closest related work to ours. It also aggregates and compares binarized ASTs with a Siamese Recursive Neural Network with LSTM units by supervised learning. In contrast to our approach, CDLH employs an additional layer and non-linearity after the aggregation to obtain binary features for the comparison. Only terminal nodes seem to have a word2vec vector representation since there is no mention of discrete information that does not have sequential context. We binarize our ASTs in a balanced fashion whereas CDLH creates rather deep trees. Also, in our work we focus on generalizability which requires careful cluster-aware data splitting (see Section 2.3.2).

A recent supervised approach [CYZ19] uses local aggregation (tree-based convolution) of embedding vectors followed by pooling and a fully connected layer (TBCNN [MLJ+16]). A Siamese network compares the vectors by cosine similarity and fine-tunes on supervised data from BigCloneBench (Java) and OJClone (C).

Also unsupervised approaches are being proposed recently [ZWZ+19, GWL+19]. Instead of using autoencoders as in [WTVP16, TWB+18], both employ a token prediction language modelling objectives. The approach presented in [ZWZ+19] uses AST trees that are sequentialized by preorder traversal. Embedding vectors for the tokens are trained with word2vec. This is then fed into a bidirectional Recurrent Neural Network with GRU units. In [GWL+19], node vectors are trained by *node2vec* [GL16]. This algorithm works very similarly to word2vec, but works on paths within graphs that are generated by random walks. The vectors representing AST nodes are then aggregated via a weighted average representation which is further refined by removing the first principal component as proposed in [ALM16] and compared by Euclidean distance.

In general, any learned vector representation is a candidate for Code Clone Detection, by simple comparison through distance metrics like Euclidean or cosine distance. A survey over such representations is given in [ABDS18]. Recently, the Transformer architecture that uses self-attention has started to be explored for code representation [KMBS19, KZTC20, FGT$^+$20]. An overview over embedding single code units like tokens or AST nodes is given in [CM19, KBR$^+$20] with a special focus on the open vocabulary problem in [KBR$^+$20]. A first benchmark measuring how well different identifier embeddings reflect similarity and relatedness was proposed in [WRP19].

3.4.2 Other Software Engineering applications

The following works study Metric Learning approaches in the same domain of Software Engineering. In contrast to our work, the data type either differs from code or is represented differently from our approach.

One of the seminal papers in Metric Learning proposed Information Theoretic Metric Learning (ITML [DKJ$^+$07]) which is a form of Linear Metric Learning with regularization. Among others, ITML is evaluated on a dataset about software failures. It uses counts of function usage as representation for program runs. It matches similar program configurations and inputs of failed runs by k-Nearest-Neighbors classification.

More recently, Metric Learning has been applied to match artifacts in Software Engineering contexts. The paper [ATGW15] proposes an approach to match artifacts of different modality (code and natural language). The code fragment representation is based on its parse tree, relative to a probabilistic context free grammar, while the natural language sequences are averages of word vectors. These two representations are then fine-tuned by noise contrastive estimation to optimize for the matching task. The authors propose as application *snippet retrieval* that matches code to natural language queries and *query retrieval* that retrieves a natural language description for a code fragment. More works that embed natural language artifacts and code fragments have been published in recent years. An overview and comparative evaluation is given in [CLK$^+$19].

In [GCCH17], a Siamese Neural Network is used to jointly embed requirement specifications and design descriptions. The natural language tokens are first mapped to word2vec vectors which are then aggregated via Recurrent Neural Networks. These representations are then compared by a shallow Neural Network.

4 Metric Selection for String Matching

There exists a host of string similarity measures developed in different fields (see Section 2.2.1). Each one is suited to solve a family of similar use cases. Even applications that look very different on the surface might deal with very similar kinds of sources of variation like typos or formatting variants. So in principle, one can expect that this big “zoo” of string similarity measures offers a sufficient toolkit to solve almost any application a practitioner might face.

However, very few practitioners that are experts in their domain are familiar with many string similarity measures and can confidently tell which one will perform better or worse in a given case. To make matters worse, even if the decision about the string similarity measure is taken a definite matching of strings requires a threshold for the similarity. This is difficult to choose because it involves arbitrary gradations and the consequence of this choice is not easily grasped.

This kind of information is an essential input to many standard approaches for Record Matching or Deduplication. In both tasks one seeks to identify pairs of records or representations that refer to the same entities. Solutions to this problem rely on similarity measures to compute similarity values on attribute level [KR10]. For example, the Fellegi-Sunter model for probabilistic Record Matching [FS69] needs both similarity measures and thresholds for all attributes. In fact, the setting of similarity measure and decision threshold is often an important factor in determining the matching quality [BN09]. Even in modern approaches that determine decision thresholds in themselves (or are independent of this concept) still rely on the definition of a similarity measure [KR10].

In this chapter, we propose and evaluate an approach to select both a string similarity measure and a threshold to solve a given string matching task. Our approach defines an interactive process that formulates queries of example cases to a user.

This chapter is based on a peer-reviewed publication [BA15].

4.1 Background

Record Matching and Deduplication are problems that frequently occur in managing and analysing data. Relational data is ubiquitous in any field and combining different sources of data can leverage the data better. However, if there is no unique key available with which to define a join, one needs to employ Record Matching. Similarly, an existing data source might have duplicates from imperfect data entry or joins.

As outlined above these tasks usually require the choice of a similarity measure. Figure 4.1 illustrates how different similarity measures yield different distributions of similarity values on the same data set. Choosing a decision threshold is very hard and not intuitive, because usually one does not have such supervised data available. The matches and non-matches usually cannot be perfectly separated by any threshold (unlike TagLink in Figure 4.1) because the distributions usually overlap. So, the threshold arbitrates a trade-off between the two types of error (see Section 2.3.3).

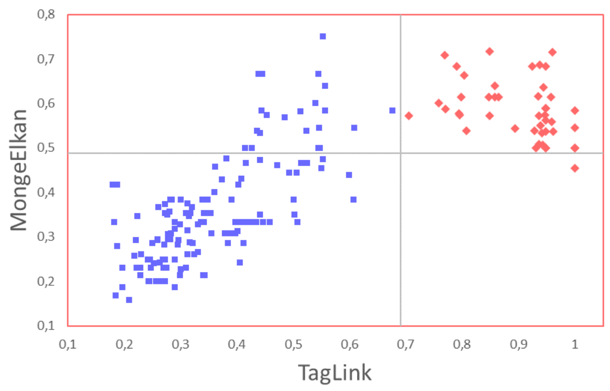


Figure 4.1: Similarity values and F_1-optimal thresholds in data set **ucdFolks** (Section 4.3.1) for MongeElkan edit distance similarity and the TagLink metric; blue squares represent non-matching pairs, red diamonds matching pairs with respect to the ground-truth

In this chapter, we present an approach for solving both problems—choosing a similarity measure and threshold—at the same time with minimal overhead for the user. The only required knowledge is the domain knowledge about the data. We build an Active Learning loop that iteratively queries the user for a label match/non-match of sample pairs of strings until a stopping criterion holds. We show that this approach is capable of arriving at solutions close to the optimum with very little user input by applying the approach to a diverse set of 17 string matching/deduplication datasets.

4.1.1 Active Learning

Active Learning is a form of semi-supervised learning. In this paradigm, a function on a domain is supposed to be approximated, but (initially) there is no data for the desired values of the function. Active Learning algorithms have the option to request the value of an instance in the domain under the target function from a so-called *oracle*. Apart from the objective to approximate the target function as accurately as possible (with respect to some quality metric), Active Learning tries to minimize the number of these queries to the oracle.

Which instances are available for querying is one important aspect along which to classify Active Learning strategies. In *pool-based* Active Learning, the algorithm can always access the same pool of instances. In *stream-based* Active Learning, where unlabelled instances are observed in a stream, the algorithm can at each iteration decide whether or not to query the current instance. An early variant allows *query synthesis* where instances are not sampled from real instances but instead synthesized.

The *query strategies* that govern how to pick the next instance or instances given the already labelled data are broadly organized under the following overlapping categories. *Uncertainty sampling* relies on a notion of uncertainty associated with each prediction made by the current model. The idea is that it is not worthwhile to query those instances for which the prediction under the model is highly certain assuming that the intermediate model already has some predictive power. Instead, one should query the label of instance with high uncertainty which is thereby removed locally.

The *query by committee* strategy works in scenarios where an ensemble (or *committee*) of models is maintained. At each step, a level of disagreement is derived from the predictions of each model in the committee for any input instance so that one can choose the instance for which the committee disagrees the most. The oracle then reveals which prediction is accurate for that instance and settles the disagreement. The models can then be updated accordingly. *Query by disagreement* is a specific variant of query by committee that assumes that the data can be perfectly reflected by a model in the solution space.

Finally, there are strategies that estimate the concrete impact of any possible query, taking into account all different outcomes. These are able to maximize the expected model change or minimize the expected output error or variance (for specific models, under certain conditions). The best primers for Active Learning are provided by Settles [Set12] and Munro [Mun21]. A study [PSPdC19] evaluates 15 Active Learning algorithms organized in 7 query strategies for 5 learning algorithms on 75 data sets and comes to the conclusion that Uncertainty Sampling and a strategy called *Expected Error Reduction* can generally be recommended.

There is a connection between Active Learning and the sampling applied in supervised Metric Learning which we describe in Section 2.3.1. And there is even a direct Active Learning formulation of linear transformation distance learning (generalized Mahalanobis distance) called Bayesian active distance metric learning (BADML, [YJS07]).

4.2 Approach

The goal of our approach is to pick the best similarity measure and threshold for matching similar strings in our data in terms of F_1-score (see Section 2.3.3). For our purpose, the similarity measures are a fixed set of black box functions of signature $\Sigma^* \times \Sigma^* \to \mathbb{R}$ for some finite alphabet Σ. On an abstract level, the idea of our approach is to combine all available similarity measures in a committee. Each one is associated with a threshold that empirically works best on the labelled data seen so far. Their predictions are then evaluated and aggregated to determine the string pair where these predictions are the most uncertain (query by committee). This string pair is then labelled by the oracle (the user) as match or non-match. According to this new information, some of the thresholds have to be updated. This also updates the whole committee and the notion of uncertainty for all remaining unlabelled instances. In each such round, a stopping criterion is evaluated and the loop is broken once it holds. The classifier (similarity measure and threshold) that works best on the labelled data is the output of the whole process. Figure 4.2 gives a schematic overview.

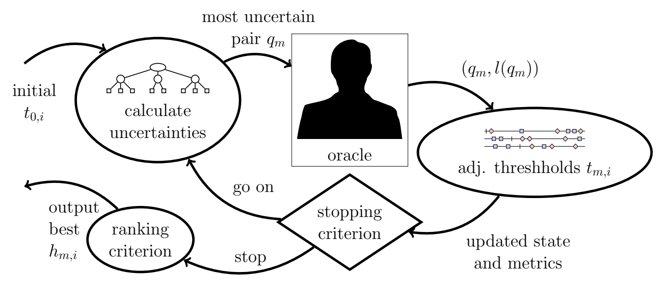


Figure 4.2: The process of Active String Matching

In the following we detail each aspect of this process concretely. We will use the terms and symbols listed in Table 4.1. The pool of all string pair instances is denoted by Q. The number m represents the number of iterations that are done which also counts the number of labelled instances. The string pair queried for labelling in iteration i will be denoted by $q_i \in Q$ and the total set of queried string pairs $q_1, \ldots, q_m$ in the first m rounds is denoted by Q_m. The function l gives us the ground-truth label $l(q_i) \in \{0, 1\}$ provided by the oracle for string pair q_i. Here, 1 (*positive*) represents a *matching* string pair, while 0 (*negative*) represents a *non-matching* pair. For each set of string pair instances T and a function p predicting their labels, we denote with $F_1(T, p)$ the F_1-score of the predictions of p with respect to the ground-truth.

Symbol	**Definition**
$m \in \mathbb{N}$	iteration number
Q	set of string pairs
$Q_m = \{q_1, .., q_m\}$	set of m queried string pairs
$l(q_1) \dots, l(q_m)$	labels provided by the user
$F_1(T, p)$	F_1-score a predictor p and ground-truth of a set $T \subseteq Q$
$S = \{s_1, \dots, s_n\}$	set of n similarity measures
$t_{m,i} \in \mathbb{R}$	threshold assigned to s_i in iteration m
$p_{m,i}$	predictor using s_i and threshold $t_{m,i}$ (in iteration m)
p^*_m	best predictor found in iteration m

Table 4.1: Important symbols for defining our Active Learning approach

4.2.1 Updating of thresholds

Next, we define how we update the thresholds associated with a similarity measure in a given round. Table 4.2 summarizes the symbols and terms used. We denote with $t_{m,i}$ the threshold associated with similarity measure s_i. This tuple defines a prediction function (or *predictor*) on the set of string pairs:

$$p_{m,i}(q) = \begin{cases} 1 & \text{if } s_i(q) \geq t_{m,i} \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

The function $p_{m,i}$ predicts a match, when the similarity value for the string pair exceeds (or equals) the threshold. In each iteration, the threshold is chosen to maximize the observed F_1 score. Figure 4.3 illustrates this in a concrete example. Each of the intervals between two adjacent string pairs and the two intervals before and after all pairs yields the same empirical F_1 score for every threshold in that interval. First, the interval with the maximal F_1 value is chosen from the (at most) $m + 1$ intervals. This procedure is called the *straight-forward empirical method* in [AvH01]. It is possible that the maximum is reached in two intervals (see Figure 4.4 for an example). In that case,

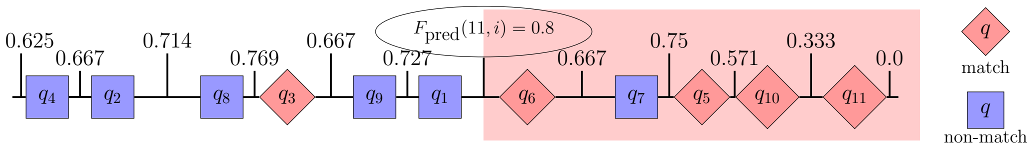


Figure 4.3: Determining of threshold $t_{m,i}$ based on maximizing the F_1 score w.r.t. all revealed labels. Similarity values are increasing from left to right (not shown). Matches are represented by red diamonds and blue squares indicate non-matches. Unlabelled string pairs are not shown for clarity.

Symbol	Definition
$F_{\text{pred}}(m, i) := F_1\{Q_m, p_{m,i}\}$	empirical F_1; the F_1 score for $h_{m,i}$ w.r.t. observed labels only
$C_1, \ldots, C_k \subseteq S$	k clusters of similarity measures
$p_{m,i} : Q \to \{0, 1\}$	prediction function of s_i with $t_{m,i}$
$p_m^{C_j} : Q \to [0, 1]$	(intermediate) prediction function associated to cluster C_j
$p_m : Q \to [0, 1]$	aggregated prediction function
$w_{m,i}, w_m^{C_j}$	weights used in the aggregation
$p_m^{\text{target}} := 1 - r_m$	target prediction value

Table 4.2: Symbols used for the query strategy

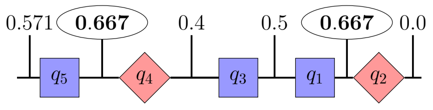


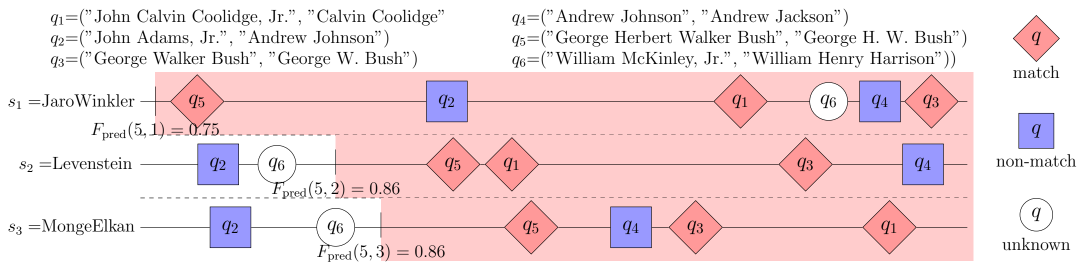
Figure 4.4: Maximum F_1-score values for a measure s_i may be achieved by multiple candidate thresholds

we simply pick the rightmost (containing the highest threshold) interval. That way, we favour precision over recall if F_1 is identical. In our experiments, we did not found much difference when comparing this solution to others. The threshold $t_{m,i}$ is finally defined as the arithmetic middle of the borders of the chosen interval.

Once the Active Learning loop terminates, s_i and $t_{m,i}$ with the maximal empirical F_1 score are the final output. However, this maximum might be not uniquely defined. This is especially true in early rounds when not many different F_1 score values are possible at all. Figure 4.5 shows an example with queries arranged by their similarity values. In this situation, we pick the similarity measure s_i with the least unlabelled pairs in the threshold interval around $t_{m,i}$ (in the example, there is only one such pair visible, for clarity). The intuition is that this threshold is more certain since there are less unlabelled instances around it. By p_m^* we then denote this currently best predictor. Figure 4.8 shows the range of true F_1 scores of the several prediction functions $p_{m,i}$ that yield the same maximal empirical F_1 score in an example experiment. The solid line corresponds to the prediction function p^*m picked by the heuristic.

4.2.2 Aggregated prediction function

Now we explain the specifics of how the committee is constituted and how the notion of its uncertainty is defined. All similarity measures used are treated as black boxes



q_1=("John Calvin Coolidge, Jr.", "Calvin Coolidge"
q_2=("John Adams, Jr.", "Andrew Johnson")
q_3=("George Walker Bush", "George W. Bush")
q_4=("Andrew Johnson", "Andrew Jackson")
q_5=("George Herbert Walker Bush", "George H. W. Bush")
q_6=("William McKinley, Jr.", "William Henry Harrison"))
s_1 =JaroWinkler
q_5 q_2 q_1 q_6 q_4 q_3
$F_{\text{pred}}(5,1) = 0.75$
s_2 =Levenstein
q_2 q_6 q_5 q_1 q_3 q_4
$F_{\text{pred}}(5,2) = 0.86$
s_3 =MongeElkan
q_2 q_6 q_5 q_4 q_3 q_1
$F_{\text{pred}}(5,3) = 0.86$
q match
q non-match
q unknown

Figure 4.5: A possible state during learning. Red diamonds indicate matches, blue squares indicate non-matches and the circle indicates a pair with unknown label. The highlighted areas (on the right) indicate that the respective prediction function predicts matches, the white area (on the left) non-matches. All labelled string pairs ordered by their similarity value.

and our approach does not rely on understanding how they work. Instead, we just use the observed behaviour. In reality, some similarity measures may coincide with their predictions on specific data, or even in general (given their default parameters).

Clustering of similarity measures

When aggregating the predictions to a committee, we do not want to give redundant similarity measures implicitly more weight than others just because they offer essentially duplicate views on the data. That is why the first step in the aggregation is clustering of similarity measures by their predictions. We represent each similarity measure by the vector of its similarity values on the current data. Then, we compare similarity measures by the Pearson correlation coefficient of their respective vectors. This similarity measure for similarity measures makes sense since Pearson correlation is invariant under affine linear transformation (with positive determinant) in both arguments. And such a transformation (a combination of scaling by a positive real factor and adding a constant) yields an equivalent classifier (i.e., similarity measure and threshold). If applied to the similarity values and the threshold, the resulting classifier will make the same predictions as the original classifier. We compute the correlations of similarity values with respect to the filtered data set that is rid of the more trivial non-matches (see Section 2.3.1 about Indexing). The correlation on this part of the data is more informative because it does not have a long tail of trivial non-matches with low similarity values.

This results in a symmetric matrix of correlation values between pairs of similarity measures which implies a weighted undirected graph. In this graph, nodes correspond to similarity measures and the weights on the edges are the correlation coefficient between the vectors of similarity values. We then take this graph of similarity measures and

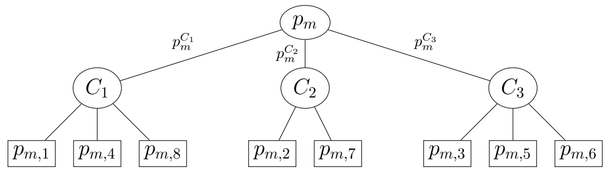


Figure 4.6: Structure of the aggregated prediction function p_m

apply a graph clustering algorithm [Noa09] that maximizes the so-called *modularity* of a graph clustering. We used the software Linloglayout[1]. Note that this can be computed on unlabelled data and is independent of any thresholds. It is therefore computed just once before the Active Learning loop and is fixed. In our experiments with 24 similarity measures, this procedure resulted in $2 - 5$ clusters of size $1 - 12$ depending on the data.

Aggregating

Now we use the prediction functions $p_{m,i}$ and aggregate their predictions using the clustering of the underlying similarity measures. Figure 4.6 gives an overview of the two step process. First, we combine the prediction functions $p_{m,i}$ according to the cluster membership of s_i to intermediate prediction functions p_C. We weight each function $p_{m,i}$ by its empirical F_1 score $w_{m,i} = F_{\text{pred}}(m,i)$, assuming that functions that are better able to reflect the revealed labels will be better at predicting the unrevealed labels:

$$p_m^C(q) := \left(\sum_{s_i \in C} w_{m,i}\right)^{-1} \cdot \left(\sum_{s_i \in C} p_{m,i}(q) \cdot w_{m,i}\right) \tag{4.2}$$

Then we combine the prediction functions of the clusters to an overall prediction function p_m:

$$p_m(q) := \left(\sum_{j=1}^{k} w_m^{C_j}\right)^{-1} \cdot \left(\sum_{j=1}^{k} p_m^{C_j}(q) \cdot w_m^{C_j}\right), \tag{4.3}$$

where $w_m^{C_j} = \frac{1}{|C|} \sum\limits_{s_i \in C} F_{\text{pred}}(m,i)$. These weights $w_m^{C_j}$ make sure that the more promising clusters contribute more to the prediction. Finally, the expression in equation 4.3 can be simplified to:

$$p_m(q) = \left(\sum_{j=1}^{k} \sum_{s_i \in C_j} \frac{1}{|C_j|} w_{m,i}\right)^{-1} \cdot \left(\sum_{i=1}^{n} p_{m,i}(q) \cdot w_{m,i}\right) \tag{4.4}$$

[1]https://code.google.com/archive/p/linloglayout/

This aggregated prediction function p_m (with the threshold p_m^{target}) is used to find the most uncertain string pair, that hence would serve best to tell the more adequate similarity measures from the rest. As such, it aggregates over all similarity measures, including the poorly suited ones. For this reason, p_m performs worse with respect to the ground-truth than p_m^*, which is the output of our Active Learning loop. Since p_m^* comes from a singular similarity function, it is also more efficiently evaluated and more readily interpreted than p_m.

4.2.3 Query strategy

In this section, we explain the query strategy used to pick the next string pair to be labelled by the oracle. Our query strategy can be described as an instance of uncertainty sampling, but also as query by committee (in terms of the categories presented in Section 4.1.1). Each similarity measure gets a threshold that optimally reflects the revealed labels. Then, all associated prediction functions are aggregated to an overall prediction function. Weighting accounts for redundancy among similarity measures and favours the predictions of functions that reflect the seen data better. This committee might completely agree on the label of certain yet unlabelled string pairs. There is no point in querying these, since they very likely actually have the predicted label. To aim for the opposite of these instances with high prediction certainty, we formalize our notion of *uncertainty* based on the prediction function of the committee.

The most certain outcomes are associated with the predictions $p_m(q) = 1.0$ and $p_m(q) = 0.0$. The most uncertain is the outcome of string pairs q with predictions $p_m(q) = 0.5$. Whatever the real label turns out to be in such a case, it is certain that many prediction functions $p_{m,i}$ will be wrong because of the way we defined the function p_m. As a consequence, many thresholds $t_{m,i}$ will have to be updated to reflect this new information $(t_m \neq t_{m+1,i})^2$. The first idea is therefore to find a string pair q that minimizes the expression $|0.5 - p_m(q)|$.

Cold start problem

This strategy works fine in later iterations, but initially some of its assumptions just do not hold yet. One of these assumptions is that the committee already is quite good at predicting. In the very beginning, when the thresholds are not based on much information, this is not very likely. This problem of bootstrapping informative data

[2] Note that according to our rules, a threshold $t_{m,i}$ might have to be updated, even if the associated prediction function was correctly predicting $p_{m,i}(q_{m+1}) = l(q_{m+1})$, simply because q_{m+1} happens to shrink the interval $t_{m,i}$ is in.

from imperfect intermediate classifiers to create better classifiers is a common problem in Active Learning and known under the name *cold start problem* [AP11].

Secondary query ranking At the very beginning, not all parameters described in our query strategy are even well-defined under the initial conditions. First of all, the thresholds $t_{m,i}$ do not have any value in the first round $m = 1$, because they are always set to separate the seen data, which is simply not available at first. Consequently, also the weights $w_m^{C_j}$ cannot be defined initially as they can be in later iterations. We therefore set $w_m^{C_j} = 1.0$ while the empirical F_1 score is undefined. That is the case as long as there is no positive, that is, matching label among the revealed labels, because precision is undefined. Even if set to zero, no F_1 score can be defined, because also no recall can be achieved.

To define the prediction functions, we also require thresholds for all similarity measures. The initial thresholds $t_{0,i}$ can actually be set arbitrarily, because of the following observation. When aggregating the prediction functions defined by these thresholds, initially all weights $w_m^{C_j}$ are identical. This remains true as long as the first queries have the same label, so at least for one more iteration. Then there is also a limited number of ways string pairs can be distinguished by their overall prediction under p_m. Then minimizing the distance to the target prediction is in general not uniquely defined.

In a situation like this, it is advisable to apply a secondary ranking to the string pairs, to make sure that an informed choice is still made. We first determine the rank of each string pair with respect to the similarity value under a given similarity function. So each string pair is associated to a vector of n ranks, for the n similarity functions. We then calculate the variance of these numbers and pick the string pair with highest variance. The idea is that if all similarity functions agree to rank a string pair relatively consistently (no matter how high), it will not yield much information to reveal its label.

Regulating target uncertainty Even after the initial cold start, the prediction functions might not be very well tuned. In particular, there might be an overall bias of the individual prediction functions towards making more false positives or more false negatives. As a consequence, what the committee predicts as close to 0.5 might be a match (or non-match) more often than not. So these instances just contain less information because of this bias. In fact, in this situation, a lower value than 0.5 would give us more uncertain string pairs. Therefore, we build a self-correcting mechanism into our query strategy. We track how often we reveal a match label (value 1) and how often a non-match label (value 0) as a measure of the current bias. Instead of aiming for prediction 0.5, we will aim for a target prediction p_m^{target}, which is the complement to the ratio of matches relative to all queries:

$$p_m^{\text{target}} := 1 - r_m = \frac{1}{m-1} \sum_{n=1}^{m-1} 1 - l(q_n)$$

If, for example, there is a bias of predicting matches (i.e., making more false positives), the revealed labels will more likely be *non-matches*. Consequently, p_m^{target} will be higher than 0.5. This new aim will it make more likely to reveal a matching string pair in the next round. In this way, the query strategy self-regulates to account for any bias.

Adjusting this target ratio makes sure that revealed labels have a high chance of being relatively balanced. We see this empirically in Section 4.3. A desirable side-effect is that this increases the possibility for the similarity measures to be told apart by their empirical F_1 scores. When one reveals k match labels and $n-k$ non-match labels, there are $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ many distinct ways of ordering the string pairs (modulo equal labels). And for measuring the maximum empirical F_1 score, the ordering of the labels (not the string pairs) is all that matters. This number is maximized by $k = \lceil n/2 \rceil$ and $k = \lfloor n/2 \rfloor$[3].

4.2.4 Stopping criterion

Active Learning is an iterative process and its strength lies in choosing highly informative instances for labelling which constitute a fraction of the total data set. In contrast to fully supervised approaches which rely on a big dataset that has already been labelled, Active Learning can make quick gains and approximate a good solution. In this sense, Active Learning is an instance of multiobjective optimization. One objective is to approximate the ground truth while a competing objective is to minimize the labelling effort incurred in doing so.

Defining a stopping criterion is a systematic way of striking this balance. It is not advisable to stop too early when the biggest gains are made. However, one should also not continue to query once the returns start to diminish. We noticed that the progress on different data sets was made at different speeds. As such, it did not make sense to simply stop after a fixed number of queries. Instead, we measured the total number of changes that were done to the thresholds during the learning loop. This was a reasonable proxy for the real progress in terms of prediction quality.

[3]By using Sterling's formula to bound $n!$ and $\frac{n}{2}!$ (from below or above, accordingly), one finds the asymptotic behaviour $\binom{n}{n/2} \in \Theta\left(\frac{2^n}{\sqrt{n}}\right)$.

Formally, we measured this quantity, where δ is the Kronecker-delta indicating equality of its arguments:

$$TC_m \coloneqq \sum_{i=1}^{n} \sum_{j=0}^{m-1} 1 - \delta(t_{j,i}, t_{j+1,i}),$$

Our stopping criterion then simply holds once this measure exceeds a given threshold. The problem remains how this threshold is to be set. But since the progress relative to this metric is more uniform, one can choose the threshold based on the observed performance on some known problem instances and expect a similar relative performance on novel problems.

4.3 Evaluation

We evaluate our Active Learning approach to String Matching on a wide range of data sets. We study the trade-off between labelling effort and quality and show a stopping criterion that can help strike the trade-off.

4.3.1 Experimental setup

Data sets

All external data sets were used in the evaluation [CRF03] for the SecondString string similarity toolkit [CRFR03]. Therefore, we use the same names for these datasets. Most of these external data sets go back to the evaluation [Coh00] of the WHIRL DBS. Original data can be found with the WHIRL website[4], while the final data used in SecondString can be found in its GitHub repository[5]. Unfortunately, the documentation of the datasets is very weak and contradictory at times.

A total of 8 external data sets that we used were created for [Coh00]: **bird1, bird2, bird3, bird4, game, park, business** and **animal**. Three of the bird datasets (**bird1**, **bird3** and **bird4**) and the **park** dataset were created via so-called *hotlists* on the web (on governmental, educational and research websites). That means that manually curated lists that hyperlink to web pages corresponding to individual entities (bird names and names of US National parks, respectively). The name of the hyperlink and a fixed field in the target web page can thereby reliably match two variants of identifiers. Similarly, **business** was created by identifying company names by their common web

[4]`https://www.cs.purdue.edu/commugrate/data/whirl/match/`

[5]`https://github.com/TeamCohen/secondstring/tree/master/data`

Dataset name	**src 1**	**src 2**	**Original**		**Reduced**		
			pairs	**match**	**pairs**	**match**	**m / p**
string matching problems							
bird3	23	15	345	15	25	*14*	56.00%
USPresidents$^+$	43	43	1,849	43	173	43	24.86%
DBconferences$^+$	54	54	2,963	54	2441	54	2.21%
bird1	317	20	6,340	19	672	19	2.83%
faoMembers$^+$	194	194	37,636	194	2,633	*192*	7.29%
bird2*	914	68	62,152	64	4,089	64	1.57%
game	798	105	83,790	41	4,276	41	0.96%
bird4	564	155	87,420	155	11,297	155	1.37%
park	393	258	101,394	252	6,767	*250*	3.69%
census	449	392	176,008	329	18,438	*326*	1.77%
fodorZagrat	532	331	176,092	114	73,657	*112*	0.15%
nobelLaureates$^+$	839	839	703,921	839	27,011	*831*	3.08%
business*	1,162	962	1,117,844	310	502,316	*309*	0.06%
animal*	4,719	817	3,855,423	178	93,661	*175*	0.19%
string deduplication problems (single source)							
UVA	116		6,670	280	2,932	*272*	9.28%
coraATDV	956		456,490	7,766	453,987	7,766	1.71%

Table 4.3: Data sets (ordered by number of pairs) for string matching (upper part) and deduplication problems (lower part). Column "Reduced" shows effects of the indexing step. Asterisks * indicate that ground-truth has been corrected; italic number of matches means false negatives due to indexing. Plus signs $^+$ indicate newly introduced data sets.

page URL. These were taken from lists provided by an electronics hardware supplier (Iontech) and a commercial information broker (Hoover). After applying the joint key, 10-15% false negatives had to be corrected. The matching in **animal** was created by matching the corresponding scientific name of the entities. Here, it was only required that one of the two is a substring of the other. Finally, **bird2** and **games** (educational computer games) were matched manually. On inspection of the data we found and corrected errors in the matching in three data sets (**business**, **animal**, and **bird2**).

The **census** dataset was also introduced in [CRF03] and is attributed in the documentation to William Winkler of the US Census Bureau. Rather than real data, it is reportedly synthetic data. It contains realistic variations like misspellings but it is not specified how this data was generated. Web research and email exchanges with former collaborators of William Winkler could not fully elucidate the genesis of the data.

The remaining external data sets originate in other work. The **restaurant** dataset goes back to [TKM01]. It uses data from an online catalogue of restaurants (Zagat) and the Department of Health[6]. Records were matched by their common telephone number followed by a manual revision and concatenation of their four fields into one string (name, address, phone number and restaurant/food style). The **coraATDV** data set was introduced in [MNU00]. It consists of (duplicate) citations of papers belonging to a fixed set of three specific authors. Four values of these records (author, title, date, and venue) were concatenated into single strings that were finally truncated at the length of 60 characters. The data originated from the Cora website[7] that provided a search interface to over 50,000 computer science research papers.

In [CRF03], both **ucdFolks** (personal names of staff at UC Dublin) and **UVA** (names of institutions near University of Virginia) are attributed to [ME96]. However, [ME96] only deals with names of academic apartments at the University of California, San Diego, and Stanford University. The SecondString documentation[8] goes on to claim that Nicholas Kushmerick is the "source" of these data sets while continuing to claim that Monge (co-author of [ME96]) is the "original source"[9]. We could not verify this claim [EM20]. In the case of **ucdFolks**, it makes sense that Nicholas Kushmerick would be among the authors because his dissertation was on generating what today would be called web scrapers [Kus97]. He had used a similar dataset (names of faculty members of 30 computer science departments) in [FK00] and he was working at the University College Dublin at the time. However, we could find no definite proof for his authorship of **ucdFolks** outside of the (unreliable) SecondString documentation and his email address at the University College Dublin is no longer active. The dataset **UVA** is attributed by SecondString in the same way to Kushmerick ("source") and Monge ("original source") but we could not find any further evidence for that. A parsimonious explanation for this attribution could even be a copy-and-paste error (because of the exact same wording).

In addition to the data sets available in the SecondString repository, we created four new data sets. For some, we used the above described "hotlist" approach. For **USpresidents** (names of US presidents), this was done using Wikidata[10], for **faoMembers** (country names) the website of the Food and Agriculture Organization (FAO)[11] and for **nobel** (names of Nobel laureates) we used Wikipedia[12]. The dataset **DBconferences**

[6]The SecondString documentation claimed it was data from Zagat and another online restaurant platform (Fodor's).

[7]`http://www.cora.whizbang.com`, now defunct

[8]`https://github.com/TeamCohen/secondstring/blob/master/data/README.txt`

[9]literally: "original source: Alvaro Monge, I think."

[10]`https://www.wikidata.org/`

[11]`http://www.fao.org/`

[12]`https://www.wikipedia.org/`

was created by matching entries of database conference names in WikiCFP [13] with the CORE Conference Ranking[14] by their abbreviations followed by a manual revision. These new data sets as well as the three corrected data sets from SecondString are available online[15].

To counteract the high skew in the label distribution we employ blocking (see Section 2.3.1). We use n-gram blocking as implemented in [CRFR03] with $n = 4$. Note that in principle, the concerns about cluster-aware data splitting apply (see Section 5.3.2). But only two of our String Matching data sets actually correspond to deduplication problems, and because of the incremental sampling by Active Learning, the amount of seen data is tiny compared to the unseen data.

Similarity measures We used the following 24 string similarity measures implemented in SecondString [CRFR03] organized by psychological similarity model (see Sections 2.1, 2.2.1). We use the names used in the source code for easier reference.

Spatial model The first set of similarity measures from the spatial model family are based on cosine similarity of TF-IDF vectors. In addition to the standard version **TFIDF**, there are variations that take into account variations of tokens (**SoftTFIDF**, **SourcedTFIDF**, **SourcedSoftTFIDF**) when constructing the document vectors. And finally some concrete derivations of SoftTFIDF named after their employed secondary similarity measure: **JaroTFIDF**, **JaroWinklerTFIDF** and **MongeElkanTFIDF**. One other metric we use is called **Mixture** in SecondString. It considers each tokenized string as sample from an unknown distribution of tokens. These distributions are then compared by Jensen-Shannon distance which is based on the Kullback-Leibler divergence. The distributions are estimated using maximum likelihood estimation.

Contrast model The only similarity measure from SecondString we used that falls into the contrast model category, is measuring the **Jaccard** overlap of tokens.

Structural Alignment The Level2 algorithm introduced by Monge and Elkan [ME95] and coined by Cohen et al. [CRF03] falls into the category of structural alignment, because it matches (unidirectionally) the tokens in one string with the tokens in the other. We use **Level2Levenstein**, **Level2Jaro**, **Level2JaroWinkler** and **Level2MongeElkan** which are named after their secondary similarity measure.

[13] `http://www.wikicfp.com/cfp/`
[14] `http://www.core.edu.au/conference-portal`
[15] `https://pvs.ifi.uni-heidelberg.de/team/lb/`

Here, ‘MongeElkan’ refers to the SmithWaterman edit distance with Monge-Elkan weights (see next paragraph). We also use the **TagLink** metric which finds a maximal matching with respect to an internal similarity measure without favouring any of the two input strings.

Transformation model In this category, we include several edit distances. There are **Levenstein**, and an adaption that normalizes the Levenstein score by the maximal length of both inputs, called **ScaledLevenstein**. Further, we include the two distances originating from the biology domain, **NeedlemanWunsch** and **SmithWaterman**, as well as an approximation to NeedlemanWunsch (**ApproxNeedlemanWunsch**). Further, we include the **MongeElkan** edit distance, which is a variant of Smith-Waterman with adjusted weights and equivalence classes of similar characters, and finally, a variant without the character classes (**AffineGap**).

Hybrid model The **Jaro** metric developed at the US census bureau and its Winkler modification (**JaroWinkler**) combine both ideas of overlap as well as edit distances. And the **AveragedStringDistanceLearner** averages the result of a set of secondary similarity measures (by default, these are JaroWinkler, ScaledLevenstein, Jaccard, TFIDF and JaroWinklerTFIDF).

Some similarity measures actually coincide (given their default parameter settings), others strongly correlate or coincide on certain data sets. Our approach is designed to be robust to these phenomena, given that many users may not have the resources to understand all algorithms in detail, before proceeding.

4.3.2 Quality metric

In our approach we built in the target quality metric F_1. However, nothing in our approach relies on this exact choice. Any other metric that accounts for both false positives and false negatives can in principle be employed to ranking the similarity measures like Accuracy, MCC, AUROC, AUPR or AUPRG (see Section 2.3.3). To turn the similarity measures into prediction functions, one needs a decision threshold. And these metrics are all defined without a threshold (except Accuracy). So one would have to pick a different metric or a completely different approach to define the best thresholds, to define the (aggregate) prediction functions, which in turn is needed to define uncertainty.

Note that we report the F-score after matching according to the final chosen similarity measure and decision threshold. This can likely be improved for matching problems by further considering the whole of the matching problem as a weighted bipartite assignment problem (with respect to similarity weights). This includes the logical constraint that each record in both data sets can be matched with at most one other record as done in [Tej02]. This logic likely excludes some matches in favor of safer matches and can be expected to improve the precision without harming the recall (see also Section 2.3.1). Instead, we decide locally on an absolute threshold for the evaluation that may induce contradictions globally[16].

We have already defined the empirical F_1 score $F_{\text{pred}}(m, i)$ (see Table 4.2) that we used to define the weights of the aggregation and the ranking of the prediction functions. When we measure the F_1 with respect to the whole ground-truth, we define the *true* F_1 of the best prediction function in iteration round m as follows:

$$F_{\text{true}}(m) = F_1(Q, p^*_m).$$

This is an estimation of the general quality on the whole data. We also define a ceiling for what we can possibly achieve with this approach, given the solution space. So we define:

$$F_{\max} = \max_{p \in P} F_1(Q,\, p),$$

where P is the set of all possible prediction functions resulting from any of the similarity measures S with any decision threshold in $\mathbb{R}$.

4.3.3 Labelling effort vs quality

The central idea of the Active Learning paradigm is to make fast progress in learning. So the most important aspect of our evaluation is how labelling translates into progress in terms of classification quality. It is intractable to show and discuss the true F_1 score for all experiments and iterations and derive conclusions from that. Instead, we use the stopping criterion introduced in Section 4.2.4 as a way to measure progress. It was conceived so as to make the progress on different datasets more comparable than is possible with raw iteration count.

In Table 4.4 we show the results for two different values for TC_m (total number of threshold changes). The lower decision threshold of $TC_m \geq 75$ means that each of the 24 similarity measures in S have had their threshold changed just over 3 times on average. It is trading some of the achievable quality off for a lower iteration count. So

[16]‘Unconstrained ManyMany’ in terms of [GRC11]

for roughly 7 on average, one achieves roughly 92% of the achievable F_1 on average. A higher quality solution is reached at $TC_m \geq 75$ with a bit more than double the effort (∅15 labels) one achieves 97% of the maximal quality. Both seem like acceptable amounts of human effort given that only domain knowledge of the data is required.

Note that the labelling effort is likely over-estimated. Leveraging logic inference (Section 2.3.1), one can increase the amount of negative labels that can be deducted from the feedback from the oracle without increasing the labelling effort. This is done in [Tej02, CVW15]. In [XAF13], it is similarly employed for deduplication. Although the inference there is less productive than in a matching problem which is much more constrained. It would be a promising extension of our approach to make use of this information. This would likely introduce a systematic bias towards discovering negative labels and the self-regulating feature of our query strategy would likely favor more obvious positive samples.

		stop at $TC_m \geq 75$					stop at $TC_m \geq 130$				
Data set	$F_{\max}$	final m	s^*	t^*_m	F_{true}	$\frac{F_{\mathbf{true}}}{F_{\mathbf{max}}}$	final m	s^*	t^*_m	F_{true}	$\frac{F_{\mathbf{true}}}{F_{\mathbf{max}}}$
bird3	1.000	5	L2_JW	0.923	1.000	100.00%	15	TL	0.598	1.000	100.00%
USPresidents	0.953	6	ST	0.403	0.925	96.98%	14	TL	0.778	0.943	98.85%
ucdFolks	1.000	5	ASD	0.342	0.917	91.67%	12	TL	0.643	0.989	98.90%
DBconferences	0.874	8	ME	0.779	0.851	97.45%	14	ME	0.840	0.845	96.75%
bird1	0.947	7	TFIDF	0.513	0.947	100.00%	16	TL	0.766	0.944	99.69%
faoMembers	0.961	9	L2_L	-0.300	0.958	99.70%	21	L2_ME	0.949	0.897	93.37%
bird2	0.944	7	JWT	0.982	0.918	97.25%	12	JC	0.633	0.929	98.42%
game	0.846	10	L2_JO	0.886	0.805	95.16%	20	L2_JO	0.927	0.827	97.70%
bird4	0.980	6	TL	0.615	0.951	96.96%	12	L2_JW	0.900	0.955	97.45%
park	0.970	7	SL	0.675	0.913	94.08%	15	ST	0.571	0.921	94.94%
census	0.899	6	A	0.609	0.665	73.94%	12	TL	0.816	0.817	90.80%
fodorZagrat	0.978	8	TL	0.751	0.959	98.09%	17	JWT	0.644	0.968	99.05%
nobelLaur.	0.989	7	JWT	0.399	0.790	79.93%	15	ST	0.443	0.882	89.21%
business	0.971	8	ASD	0.685	0.960	98.93%	16	ASD	0.663	0.964	99.30%
animal	0.926	8	M	0.495	0.891	96.24%	18	M	0.444	0.893	96.43%
UVA	0.907	7	JO	0.858	0.633	69.72%	13	TL	0.721	0.899	99.04%
coraATDV	0.800	6	ME	0.551	0.648	81.07%	15	L2_L	-0.675	0.736	92.07%
average		**7.06**				**92.19%**	**15.12**				**96.59%**
median		**7.00**				**96.96%**	**15.00**				**97.70%**
std. dev.		**1.30**				**9.39%**	**2.63**				**3.24%**

Table 4.4: Results after stopping learning at $TC_m \geq 75$ and at $TC_m \geq 130$. Final m is the number of iterations, s^* is the best final similarity measure, and t^*_m the corresponding final threshold. $F_{\mathbf{true}}/F_{\mathbf{max}}$ gives the ratio of solution quality (i.e., F_{true}) to maximum achievable quality ($F_{\max}$). Similarity measures: L2_*=*Level2**, JW=*JaroWinkler*, ST=*SourcedTFIDF*, ASD=*AveragedStringDistanceLearner*, ME=*MongeElkan*, L=*Levenstein*, JWT=*JaroWinklerTFIDF*, JO=*Jaro*, TL=*TagLink*, SL=*ScaledLevenstein*, M=*Mixture*, JC=*Jaccard*. Note that thresholds may be negative.

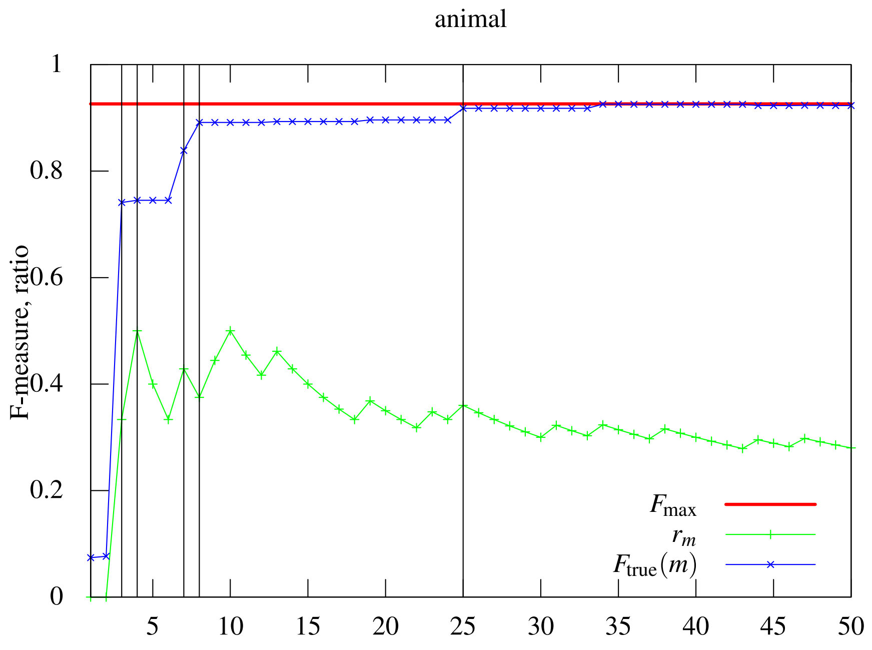


Figure 4.7: Convergence behavior in the active learning of **animal**. The x-axis shows the iteration number m. The values of $F_{\max}$, r_m, and $F^*_{\text{pred}}(m)$ are plotted on the y-axis. Changes in $F^*_{\text{pred}}(m)$ result from adjusting thresholds and switching the best ranked similarity metric (latter changes are indicated by vertical bars).

4.3.4 Experimental result discussion

In this section, we look at the experimental result of one exemplary data set, **animal**, to discuss general observations. On this data set, we observe at first rapid, and then slower improvement until we almost reach the maximal quality (see Figure 4.7). Of the 17 total data sets, 10 yield qualitative similar behavior and another 5 show slower convergence or slight instabilities. Two data sets (UVA, census) show stronger instabilities.

The vertical lines indicate iterations in which the highest ranking similarity measure changes. Typically, this happens frequently in the first few iterations and then only occasionally later.

In Figure 4.9 we can see that generally, empirical F_1 scores $F_{\text{pred}}(m, i)$ tend to decrease over time for all similarity measures. In the end, the highest values are around 0.7 which is far less than the best true F_1 score in that iteration which is above 0.9 (see Figure 4.7). So, the empirical F_1 score is not a direct approximation of the true score but the *ranking* it implies finally yields one of the best solutions in the solution space. The empirical scores are so low because the query strategy samples those string pairs

m	**string 1**	**string 2**	**?**
1	houndfish	Woundfin	✗
2	margate	Margay	✗
3	watercress darter	Darter, watercress	✓
4	Alabama red-bellied turtle	Turtle, Alabama redbelly (=red-bellied)	✓
5	Ash Meadows poolfish	Dace, Ash Meadows speckled	✗
6	Jaguar	Jaguarundi	✗
7	bonytail	Chub, bonytail	✓
8	Red-bellied turtle	Turtle, Alabama redbelly (=red-bellied)	✗
9	White-tailed deer	Deer, Columbian white-tailed	✓
10	Puaiohi	Thrush, small Kauai (=puaiohi)	✓
11	Hawaiian Duck	Hawaiian vetch	✗
12	Red Salamander	Salamander, Red Hills	✗
13	Red-bellied turtle	Turtle, Plymouth redbelly (=red-bellied)	✓
14	bluehead shiner	Shiner, blue	✗
15	blackside darter	Dace, blackside	✗
16	jaguar guapote	Jaguar	✗
17	Long-nosed leopard lizard	Lizard, blunt-nosed leopard	✗
18	Seaside Sparrow	Sparrow, Cape Sable seaside	✗

Table 4.5: Complete query history on dataset **animal** until $TC_m \geq 130$. The last columns show the label: ✓ denotes *match* and ✗ *non-match.*

for which the committee is most uncertain. This means that these must be hard to decide cases.

We can see the sequence of queries in this experiment in Table 4.5. In this example, all pure edit distances (like **Levenstein**, **MongeElkan**) are soon ranked down because they do not have any threshold that correctly separates the non-matches with low distance (queries 1, 2, 6 and later 11) from the matches with relatively high distance (queries 3, 4 and later 9). So the token-based approaches that can swap word order with no or little cost will win. In comparison, on the data set **DBconferences**, the edit distance **MongeElkan** was best at separating the revealed labels. Conference titles usually do not reorder their tokens but only abbreviate or introduce optional tokens, like ordinals ("17th") or affiliations ("IEEE") which MongeElkan with its gaps is well-prepared to deal with.

Figure 4.8 gives us a slightly different view of the experiment on data set **animal**. The shaded area shows the interval between the best and the worst true F_1 score among all prediction functions with the same maximal empirical F_1 score. One graph shows the total number of threshold changes TC_m as a function of the iteration number m. It is clear that many adjustments are done at the beginning. This later becomes less and

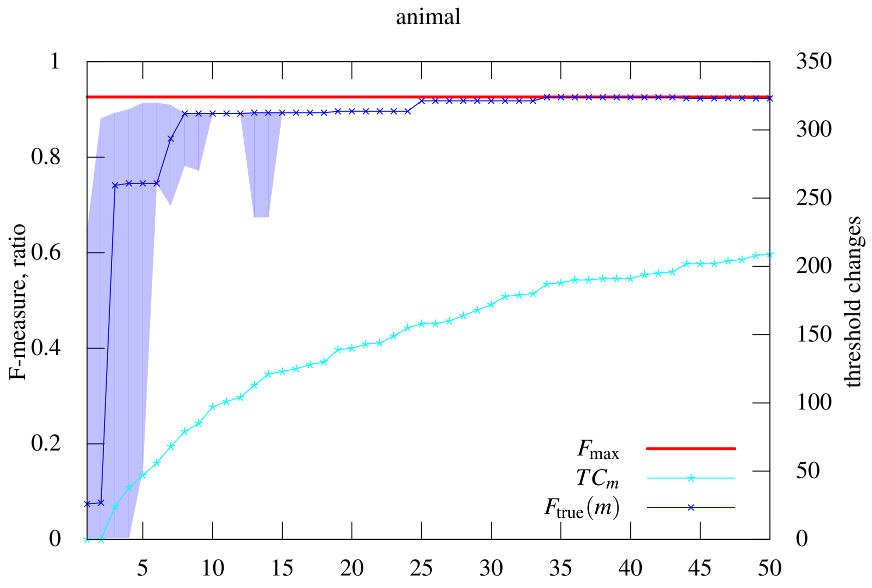


Figure 4.8: The x-axis shows the iteration number m. The left y-axis shows the F_1-scores, while the right y-axis reports the value of TC_m. The line labelled with $F_{\text{true}}(m)$ shows the true F_1-score of the best ranked predictor according to our ranking. The highlighted interval indicates the minimal and maximal true F_1-score among all predictors with maximal empirical F_1-scores.

less. Note that the slope of this curve varies for different data sets which turns out to be a good proxy for measuring the speed of progress.

Figure 4.9 gives us an important view into the state of the Active Learning model; It uses the ranking of the prediction functions by their empirical F_1 score. This measure generally *decreases*, in contrast to the true performance. The reason for this is that the query strategy does not sample randomly which would result in a representative set of string pairs. Instead, its objective is to find those string pairs where the committee is the least certain about. That means that neither pairs that have high similarity values for all similarity measures nor those with only low values will be picked. Those string pairs can be taken for granted and whichever similarity measure is best able to separate the difficult cases correctly will likely be best prepared to separate the whole data. That is why, intuitively, F_1 score on this small, difficult subset is a good proxy for ranking the prediction functions truthfully.

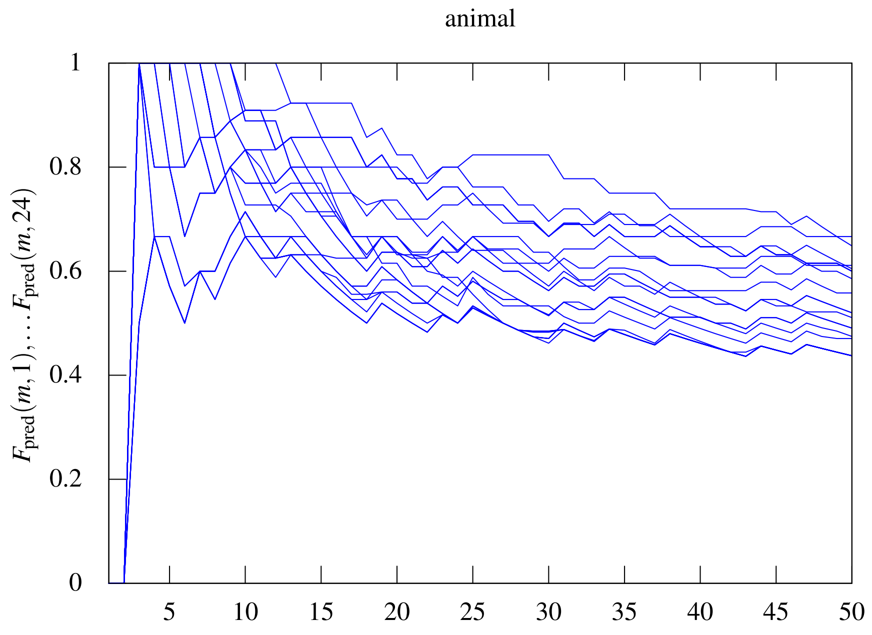
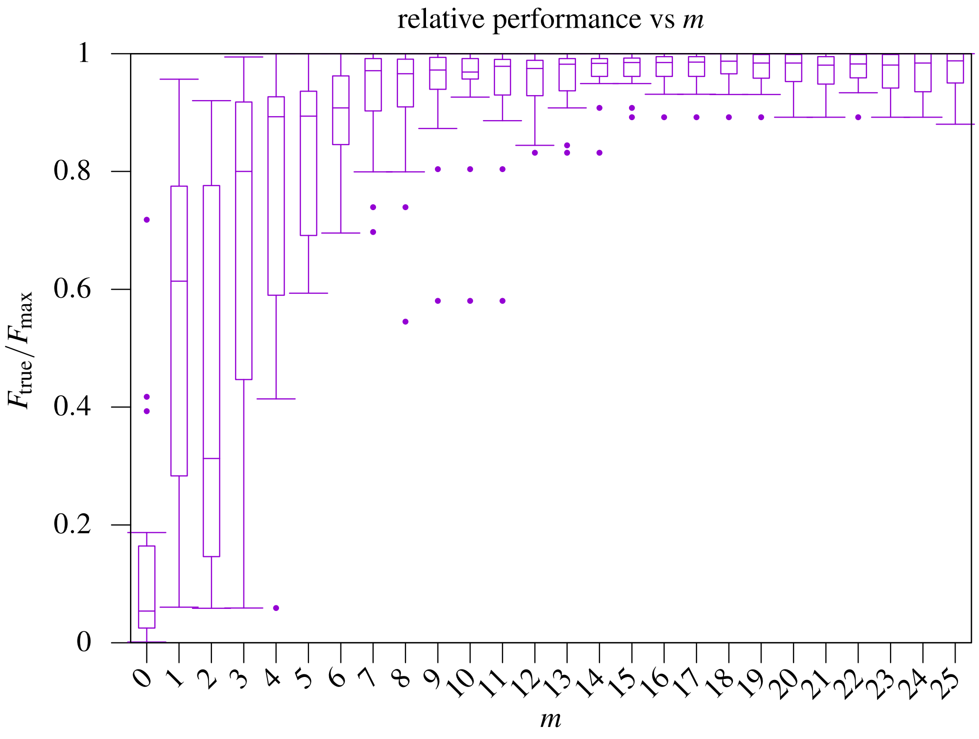


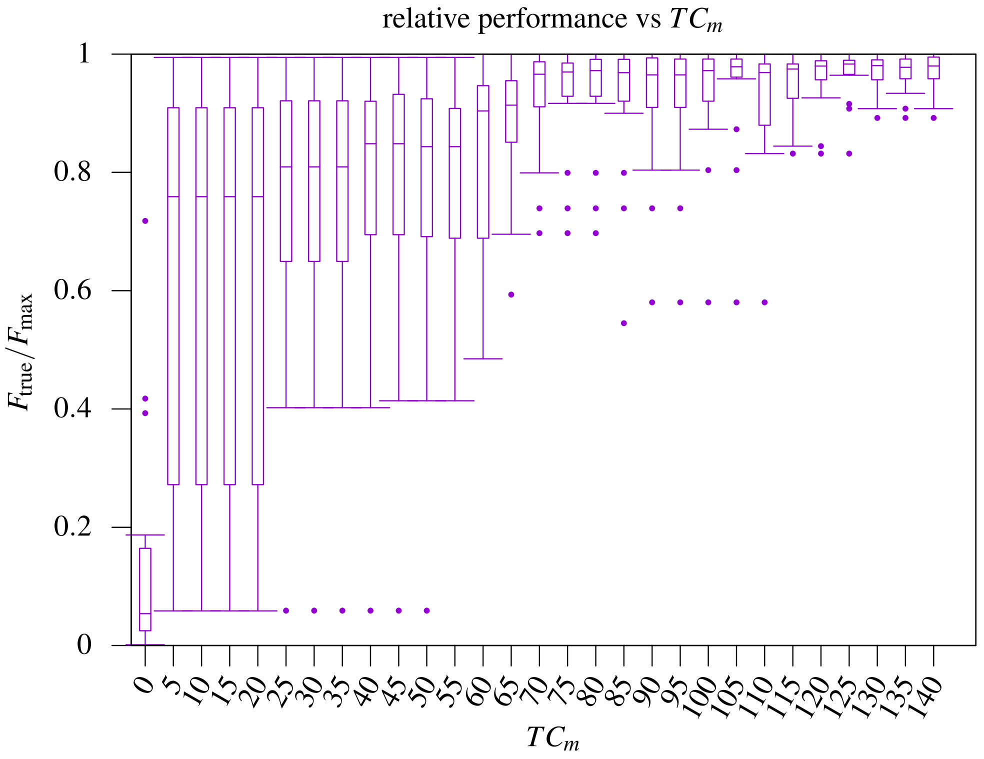
Figure 4.9: The empirical F_1-scores decrease with higher number of iterations.

4.3.5 Stopping criterion

In Figure 4.10 we can see how the quality of the current solution evolves over time. Note that the quality is relative to the maximally achievable quality in the solution space. In Figure 4.10a we see this value plotted as a function of iteration count directly, aggregated over all experiments. We can see that the query strategy rapidly advances the solution by picking informative string pairs for querying. Below, in Figure 4.10b, we see the values binned after passing certain values for TC_m. The same trend is observable here. However, the first very high quality with low variance occurs around $TC_m = 75$, which corresponds to an average iteration count of $m = 7$ (Figure 4.4). A similar performance for uniform m can only be achieved at $m = 9$. So the total number of threshold changes gives us a better measure of progress in our experiments. And of course, once the solutions stabilize, we would observe less and less threshold changes while the iteration count linearly increases.



(a) as a function of the iteration number m



(b) as a function of the total number of threshold changes TC_m

Figure 4.10: Relat. performance of the best predictor aggregated over all data sets

4.4 Discussion

In this chapter, we presented our approach to selecting an appropriate similarity metric and decision threshold for a given string matching problem by requiring only very little human input.

The key ingredient is the committee of basic prediction functions that leverage the in-built mechanisms of the similarity measures to reflect diverse sets of variations in string data. By clustering the similarity measures by their similarity on the unsupervised data, we account for the redundancy in the hypotheses.

The cold start problem (see Section 4.2.3) is addressed by employing a secondary ranking of pairs that allows picking informative samples before any labels have been revealed. Also the self-regulating mechanism driven by the match ratio helps the model to learn quickly. Measuring the total number of threshold changes offers less of a definite stopping criterion but at the same time models the progress that has different speed on different problems.

Our approach could most likely benefit from using logic inference (see Section 2.3.1). It could provide extra information during the learning phase for both matching and deduplication problems and likely would reduce the labelling effort noticeably. During inference, constraint-based matching could be an alternative to the empirical threshold on matching problems. It is plausible that the quality of matchings could benefit from favouring safer matches in cases of logical collisions.

Further improvement is possible by widening the hypothesis space. Adaptive similarity measures could mean that the optimal matching quality would be higher and possibly introduce more variance in the committee. This could be achieved through either parametrized similarity measures (like edit distances) or through more complex solution spaces as in the Program Synthesis paradigm. Our own tentative experiments in this direction (introducing transformations into similarity measure) were not successful. For adaptable and learned similarity measures, refer to the related work in Section 3.1.

5 Metric Learning for Code Clone Detection

We studied the problem of string matching in the previous chapter. Arguably any kind of data that a computers processes can be turned into a string representation (serialization). However, this is not always a natural representation as much of the data's structure might become implicit. For example, a (unlabeled, undirected) graph can be represented by a binary matrix which can be serialized by printing each of its rows in order. Comparing these strings directly as representations of the graphs is an inadequate approach.

In the following we are going to study similarity of data of a complex type, namely programming code. In terms of Metric Learning (see Section 2.3), we will study a non-linear model trained with supervision. We employ the contrastive loss via a Siamese Neural Network.

This chapter is based on a peer-reviewed publication [BA19].

5.1 Background

5.1.1 Code Clones

A *Code Clone* is a code fragment that is the result of copying and pasting an existing code fragment possibly followed by modifications to fit its new context [RBS13]. Generally this definition can be effectively extended to highly similar code fragments that are similar by coincidence. It can happen that two developers write very similar code fragements without copying from each other. This is unavoidable and more likely the shorter the fragment and the looser the threshold for similarity. However, the copy-paste pattern is of more practical relevance.

Using question-answer forums like StackOverflow is an important part of the present day software developer's work and there is evidence that code snippets posted on these platforms are used in software projects directly, many including comments [YMSL17]. A large scale study [LMM$^+$17] has shown that duplication on the file level (on non-forked projects) ranges from 94% (JavaScript) to 40% (Java). Only between a quarter and a third of cases of duplication are properly attributed (when required by the license) [BKD17, BD19]. There is also evidence that some code on forums like StackOverflow itself originates from other software projects [RKP$^+$19].

This illustrates the legal and ethical problems with Code Clones. In addition, there are technical issues with cloning. In the context of a software project, code duplication can increase maintenance cost and can lead to unexpected bugs due to inconsistent changes [JDHW09]. Overall, the answers as to whether cloned code is *harmful* are mixed (summarized in [RBS13]). So motivations to detect code clones include discovering copyright violations and finding refactoring opportunities to make a code base more maintainable. A comprehensive overview of the disadvantages and dangers around code cloning and other applications for detection of code clones is given in [RC07].

Code clones are stratified into 4 types according to a hierarchy of variations they exhibit [CFT93]. The details of their definition have slightly changed over time. A comprehensive overview over Code Clone definitions and taxonomies in the literature is given in [RC07]. An up-to-date typology [RBS13] (sometimes with Latin numbers) is presented here:

Type 1 clones Pairs of code fragments that are identical or differ only in whitespace or comments.

Type 2 clones Also changes in identifiers, literals, types and layout is permissible.

Type 3 clones The code fragments may also differ by added/missing statements (sometimes *near-miss clones*)

Type 4 clones Pairs of code fragments that are only semantically, but not syntactically similar (sometimes *semantic clones*)

The last category of type 4 clones is made up of functionally (or behaviourally) equivalent or similar code fragments, that is, only exhibiting the same or similar input/output relations, without being syntactically similar. Since they cannot be the result of actual *cloning*, ‘type 4 clones’ is a misnomer. A better name for these pairs of code fragments is ‘simion’ (proposed in [JDH10]) which stands for ***simi***lar c***o***de fragme***n***ts. This acronym has been reinterpreted as ***sim***ilar ***i***nput ***o***utput functio***n***s in [MPS20].

It should be noted with regard to simions that, as an exact problem—equivalence of programs—is undecidable[1]. However, Code Clone Detection never aspires to solve this in its entirety. For one, it only deals with real world programs that are limited in their size, scope and nature. For example, many non-sensical codes that have to be

[1]The Halting Problem [Tur37] can be Turing-reduced to this problem.

considered in theory will never be implemented (on purpose)[2]. And even in this limited domain, Code Clone Detection tools are very useful without perfect accuracy.

5.1.2 Code Clone Detection

Code Clone Detection (CCD) techniques are mainly categorized by the representation of code they work with. In the following, we will give a short overview. Note that there are of course also hybrid approaches that will not fit neatly in any of these categories. A good survey over approaches in the following categories can be found in [SK16b]. Code Clone Detection techniques are usually categorized as follows:

Text-based approaches like dude [WM05] or NICAD [RC08] do take code fragments simply as strings of characters. They may use *pretty-printing* – a way of normalizing the non-essential aspects like whitespace.

Token-based approaches use lexical analysis to tokenize code strings, to obtain a string or bag of tokens as a representation. Baker introduced efficient identification of code clones using suffix trees and implemented that in her tool Dup [Bak92]. Several works picked up on this idea (CCFINDER [KKI02], iClone [GK09], SourcererCC [SSS$^+$16]). CP-miner [LLMZ06] is a tool that mines frequent token sequences and SourcererCC [SSS$^+$16] matches bags of tokens to detect clones in a scalable way.

Tree-based approaches will first parse the source code to obtain an *Abstract Syntax Tree* (AST). A pioneering tool, CloneDr [BYM$^+$98], uses subtree matching to detect clones. Hashing the subtrees accelerates the process by avoiding unnecessary comparisons. Similar to token-based approaches, cpdetector [KFF06] considers sequences of AST nodes of the serialized Abstract Syntax Tree. Deckard [JMSG07] aggregates the AST into characteristic vectors whose distances approximate the tree edit distances of the according ASTs.

Metrics-based approaches measure software metrics to create a *fingerprint* of the code fragment as its representation. These can be simple lexical metrics like the number of certain kinds of tokens. But mostly, they are derived from higher level representations of the code; the AST, the *Control Flow Graph* (CFG) or the *Data Flow Graph* (DFG) (or the *Program Dependence Graph* (PDG) which combines both graphs). Such metrics include the number of CFG edges or McCabe's cyclomatic

[2]For example, it is rare to come across a human-written program that does not halt on an infinite class of inputs, unless it is a long-running application (run for its side-effects).

complexity. A very early metrics-based approach is proposed in [Ott76]. It considers numbers of unique operators, unique operands, operators, and operands for a very limited set of programs (the submitted solutions to a homework assignment). A more sophisticate approach is proposed in [MLM96]. It records number of lines of code, number of function calls, metrics based on names, expressions and layout, and simple metrics based on the CFG (e.g., number of independent paths, number of loops).

Graph-based approaches use the PDG as the representation of code fragments. Tools like duplix [Kri01] and gplag [LCHY06] search for similar subgraphs within PDGs to identify code clones.

Another way of organizing the different techniques is through the lens of the different models of similarity that psychological research brought us (see Section 2.1). The spatial model is mostly found in the metrics-based approaches that calculate real-valued vectors as fingerprints that are compared. The contrast model is behind approaches that compare overlap of discrete features that are stripped of their interrelations, like in bags of tokens (e.g., [SSS$^+$16]). The structural alignment model is represented by approaches searching for common sub-graphs in PDGs (e.g., [Kri01, LCHY06]). There is significant overlap with the transformations model which measures string edit distance in text-based approaches (e.g., [DLST04]), tree edit distance in tree-based approaches (e.g., [JMSG07]), and graph edit distance in graph-based approaches (e.g., [PNN$^+$09]).

In this chapter, we study a newer combination of representation and similarity model. We start with the AST representation that is aggregated into a fingerprint by a learned scheme. These fingerprints are then compared in spatial terms. Our approach is published in [BA19].

5.1.3 Vocabulary

When we want to feed programming code as input into a Neural Network, we have to think about the basic units of meaning that make up each code. Programming languages of course do have a fixed set of reserved symbols (e.g., `for`). But there also symbols the users can create themselves. In Java, for example, identifiers like variable names are governed by their own (*lexical*) grammar and are unbounded in length. Natural languages like English are not formally defined[3] and its speakers determine its evolution by their usage in an organic process. Some languages even feature mechanisms of word

[3]nor does it seem to be formally *definable*

formation (productivity) for their speakers to produce one-off words that would be recognized by other speakers as grammatical. In programming languages, the vocabulary is fixed at runtime and saved in a symbol table.

But the vocabulary of programming languages comes with its own difficulties [HD17]. The meaning of a user-defined identifier is not fixed. One user can use it with one meaning, while another uses it with another. The meaning can even change in the same program and the same symbol can represent different entities in different scopes. Nevertheless, symbols are not used quite as arbitrarily. Usually, certain symbols are reserved for special purposes (like single-character loop variables) or they are natural language words or concatenated phrases that “speak” to the reader (e.g., `AbstractStringDistance`).

Vocabularies in natural languages and programming languages are very big and follow a Zipf distribution; Many rare words make up the long tail of the vocabulary whereas a small subset of words make up the bulk of the usage. So the vocabulary of words that may potentially be used is big and not knowable in advance. There are several ways to address this. One is to use characters or sub-words of a fixed maximal length as the basic unit of representation [KBR+20]. That has the advantage that it does not require a fixed definition of permissible words or modelling of the morphology of the given language. And at the same time, one has a small set of basic units representing the sentences. Another approach is to fix a vocabulary and represent all *out of vocabulary* words with a placeholder (`UNK`, for *unknown*, or `OOV`, for *out-of-vocabulary*).

5.1.4 Embeddings

Neural Networks are prepared to take real-valued vector input of fixed dimensionality. For example, fixed-sized bitmap image data is in this format. Anything that deviates from this pattern requires some adaptation. Languages—both natural and formal—come with a lexical inventory as the basic unit that is composed to sequences. These are not easily organized meaningfully in a vector space like colors are in RGB, HSV or CIE color spaces.

A function that does map this discrete set of symbols into a vector space is called an *embedding* or sometimes *distributed representation.* The goal of a good embedding is to associate symbols with vectors such that close vectors belong to similar symbols. In other words, it is to engineer a mapping that realizes the existing similarities in a spatial model (cp. Section 2.1).

It was recognized in linguistic research that contexts of words determine their overall and situational meaning; “You shall know a word by the company it keeps” [Fir57]. More technically, Miller formulated the *Strong Contextual Hypothesis*: “Two words are semantically similar to the extent that their contextual representations are similar” [MC91]. One concrete instance of such contextual representations discussed in the paper are probabilities of co-occurrences with other words. In this instance, words that share a lot of their contextual usage are likely to be semantically similar. Miller also noted, ibd.: “Consequently, measures of contextual similarity based on substitutability come closer to the desired goal. But the disadvantage of measures based on substitutability is that there is no quick and easy computer algorithm for calculating them.”

Early information retrieval encoded word occurrences in documents in a large $m \times n$ matrix for m words and n documents. This allowed comparing words by their respective row vector. Co-occurrence of words over a corpus encodes this information even more fine-grained but results in large sparse matrices since the matrix size grows quadratically with the number of words. Normalization with TF-IDF or Positive Pointwise Mutual Information and smoothing gave good embeddings. And Singular Value Decomposition (SVD) transformed the long sparse vectors into a short dense representation. That enforced generalization and made downstream learning easier.

In 2013, a ground-breaking publication introduced the *word2vec* algorithms [MSC$^+$13, MCCD13]. It builds on the idea of predicting word usage. It does so in two different self-supervised learning tasks: *Continuous bag-of-words* (CBOW), in which a missing word has to be predicted from its context words, and *Skip-gram*, in which the context words have to be predicted from the word in the center.

First, the embedding vectors for all words are randomly initialized. Then, the artificial learning task on the sequences of words is solved. The tasks are defined such that the supervision signal can be constructed from raw corpora (self-supervised learning). The entries of the embedding vectors are optimized as part of the parameters of the Neural Network. The model learns to generalize by exploiting regularities in the data. That is achieved by incrementally positioning embedding vectors of similar words close to each other in the vector space. The part of the network that does the prediction based on the word vectors is discarded after the embedding emerges as a side product of the learning task.

The resulting word embeddings have shown to capture a lot of semantics as witnessed by solving analogy tasks through simple vector space arithmetic [MYZ13, MSC$^+$13, MCCD13]. However, if words have polar opposite meaning, it is likely that they share a lot of their contexts too (antonyms; e.g., the words “weak” and “strong”) [MC91]. As a result, words with opposite meanings can have very similar vectors in a word embeddings based on co-occurrences in neighbourhood windows [SPH$^+$11, WRP19].

The details of our variant of word2vec are explained in the next section. For related work on embeddings in Software Engineering applications, refer to [CM19, KBR+20].

5.2 Approach

Our approach to Code Clone Detection is tree-based, that is, it takes in code represented by its Abstract Syntax Tree. The nodes in this tree are combinations of discrete tokens which we represent by a learned vector representation. Then, a Recursive Neural Network aggregates the AST from the leaves to the root so that we are left with a single vector of fixed length representing the whole code fragment. These vectors are then compared as a proxy for the code they represent. In this section, we go into detail about what the different concepts mentioned mean, how we put them to work, and what details have to be considered on the way.

5.2.1 Node embeddings

Word2vec denotes two self-supervised algorithms that learn to model co-occurrence of words (see previous Section 5.1.4). We will explain the variant which we employ further, which is called *skip-gram*.

The skip-gram objective is to predict for a given focus word, if a context word likely occurs in the close neighbourhood of the focus word. The focus word and the context word are represented by randomly initialized embedding vectors which are compared by their dot product. There is a separate lookup table of vectors for the focus word and the context word – otherwise the dot product of each word vector with itself would always be the square of its length.

The embedding vector dot products for one focus word with all words in the vocabulary are then translated into a probability distribution by the softmax function. The objective is to maximize the probability of observed co-occurrences within small context windows. The corresponding loss function is usually computationally infeasible because of the size of the vocabulary. One way of approximating this is negative sampling [MSC+13], in which contrasting negative examples of co-occurrences that occur nowhere in the training corpus are synthesized.

Interestingly, word2vec with skip-gram and negative sampling can be straightforwardly conceptualized in terms of Metric Learning. Word2vec's (instrumental) objective is to learn not a distance function but another very similar binary relation – co-occurrence. In terms of Metric Learning it then uses contrastive loss to learn the co-occurrence relation. In [DS17], this concept is applied to the paraphrase relation for n-grams and in [KBdR16] to co-occurrence of sentences. It was also shown that word2vec skip-gram with negative sampling is equivalent to factorizing a word-context matrix of pointwise mutual information [GL14].

We train two separate embeddings; one for node types and one for node contents. We make a small adjustment to the normal word2vec skip-gram with negative sampling algorithm[4]. Since we deal with a formal language, we know that word usage is more regular and constrained by the Java grammar.

For the embedding of node contents, we fix the context window for co-occurrences to two tokens both before and after the focus token. In the case of node types, these do not occur in sequences. Instead of a sequential context, we therefore define the types of parent and children of the according node as the local neighbourhood. We reserve a separate embedding for each of these roles and aggregate the three dot products by learned affine linear transformation into a single scalar before finally applying the sigmoid non-linearity.

5.2.2 Recursive Neural Network

A *Recursive Neural Network* really does not refer to any specific type of network, but rather to the mode of evaluating it on its input. The most well-known use case for Neural Networks assumes an input of fixed size, which is connected to artificial neurons, which are connected to another stratum of neurons, and so on, until a final layer of neurons computes the output. This static setup is called *Feedforward Neural Network*.

In contrast, a Recursive Neural Networks operates like a Feedforward Neural Network on *some part* of the input. Once that computation is done, the output of the Neural Network is taken as additional input when evaluated on *some other part* of the input that is connected to the first part of the input. So the Neural Network traverses the input, is evaluated many times, and feeds its own output into itself. In a Recursive Neural Network, the input is assumed to be tree-structured and evaluation starts at the leaf nodes. Therefore, this mode of evaluation is well-defined and terminates. A special case of this is when the tree of input happens to be just linear. The mode of evaluation in this special case is described with the moniker *Recurrent Neural Network*.

Usually, a Neural Network only takes in a fixed sized input per evaluation[5]. Therefore, Recursive Neural Networks have to be designed to take in the maximal number of children

[4]Based on the implementation in `https://github.com/yoonkim/word2vec_torch`

[5]Note that this paradigm is broken by the recent advent of the highly successful attention layers [VSP+17].

seen in the data or discard or aggregate the children. We perform a transformation of the input trees to make sure that they are binary (see Section 5.3.1).

The field of Natural Language Processing deals with much less well-defined data. It has been shown that current state-of-the-art Transformer language models [RYW+19] and also (recurrent) LSTMs [STSC19] that are merely trained on self-supervised signals from raw data implicitly learn the syntactic structure of natural language sentences. In formal languages, we can assume this syntactic structure and exploit it directly, for example, by using a Recursive Neural Network.

5.2.3 Long Short Term Memory

The most popular and successful method of setting the weights in a Neural Network is by optimizing some objective function on its output by Gradient Descent. This is mostly achieved by *backpropagating* the gradient of a loss function to all the weights involved in the Neural Network. This not only works for Feedforward Neural Networks, but can also be extended to iterative evaluations in a Recurrent Neural Network (‘backpropagation through time’) and to recursive evaluations in a Recursive Neural Network (‘backpropagation through structure’).

Once networks become very deep, it is easier to encode very complex functions. But another consequence is that it is easier for gradients to either vanish or explode [HBFS01]. In Recursive Neural Networks, this can easily happen because the input structure coincides with that of the computation graph. If the input is a deep tree, this can make an otherwise shallow network effectively very deep.

Long Short Term Memory (LSTM) are a special kind of Neural Network unit that can be used in the composition of any kind of Neural Network. They have feedback connections that realize a gating mechanism that can leave the error gradient unchanged over longer distances (i.e., in RNNs). This way, long distance dependencies in the input can be learned and the problem of vanishing and exploding gradients is lessened.

We modify the n-ary tree-structured LSTM (conceived for constituency trees in Natural Language Processing) to accommodate our specific AST nodes. We also implemented the stacked [GJM13, SVL14] and nested [MK17] variants of the LSTM unit, but these were slower and did not show performance gains.

5.2.4 Siamese Neural Network

A *Siamese Neural Network* [BGL+93] (also known as *Twin Neural Network*) is almost akin to a Recursive Neural Network as it describes mostly a mode of evaluation of a given Neural Network. As the name suggests, the input is composed of two input parts of the same type. The Neural Network is then evaluated on both input parts. The two outputs are then further processed by a shallow Neural Network that conjoins the computation graph.

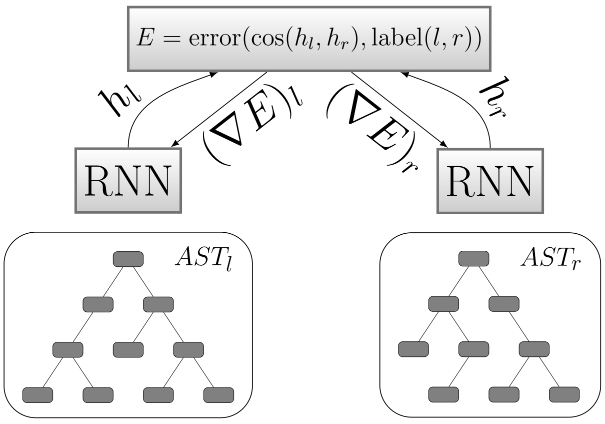


Figure 5.1: The Twin Neural Network during training

Forward pass

Putting it all together, the Neural Network operates as follows. The Recursive Neural Network is just a simple LSTM unit that is evaluated at all (inner) nodes of the AST. It receives six input vectors (see Figure 5.2). Firstly, the hidden states h_l, h_r and the cell states c_l, c_r of the left and right child. If a child is missing, or is a leaf node, the corresponding hidden state and cell state are set to zero. Secondly, the embedding vectors representing the node type (x_t) and the node content (x_c) of the present node.

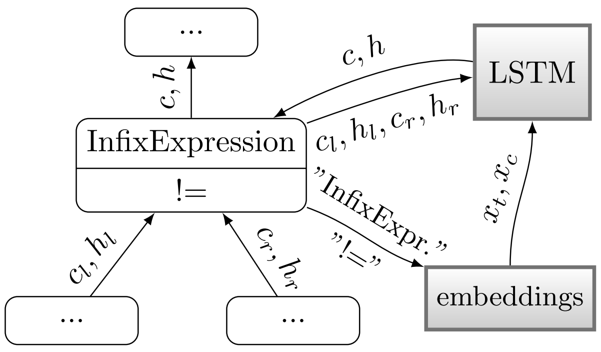


Figure 5.2: Flow of data during evaluation of the RNN

These are pretrained and are simply looked up in a fixed table. If there is no node content for the present node, x_c is set to zero.

The RNN traverses the inner nodes of the AST. The input tree can be assumed to be binary (see Section 5.3.1). The RNN can only be evaluated at nodes where all children have already been processed or do not need processing. Hence, the network starts at the nodes that are parents only to leaf nodes. It continues to evaluate on nodes for which each child is either a leaf node or has been traversed already. Lastly, it is evaluated at the root node. The LSTM hidden state in this evaluation represents the whole AST.

The following equations describe how the six input vectors c_l, h_l, c_r, h_r, x_t and x_c compute the next hidden state h and cell state c. Here, i, f_k and u are the input, forget and update gates of the LSTM, respectively:

$$i = \sigma\left(\sum_{p=t,c} W_p^{(i)} x_p + \sum_{p=l,r} U_p^{(i)} h_p + b^{(i)}\right) \tag{5.1}$$

$$f_k = \sigma\left(\sum_{p=t,c} W_p^{(f)} x_p + \sum_{p=l,r} U_p^{(f)} h_p + b^{(f)}\right), \tag{5.2}$$

$$\text{where } k \in \{l, r\}$$

$$u = \tanh\left(\sum_{p=t,c} W_p^{(u)} x_p + \sum_{p=l,r} U_p^{(u)} h_p + b^{(u)}\right) \tag{5.3}$$

$$c = i \odot u + \sum_{k=l,r} f_k \odot c_k \tag{5.4}$$

$$h = \tanh(c). \tag{5.5}$$

Figure 5.3 makes above equations more accessible. LSTM units are often defined including an additional output gate as shown in the diagram. We found in our preliminary experiments that this simpler variant without an output gate performs better. In the diagram this is equivalent to setting the gate o to the constant function $o \equiv 1$. Our implementation is based on that of [TSM15][6].

The evaluation of the RNN is done for two AST trees independently. Their hidden state vectors are then fed into a final layer that completes the Siamese Neural Network (see Figure 5.1).

[6]available at `https://github.com/stanfordnlp/treelstm`

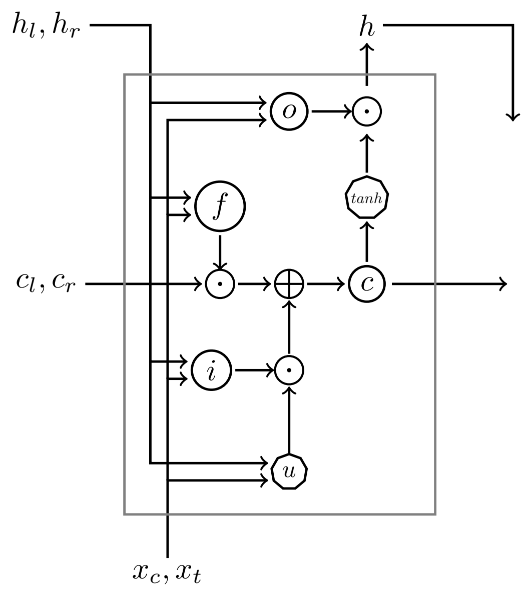


Figure 5.3: The internals of the LSTM unit. Diagram modified from [SB18]

Backward pass & weight update

The shallow network conjoining the two copies of the Recursive Neural Network is simply computing the cosine similarity between the two input hidden states h_l and h_r representing the two trees AST_l and AST_r. Comparing the similarity value s to the ground-truth label l, we define the error of the prediction as follows:

$$\text{error}(s, l) = \begin{cases} 1 - s & l = \text{clone} \\ \max(0, s - m) & \text{otherwise} \end{cases} \tag{5.6}$$

where $s = \cos(h_l, h_r)$ and $l = \text{label}(l, r)$. By default, the margin m is set to 0. We explore other values for m in the evaluation (see Section 5.3.6).

The error is back-propagated through the whole network. For the Recursive Neural Network, this means back-propagation through structure [GK96], that is, recursive back-propagation in reverse order of evaluation. It is noteworthy, that there is a lot of weight sharing in the evaluation of the whole network. Firstly, both copies of the RNN share their weights. Secondly, the LSTM unit making up the RNN shares its weights across all recursive evaluations. The top layer of the Twin Neural Network does not have any weights. The weights of the pretrained embedding layer are of course shared across all nodes.

These forward and backward passes are repeated for many AST pairs and the RNN averages the error gradient for each weight of the LSTM across evaluations in order to prepare a gradient descent step to minimize the prediction error. In Section 5.3.2 we explain the details of the optimization.

5.3 Evaluation

5.3.1 Data

We want to use supervised learning to train our Neural Network to match code fragments. As mentioned in Section 2.3, Metric Learning does not consider data instances in isolation, but tries to model their (binary) similarity relation. As such, our data consists of pairs of code fragments.

As a source of ground-truth data, we use BigCloneBench [SR15b, SR16]. It was mined from IJaDataset 2.0, a large repository of Java source code from 25,000 software projects. It contains method-level clones and non-clone pairs, which have partly been verified as such by human experts.

The name *code clone* suggests that these instances can only come into existence by (legitimate or illegitimate) copying and modification of existing code fragments. As laid out in the definition (Section 5.1.1), this meaning is only the template for a more generic notion that also allows for parallel development. Ultimately, the fact of cloning can often not be tracked after the fact and is only important for legal and ethical concerns but less so for technical ones.

The authors of BigCloneBench therefore construct their data set by building equivalence classes of methods that all implement the same distinct "functionality". Note that this circumvents the problems of asymmetry that come with priming effects in comparisons by humans (Section 2.1) and results in a transitive clone relation by construction. Version 2 of BigCloneBench contains 8.6 million clones from almost 15,000 methods implementing 43 target functionalities like "Decompress zip archive", "Connect to Database" or "Setup ScrollingGraphicalViewer Event Handler". Heuristics were employed to search for candidate methods that might implement a given target functionality. Human experts then review these candidates by confirming true positives and flagging false positives.

BigCloneBench was not intended for the training of supervised learning approaches, but many papers about supervised learning of Code Clone Detection use the supervised data for both training and testing [SK16a, LFZ$^+$17, WL17, ZH18, ZWZ$^+$19, BA19]. In

[WTVP16] it is exclusively used for training and only for testing in [TWB$^+$18]. To use this data set for training *and* testing comes with a specific pitfall, because the instances are *pairs* of methods. We describe this in detail in Section 2.3.2.

Data preprocessing

Initially, there are 43 clone clusters of sizes 9 to 3,055 (average 349, median 197). We take these 14,992 methods that have been confirmed to implement one of the target functionalities and filter them for several reasons.

Tree filtering The first filtering criterion is the labelling quality. We only allow those methods for which the clone pair with other methods within the same cluster has been judged to be a real clone with minimal confidence of 2 or greater. This confidence is defined as the difference between the pro and con votes by the experts for a given method pair to be clone. This ensures that we only train and test on high quality data. This leaves 4,408 methods in 36 clusters. Furthermore, we remove clusters that have less than 5 methods. This affects 3 small clusters with a total of 9 methods.

We then remove trees that are very big. We draw the line at trees with more than 1,000 nodes or depth greater than 28, because of limitations of our implementation. This affects only a small part of observed methods; 94 because of the depth constraint, 48 because of size, and 85 because of both, for a total of 227.

Finally, we identify pairs of method that have isomorphic AST trees, and retain only one per AST. This leaves us with 609 methods across 33 clusters. This step prevents us from comparing trees that have an identical representation. On one hand, this speeds up training. On the other, it leads to an underestimation of the overall performance in testing because these easier code clones are missing there too. The vast majority of clones (93.3%) among our method pairs are Type-4 clones as per BigCloneBench's internal definition.

Node modification We parse all methods to AST trees with the Eclipse Java development tools (JDT). Each node in a tree has a specific *node type*. These have names like "MethodInvocation", "Modifier", "FieldAccess" or "StringLiteral". There is only a relatively small set of node types and it is determined by the particular parser.

Some of the nodes have *node content* in addition to a type. These are very broad in range. Some are part of the language specification like "true". Others, like identifiers, adhere to their own (lexical) part of the Java grammar. A set of frequent values are determined by the authors of standard libraries, for example, "LinkedList" or "println", while others are defined by the user. Finally, the content of StringLiterals is basically arbitrary, for example, “Hello, World!”.

We make two changes to the nodes of the tree. First, we delete the particular nodes that holds the method name of the method the AST is representing. This is to ensure that the model cannot leverage this information, but has to instead rely on the actual body of the method. It is worth highlighting that this incurs an artificial disadvantage. One would choose to keep this information in a real world implementation. But we want to study how well Code Clones can be detected just by aggregating method bodies.

Second, we artificially limit the vocabulary of the node content. The node content frequencies follow a Zipf distribution. There are a few contents that make up the bulk of the occurrences like “double” or “false”. And then there is the long tail of singular contents like “jdbc:postgresql://localhost/test”. Overall, we observed 32,000 distinct content values where 21,000 occurred only within a single method each (29,000 only within the same cluster). Another 5,000 values where unique to exactly two methods. We only keep the content values that occur in at least six clusters and in at least 50 clone pairs.

After this filtering of the node content vocabulary, we are left with 1032 relatively frequent to highly frequent values. This allows us to deal with a node content embedding that has manageable size. Furthermore, it prevents overfitting – the model cannot pick up on spurious patterns related to one-off contents. The remaining less frequent content values are mapped to the value *<UNK>* for *unknown*.

Tree modification In an Abstract Syntax Tree, the maximal branching factor of a node can be arbitrarily big. However, our Recursive Neural Network only accepts a fixed-sized input. We use the N-ary Tree-LSTMs[7] proposed in [TSM15], for $N = 2$. We modify their implementation[8] to accommodate node type and content as input at every node. Note that [TSM15] also proposes a Child-Sum Tree-LSTMs that accepts arbitrarily many children but it discards ordering.

In order to make general ASTs consumable by the binary RNN, we transform them by what we might call *balanced factorization*. Each node that has more than two children, will get one or two artificial children that will take over the children in its stead. We first split the set of children in half. If it is an odd number, we make the first half bigger (counting from left). We assign the left half (which must be greater or equal to two) to an artificial new intermediate child. The right half might be one, in which case we

[7]In more standard terms, Tree-LSTM would be called a Recursive Neural Network with a LSTM unit.
[8]available at `https://github.com/stanfordnlp/treelstm`

```
public static void main(String args[]){
    System.out.println("Hello, World!");
}
```

Figure 5.4: *Hello World* code

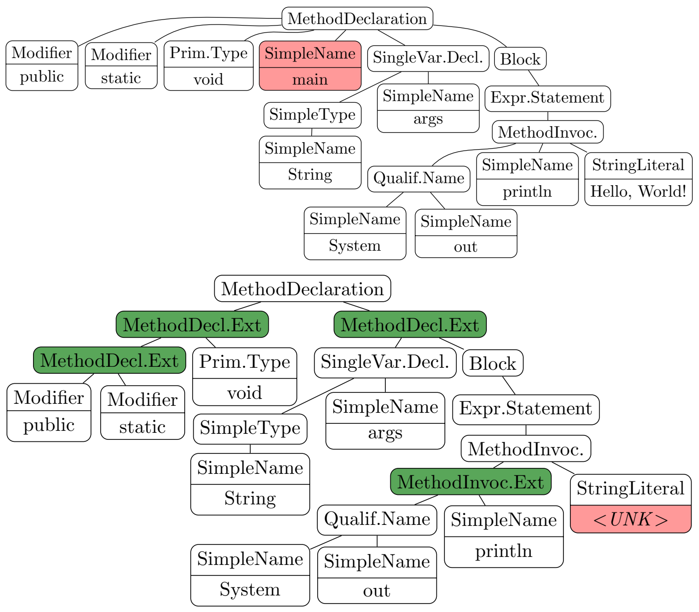


Figure 5.5: AST tree of *Hello World* code, before and after preprocessing

stop. If not, all these children are assigned to an artificial new right child of the original parent node. We repeat recursively until no set of children is greater than two.

For example, for node "MethodDeclaration" in Figure 5.5, we have to do two such recursion steps and introduce a total of three artificial children. We introduce new node types for these artificial new nodes and we denote them as an "extension" of the original parent node (e.g., "MethodDeclarationExtension" in the example). By this mechanism, we introduce additional 25 to the existing 68 node types, for a total of 93. Note that in addition to creating more nodes, this operation makes the AST trees deeper. In general, it increases paths through a node with originally n children in length by $\lceil \log(n) \rceil - 1$ to $\lceil \log(n) \rceil - 2$. See Table 5.1 for how this transformation influences the statistics of tree sizes and depths. Note that these values were measured on all parsed trees since we did the selection of the final 609 trees partly on these values.

	node counts		tree depths	
	original	**binarized**	**original**	**binarized**
min	13	15	5	7
5%	43	50	7	10
25%	80	93	9	13
median	130	151	11	16
average	203.98	240.66	11.81	17.16
75%	220	259	13	20
95%	577	681	19	28
max	6,554	8,117	55	70

Table 5.1: Statistics of tree sizes before and after binarization

5.3.2 Training

The general training procedure is to repeatedly compute the gradient of the weights of the LSTM unit with respect to the prediction error and update the weights using this information. The forward and backward pass through the Neural Network is outlined in Section 5.2.4. The one input to the error function is defined by the cosine distance in the Siamese (or Twin) Neural Network [BGL+93] (see Figure 5.1). The other input is the label according to the ground-truth. The distance is then compared to the ground-truth label (see Equation 5.6). A loss function that is defined to target low distance for matching pairs and high distances for non-matching pairs is sometimes called *contrastive loss* [HCL06].

Data split

We emphasized in Sections 5.3.1 and 2.3.2 that supervised data for Code Clone Detection (or Metric Learning more generally), has to be split along individual instances (or, if available, clusters) to produce performance measurements that will generalize[9]. Therefore, we divide the 609 methods in the 33 clusters as whole clusters. We use roughly 1/6 of clusters each for validation and testing (concretely 6 out of 33, or 18%) and the other $\sim 2/3$ (concretely 21/33 or 64%) for training (see Table 5.2).

The last three columns show the resulting numbers of code fragment pairs; clones at the top and non-clones at the bottom. Note that the ratios of cluster numbers do not reflect the ratios on the level of pairs of code fragment. The reason is that the cluster sizes are not uniform and the number of pairs within a cluster (clones) and between clusters (non-clones) do not scale linearly with the involved cluster sizes. We employ

[9]In Section 5.3.6 we demonstrate what happens if data is split in the naïve way.

	Clusters			**Resulting Pairs**		
Fold	**Training**	**Valid.**	**Test**	**Train.**	**Valid.**	**Test**
0	4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 29	39, 43, 45, 42, 41, 38	33, 32, 30, 31, 34, 36	3,277 64,619	1,140 6,000	1,140 6,000
1	4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 31, 32, 33, 34, 36, 38, 39, 41, 42, 43, 45	23, 30, 29, 26, 25, 24	22, 21, 20, 19, 17, 18	3,482 69,289	1,070 5,600	1,005 5,100
2	18, 19, 20, 21, 22, 23, 24, 25, 26, 29, 30, 31, 32, 33, 34, 36, 38, 39, 41, 42, 43	11, 12, 13, 14, 15, 17	4, 5, 6, 7, 10, 45	3,920 82,400	841 4,109	796 3,482

Table 5.2: Split of supervised data in training, validation and test set of clusters in three ways for cross-validation

3-fold cross-validation and therefore prepare three different splits of the data (one per row in Table 5.2) where no two validation sets or two test sets overlap.

Error scaling

In Section 2.3.1 we described how Metric Learning comes with a highly imbalanced class distribution and that this poses challenges during training. Over- or undersampling can help in two ways. It increases efficiency because the overall computation is reduced. And it ameliorates the class imbalance in the training data. Sampling can also have downsides. For example, undersampling reduces the amount of information used and oversampling may cause overfitting.

Sampling is not quite as efficient for us. The aggregation of the tree is the computationally expensive part. On the other hand, it only grows linearly in the number of trees. Sampled pairs of trees always involve two trees but in an incomplete sample each tree might be considered only a few times. So it is actually prudent to take advantage of all information by aggregating all trees and doing all comparisons. Therefore, we do full gradient descent steps as opposed to stochastic or mini-batch gradient descent.

However, the problem of class imbalance remains. The area of *cost-sensitive* learning is concerned with ideas that address this issue. In [KK98], these were first extended to Neural Networks. Among others, it proposes what we call *error scaling*. We divide the error contributed by negative instances (non-clone pairs) by the ratio of number of non-clones to clones to equally weight the errors of both positive and negative instances.

Hyperparameters

There are a few hyperparameters that have to be set to define the Neural Network and the training process. We largely follow the settings in the implementation of [TSM15]. The dimensionality within the LSTM is 150 per default and we explore other values in Section 5.3.5. The 1,033 node contents are represented with 10D embedding vectors, and the 93 node types with 4D embedding vectors. They are pretrained as described in Section 5.2.1.

We minimize the error by adjusting the weights based on the gradients with the *AdaGrad* ("**ada**ptive **grad**ient") heuristic [DHS11]. Note that we make full gradient descent (as explained earlier in this Section) whereas [TSM15] applies mini-batch gradient descent. We use a base learning rate of 0.05 for the LSTM unit and 0.1 for the pretrained node type and content embeddings.

We use the standard initialization of weight matrices of torch7, which initializes weights and biases of a linear transformation with random values from a uniform distribution over $[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}]$, where n is the output dimension. We also implemented the Xavier (Glorot) [GB10], Kaiming [HZRS15], and chrono[10] [TO18] initialization schemes. However, they did not turn out to give improved performance in exploratory experiments. Maybe this is due to the fact that these schemes were conceived of in the context of Feedforward and Recurrent Neural Networks that do not recursively read tree-shaped input.

We use the validation set to determine the best performing weights within the first 500 epochs. We exclude the very first 100 epochs because we observed some chaotic jumps in the performance on the validation set during the first few dozen epochs.

5.3.3 Quality metric

We use the area under the ROC curve (AUROC; see Section 2.3.3) to measure the quality of our models. By definition it is independent of any decision threshold. So we can measure the quality of the vector representation as such. In other words, how well does the similarity of the vectors reflect the similarity of the code fragments they represent? Concretely, AUROC can be interpreted as the probability of ranking any two given pairs—one matching and one non-matching pair—correctly. We train our network with contrastive loss. However it is possible to directly optimize for AUROC by using histogram loss [UL16].

[10]The name "chrono" comes from its initial use on sequential data.

Note that we systematically underestimate our real performance since the test pairs are already filtered to not contain any Type-1 clones. These would be trivially matched since Abstract Syntax Trees of Type-1 clones are identical. However, the BigCloneBench data contains mainly Type-4 clones (93.3%). See Section 5.1.1 for a definition of code clone types.

5.3.4 Baselines

We compare the recursive aggregation of embedded AST nodes against two baselines. Both are vector representations of AST trees but they are not finetuned for the objective in the same way.

The first baseline is the averaging of all node vectors (concatenation of node type and content vectors). These vectors *are* the result of a learning process but it was an unrelated, self-supervised task and the vectors mere by-products of that learning loop. We compare these fixed vectors by cosine similarity. This approach has several advantages as it requires low computation effort, is easy to implement and does not need any supervised data.

The second baseline is a specialized vector representation for Code Clone Detection. It is the vector representation calculated by the well-known *Deckard* CCD tool [JMSG07]. These vectors are not the result of machine learning but of a handwritten algorithm backed by theory. Deckard's complete algorithm approximates tree edit distance. However, we extract the vectors from the complete Deckard pipeline (that includes blocking and locality-sensitive hashing) to get a vector space representation that we can use for comparison. A small number of trees (5 out of 609) could not be parsed by Deckard. Normally, the vectors in the Deckard approach are compared by the Euclidean norm, but we observed that cosine distance gives better results.

Note that Deckard is designed to find code fragments of varying granularity, not only methods. For this comparison, we only consider fragments from BigCloneBench, which are whole methods. And as we already dismissed duplicate trees (in the sense of isomorphism), we under-estimate both baselines and the main approach because the easier code clones are missing from training and test data. Both baselines are unsupervised and as such do not require a training phase. We still evaluate them on the three test sets to get comparable performance measurements.

	dim.	#params	AUC$_0$	AUC$_1$	AUC$_2$	∅AUC
	300	752,294	0.799	0.771	0.903	**0.824**
	200	345,094	0.766	0.861	0.876	**0.834**
	150	201,494	0.764	0.841	0.929	**0.845**
	100	97,894	0.690	0.840	0.857	**0.795**
	50	34,294	0.613	0.817	0.793	**0.741**
	30	20,054	0.588	0.768	0.783	**0.713**
Averaging embeddings	14	10,694	0.784	0.752	0.885	**0.807**
Deckard, cosine	300	–	0.565	0.746	0.798	**0.703**
Deckard, Euclidean	300	–	0.543	0.653	0.602	**0.599**

Table 5.3: The performance with respect to different dimensionalities; performances of baselines

5.3.5 Influence of network layout

In the following we show the performance of the Neural Network model and how different changes in its definition influence the performance. We report the AUROC value for each cross-validation fold and the arithmetic mean.

Influence of dimensionality

A very obvious and important hyperparameter is the dimensionality. In Table 5.3 we report the different results for various values. This determines the size of the hidden and cell states h and c in the LSTM and consequently the sizes of weight matrices $U_p^{(*)}$ and bias vectors $b^{(*)}$ in the different gates. The parameter count in the matrices grows quadratically with this value and hence have great influence in the total count. The overall number of parameters is listed in the second column of Table 5.3. The sizes of the node embeddings are fixed (4 for types and 10 for contents). Their weight matrices amount to 364 parameters for the 4D embedding of 91 node types and 10,330 parameters for the 10D embedding of the 1,032 node contents and the *<UNK>* token.

Generally, more parameters improve the ability of the network to learn to rank the code fragment pairs properly. The models achieve values of 0.92 to 1.00 on the training data. Beyond 150 dimensions, the test performance slightly decreases. This is possibly due to overfitting.

We can also observe that the performance is different on the three cross-validation folds. Generally, the last fold yields the best performance. One possible reason for this that comes to mind is that in this fold, the training set is significantly bigger in terms of code fragment pairs (see Table 5.2). However, this reason can be dismissed because the unsupervised baseline of averaged embeddings also performs much better on this fold. It might just be that the clusters in the test set of this fold make for particularly easy cases of clones and non-clones.

The baseline of averaging embeddings performs remarkably well. Interestingly, it is able to outperform the learned aggregation up until the dimensionality of 100. There are easy parameter settings of an LSTM unit (with ≥ 14 dimensions) that would effectively result in averaging the node vectors. However, the LSTM unit does not seem to be able to learn this pattern – or at least not through gradient descent and with our particular optimization heuristic. We describe one weight setting that would achieve the averaging in the following for completeness. The equations 5.1-5.5 define the gates and their interactions. Figure 5.3 summarizes the equations in a diagram.

The matrices $W_p^{(f)}, U_p^{(f)}, U_p^{(i)}, W_p^{(u)}, U_p^{(u)}$ would be set to zero. The matrix $W_c^{(i)}$ would in turn be set to a zero matrix with a 10×10 unit matrix at the top, scaled by some $b \in \mathbb{R}$. Similarly, $W_t^{(i)}$ would be zero almost everywhere with a 4×4 unit matrix at the bottom and scaled by the same factor b. If the dimensionality of the LSTM unit is greater or equal to 14, these two parts will not interact downstream. The biases would be $b^{(f)} = (1, \ldots, 1)^T, b^{(i)} = (0, \ldots, 0)^T$ and $b^{(u)} = (B, \ldots, B)^T$, for some $B \in \mathbb{R}$. This would deactivate forgetting and allow the input through scaled by a constant, but otherwise unaltered[11]. The tangens hyberbolicus involved in computing c from u and h from c introduces a slight problem in that it would distort the values. However, the value for B can be chosen big enough to get arbitrarily close to the output 1 for the update gate. And if we choose the value b to be small enough, then sums of node vectors for trees with a certain maximal number of nodes would be placed in the approximately linear regime of the non-linearity tanh. Finally, an LSTM unit with these weight settings would compute the sum of all concatenated node embedding vectors, scaled by b. This scaling factor (and the missing division by the total number of nodes) is irrelevant, as the resulting ranking of pairs of code fragments would be the same as that for node vector averaging.

Influence of embeddings

One important ingredient in our model is the setting of the parameters in the node type and node content embeddings. We pretrained them in a self-supervised way hoping that this would capture some of the regularities and therefore help the model to generalize. To test just how useful the embeddings are, we run an experiment where we use the

[11]Note that here, our LSTM unit does not have an output gate.

	#params	**AUC$_0$**	**AUC$_1$**	**AUC$_2$**	**∅AUC**
no change	201,494	0.764	0.841	0.929	**0.845**
10×neg. scal.	201,494	0.744	0.817	0.912	**0.824**
margin $m = 0.9$	201,494	0.750	0.793	0.885	**0.810**
Averaged embeddings, 14D	10,694	0.784	0.752	0.885	**0.807**
margin $m = 0.5$	201,494	0.726	0.813	0.856	**0.798**
with output gate	249,194	0.700	0.763	0.822	**0.761**
no pretraining	201,494	0.668	0.807	0.763	**0.746**
*Cluster-unaware split**	*201,494*	*0.990**	*0.997**	*0.991**	***0.993****

Table 5.4: The performance for dimensionality 150 with respect to other changes

randomly initialized embeddings before the pretraining instead. Table 5.4 shows that the average AUROC score drops remarkably from 0.845 to only 0.746. We can also see that this performance is below that of the averaged pretrained embedding vectors.

We can conclude that embedding vectors are very important because embedding vectors that are trained without supervision beat a complex network trained with initially random embedding vectors. A recent paper [WK19] in Natural Language Processing comes to similar conclusions. It showed that pretrained word embedding vectors with random aggregations via an untrained LSTM or other randomly parametrized models, could rival state-of-the-art sentence embeddings.

We would like to offer some intuition as to how pretrained embeddings may be useful. Word2vec results in token vectors that are very similar for tokens that share a lot of their contexts. This is especially true for user-defined symbols that are naturally used interchangeably, like “dir” versus “directory” or “i” vs “j”. If each of these tokens was represented by a random vector, a network would have a hard time picking up that the meanings of both of these pairs of tokens are in fact very similar. The word2vec pretraining, however, provides the network with an important advantage. And any confusion of tokens that is not helpful (e.g., similar vectors for the tokens “true” and “false”) could be ameliorated by the finetuning during supervised training.

And why is the recursive aggregation by an LSTM comparatively less influential than well pretrained node embeddings? Recent work [WYLZ19] suggests that this type of aggregation may put too much emphasis on the root node and nodes close to the root. It is shown that introducing dynamic routing can boost performance especially for deeper trees. A recent ablation study [LLFZ18] that studies the LSTM in Recurrent Neural Networks for NLP concludes that, among other results, the weighted sum of context words is a powerful simple alternative to LSTMs and that in fact LSTMS implicitly compute vectors of weighted sums.

Variation of network layout

We defined how our LSTM unit works[12] in Section 5.2.4. The equations 5.1-5.5 do not describe the output gate that is depicted in Figure 5.3. The following equation defines the output gate:

$$o = \sigma\left(\sum_{p=t,c} W_p^{(o)} x_p + \sum_{p=l,r} U_p^{(o)} h_p + b^{(o)}\right) \tag{5.7}$$

In addition, Equation 5.5 has to be changed to the following to integrate the gate into the LSTM:

$$h = o \odot \tanh(c) \tag{5.5*}$$

In Table 5.4 we show that introducing such an output gate does not improve the performance. The additional parameters of the output gate's weight matrices and bias introduce about 25% more parameters. And the added complexity in the interaction of the gates may also add to the network's capability of approximating arbitrary functions. Both effects may lead the network to overfit to the training data, but this is a hypothesis that would require further experiments to test.

5.3.6 Influence of training aspects

Above evaluation concerned the makeup of the Neural Network. In the following, we attend to aspects that concern the training of the network.

Variation of loss function As explained in Section 5.3.2, we employ *error scaling*, a variant of cost-sensitive learning, to address the imbalance between the numbers of clones and non-clones. That means that the error observed when evaluating a non-clone pair with the Twin Neural Network is discounted by a factor that is equal to the ratio of clones to non-clones. This puts the entirety of clones on the same level of influence as the non-clones despite the wild mismatch.

We evaluate the influence of an additional factor of ten to this error. This explores the middle ground between no scaling (ratio roughly 1:20) and full scaling (1:1). Table 5.4 shows that this does not yield the same performance as full scaling. We observe, however, that convergence is faster and more stable under this condition. This suggests that the

[12]Note that there are slightly different competing definitions of LSTMs.

best use of error scaling might be to start with a higher factor and lower it until finally reaching equal error ratio.

A similar conclusion can be made for the margin parameter m (see equation 5.6). This parameter is a common parameter in contrastive loss. It means that non-clones that are already less similar than m will not count towards the error. The ideal setting is actually $m = 0$. In Table 5.4 we show the performance for two other values for m: 0.5 and 0.9. Both yield worse performance than $m = 0$, but the higher value less so than the intermediate value. In both cases, the convergence is faster and for $m = 0.5$, the learning curves are more stable. Again, it might be advantageous to start with a higher value for m to make use of the faster convergence. In this phase, trees that have assumed representations such that many negative pairs are relatively far away from each other already, would not receive penalty. Only when m has been decreased later, would the continued learning focus on these easier cases.

Importance of cluster-aware data splits It is not uncommon in supervised Code Clone Detection to simply split a set of known clones and non-clones into training and test sets regardless of which clusters the involved code fragments belong to. As described in Section 2.3.2, this bears the danger of yielding unrealistic test performance. Because the model has already seen instances from the same cluster in the training that it is exposed to in testing, the measured performance cannot reflect generalization. It is in fact likely that the model is evaluated on some of the exact same fragments, because only the code *pairs* are guaranteed to not reappear in testing.

As described in Section 5.3.1, we make sure to split the data sets based on single code fragments instead of pairs and we draw careful lines along clusters. This way, we can even speak of training and testing *clusters*. To make the point about the generalization, we conduct an experiment where we split the data in a cluster-unaware fashion. We simply split the set of code fragment *pairs* into training and testing sets. We prepare three such splits that exactly mirror the number of clones and non-clones in the corresponding fold of our regular evaluation. That is, the first split contains 3,277 clones and 64,619 non-clones, and so on. This eliminates the question as to whether these numbers and ratios play any role.

Table 5.4 shows the performance on this pair-based split that does not respect cluster boundaries or even reuse of individual code fragments (“cluster-unaware split”). Not unexpectedly, the performance is vastly better than in the careful evaluation. In fact, it is comparable to the performance on the training set. It should be mentioned that [ZH18] reports a similar comparison and claims to not observe such difference at all.

Qualitative assessment of node embeddings In Section 5.3.5 we demonstrated quantitatively that pretrained embeddings are a very important part of our model for code similarity. Even just averaging the node embedding vectors beats the aggregation by a trained LSTM that had had to adjust random initial embedding matrices by the supervision signal propagating from the root to the other nodes.

Now we evaluate the embedding vectors that represent the AST nodes qualitatively. In Figures 5.6 and 5.7 a projection of the 4-dimensional type vectors and 10-dimensional content vectors respectively, is shown. We use the T-SNE algorithm [MH08] to calculate this projection (perplexity of 15). T-SNE approximately preserves local neighborhood relations. It thus cannot give us information about what specific directions in higher-dimensional space may encode (if there is a discernible meaning at all). But it can give us a glimpse into what tokens are represented by similar vectors to other tokens. We identify and outline a few clusters and offer some interpretation as to why the tokens they are comprised of end up having similar vectors.

Most of the node types in cluster “A” only ever occur as leaf nodes (e.g., “NullLiteral”, “SimpleName”). They all share the common null context from the side of their (nonexistent) children. Near the top of the plot we find various block types (cluster “B”; e.g., “Block”, “TryStatement”). Node types that often have a number expression (e.g., literal or variable) as their argument, (and therefore, first child) are found in cluster “C” (e.g., “InfixExpression”, “Assignment”). Node types which often have a Java type as their first child are located at the bottom, in cluster “D” (e.g., “SingleVariableDeclaration”, “ParameterizedType”). Some of the artificial node types (suffix “EXT”) do end up near their parent types (e.g., “Block”, “TryStatement”, “IfStatement”, “SingleVariableDeclaration”). This is likely the case because these blocks come both in small sizes where the block has only up to two children, and arbitrarily big sizes, where the “-EXT” variants hold the children in lieu of the original parent, or other “-EXT” nodes themselves.

The node contents in Figure 5.7 are much more intuitively interpretable, because they hold tokens that are visible to (and definable by) the programmer[13].

There are obvious clusters of related tokens. In cluster “1”, we find exception types (bottom), logging related terms (middle) and exception variable names and logging levels (top). In cluster “2”, there are I/O stream classes whereas the I/O stream related instance identifiers are in cluster “3”. File names, suffixes and paths (e.g., “fileName”, “filename”, “src”) are in the top right cluster “4”. Note that here we find “zipFile” while “ZipFile” with a capital letter is in cluster “2” for I/O stream related classes.

[13]“EMPTY” and “SPACE” represent the otherwise invisible empty string and whitespace, and “NO_CONTENT” is the placeholder for nodes without content.

In cluster “5”, we find tokens from string related operations such as “startsWith”, “endsWith” or “substring”. In cluster “ 6” are operators, and primitive and basic types. Indices and positions (e.g., “pos”, “k”, “j”, “size” and small integer number literals) concerned with 1-dimensional data (arrays) are in cluster “7” in the bottom right. Above that, in cluster “8”, we similarly find indices and other terms around 2-dimensional data (tables; e.g., “column”, “rows”, “fields”). These last two clusters illustrate a finding by Wainakh et al. [WRP19]. Tokens seem to tend to group along relatedness (“columns” and “rows”) more than individual similarity (“pos” and “position”). In fact, they showed that generic edit distances better models similarity than the (word-based) word2vec, but not relatedness.

The fact that the pretrained embedding vectors for very similar variations or synonyms are placed nearby each other is very useful when we want to identify code clones downstream. This can help to account for the variation that distinguishes the codes in a Type-2 clone. It also allows to identify non-essential additions, like the catching of an exception, which may elevate a clone to Type-3.

It is noteworthy that conversely, some contents are very similar despite having quite different meaning. We explain in Section 5.1.4 that this can happen with word2vec-like pretraining. This affects the modifiers “public”, “private” and “protected” that happen to share a lot of their contexts. Quite odd pairs like “||” (lazy *or*) and “&&” (lazy *and*) or even polar opposites like “true” and “false” end up direct neighbours in embedding space[14]. This may seem like an undesirable artifact. However, this still does encode the information that the node holds some Boolean value. And the supervised training does backpropagate the error signal all the way to the node content embeddings, as well. So, if it is crucial to distinguish these values in order to separate non-clones in the training data, the finetuning of the embeddings can conceivably change the vectors accordingly.

[14]This effect has been noted for identifier names like “row” vs “col” in [WRP19].

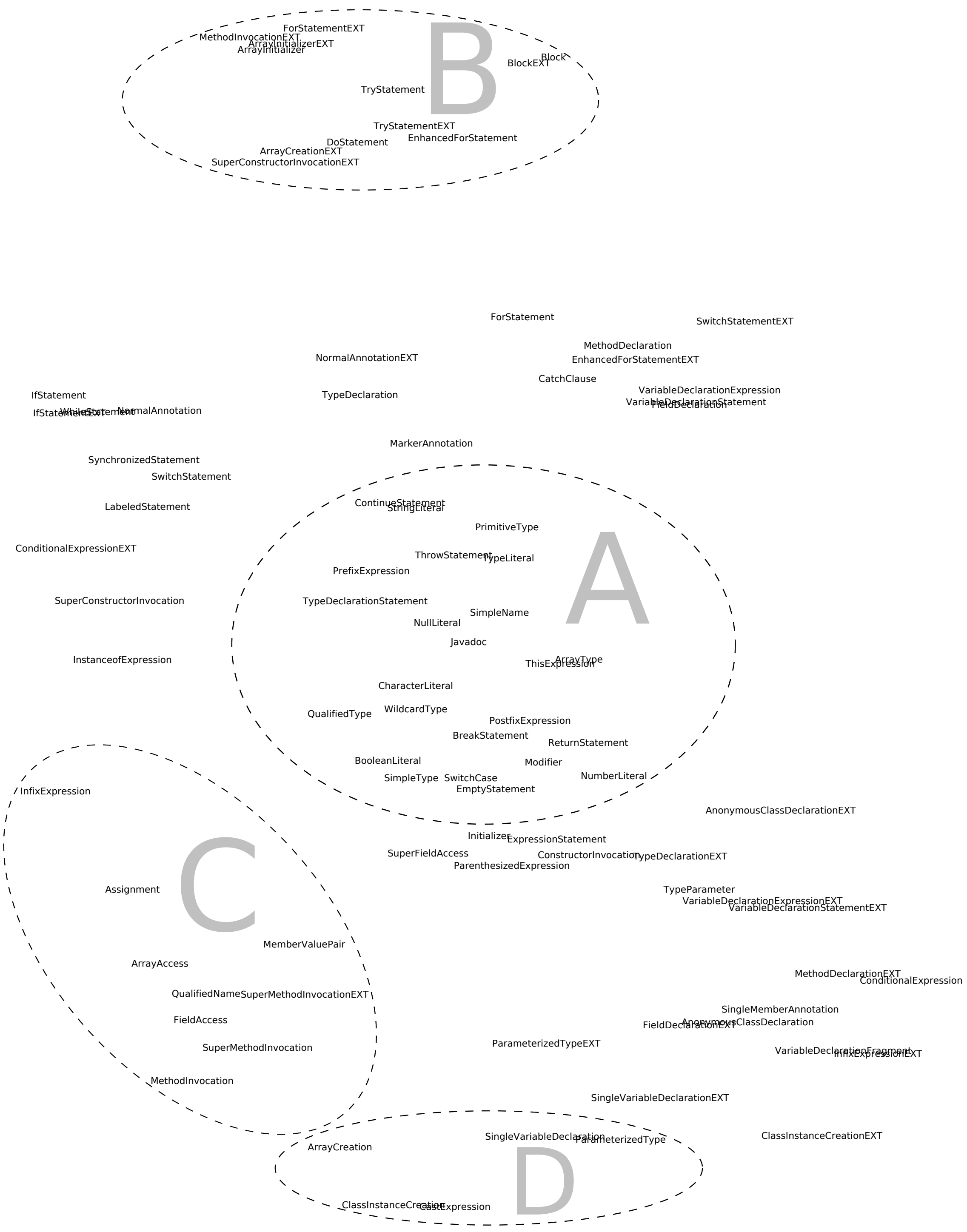


Figure 5.6: T-SNE projection of node type embedding vectors

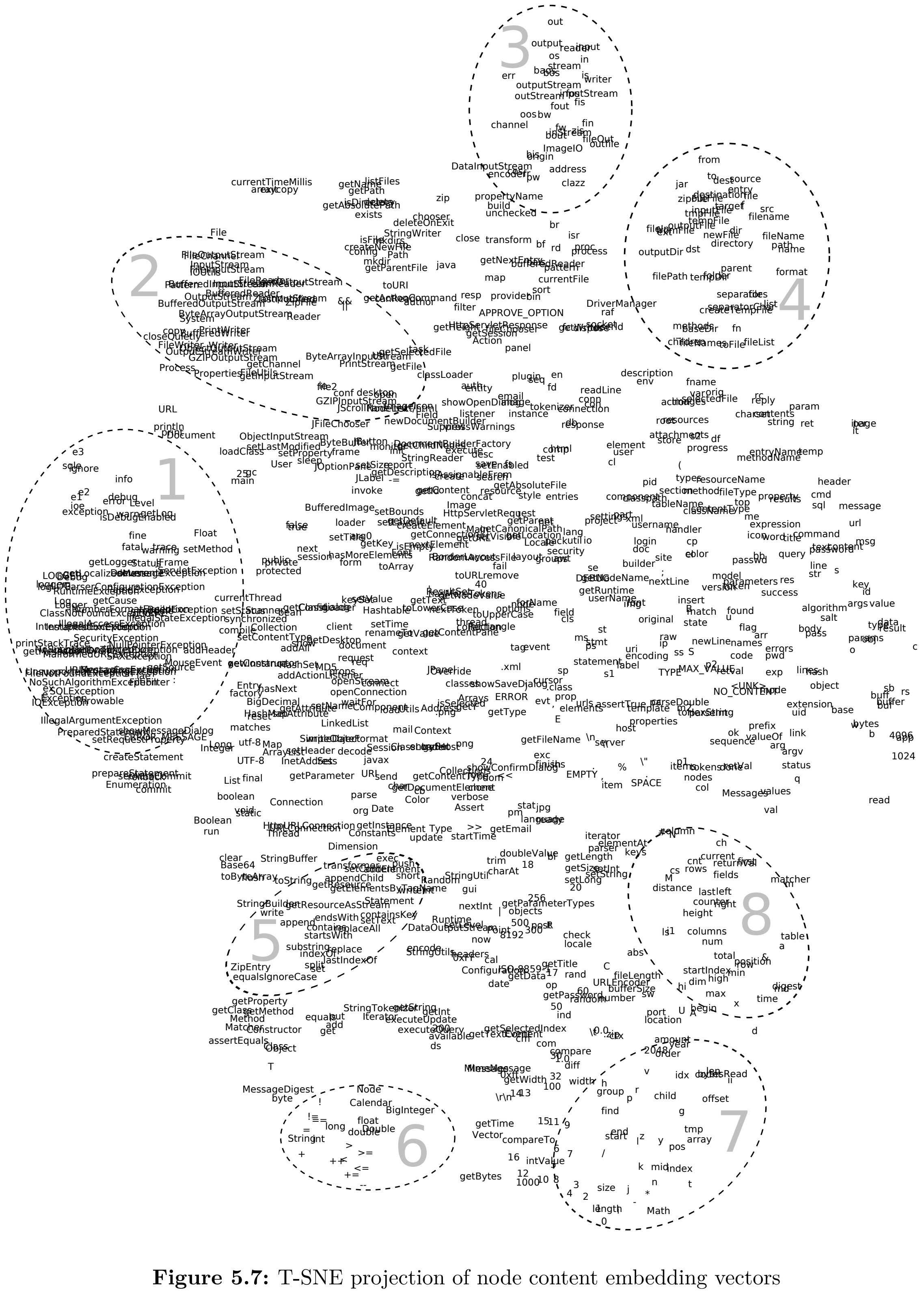


Figure 5.7: T-SNE projection of node content embedding vectors

5.4 Discussion

In this chapter we presented our approach to learn a vector representation of code fragments that allows for Code Clone Detection by comparing these vectors.

We start by learning an embedding for the discrete symbols defining the nodes of the AST trees. These are recursively aggregated by an LSTM unit. Both the embedding vectors and the parameters in the LSTM are subject to supervised optimization of the Code Clone Detection task. In order to measure the real power of the aggregation, we remove the method name node from the representation. We also clear the training and test set from duplicates in terms of isomorphic ASTs.

We found that the model benefits from a higher dimensionality up to a certain point after which performance deteriorates. This is likely due to overfitting the training data. A good and simple baseline for CCD is to simply average node vectors which especially does not involve further learning. We show that error scaling is a good way to address the class imbalance problem. Our results also imply that in supervised learning of Code Clone Detection, it is important to split training and test data by clusters to get a useful estimation of generalization to unseen clusters.

There are some interesting open questions around our findings. Most revolve around the representation of the vocabulary. One approach to represent an open vocabulary without masking the long tail with an *<UNK>* token is to encode it with a character-RNN. A Recurrent Neural Network has been used in purely unsupervised CCD in [WTVP16] but since it was not based on characters, it cannot extend to an open vocabulary. Also, supervised CCD can further finetune such embedding. Another alternative is given by subwords as basic units [KBR$^+$20] which allows different embedding algorithms from token prediction by uni-directional RNN, supports an open vocabulary, and has been shown to better model identifier similarity [WRP19]. Finally, the "tough-to-beat" baseline for sentence representation [ALM16] looks like a good candidate for a CCD baseline. Other ideas from adjacent fields include the use of histogram loss [UL16] to learn to distinguish clones from non-clones.

Generally, it would be useful to have a standard benchmark for supervised Code Clone Detection. This would ideally be part of a greater effort of making models comparable on a diverse set of tasks as it would enable studying task-independent modelling of source code (see next section). Since CCD is a task that is very easy to evaluate on any vector representation, it lends itself perfectly to study the interaction with other tasks. One could study the question if a representation that results from training on CCD gives a performance boost for training on another task. Of course, this cross-task transfer learning consideration works in the opposite direction too. As in, does the CCD performance improve by pretraining on a different task?

Finally, it would be good to get a qualitative insight into how our CCD model or others work. For the node content embeddings, we were able to show that these capture certain semantic information well (Section 5.3.6). But what about the dynamic non-linear way the LSTM aggregates the information? Is it possible to identify certain parts of the aggregation scheme as explicit and transparent features? Approaches like *Layer-wise Relevance Propagation* (LRP [BBM$^+$15]) offer an insight into how important any part of a given input is to the decision a Neural Network made. LRP has been extended to more involved models like LSTM [AMMS17].

5.5 Outlook

During the work on this chapter, it became apparent that Software Engineering in general and Code Clone Detection in particular share a lot of challenges with the field of Natural Language Processing. They both deal with human-generated data of varying size and quality with an open vocabulary and hard to grasp semantics. To elaborate on this idea, we give answers to two questions posed in the 2015 NSF Interdisciplinary Workshop on Statistical NLP and Software Engineering[15].

What's special about software?

Programming languages are obviously very different from Natural languages like English. Allamanis et al. [ABDS18] point out that they are designed and always have the goal of execution. In some ways, the formal nature makes things easier. For one, the syntax and grammar is completely specified whereas these are tasks in their own right in NLP.

Many problems with representing programming languages have to do with vocabulary. In natural languages, large parts of the vocabulary of have fixed meanings. Their might be several senses of the same lexical unit (e.g., “power”, as in physics vs “power”, as in roles), but these can usually be disambiguated from local context. In programming languages, user-defined identifiers are common-place since constructs like variables, methods and such need names. Code fragments can be left completely equivalent by refactoring operations like renaming of variables which does not have an equivalent in natural language. There might be some loose morphology that is prescribed by some written or unwritten programming style guides but in principle names are free to choose.

[15]`http://languageandcode.org/nlse2015/#workshop_program`

This open vocabulary problem is not unique to programming languages but is arguably more severe [HD17]. It can be ameliorated by subword embeddings [KBR$^+$20].

Resolving the connection between named entities and the pronouns and other anaphora that refer to them is a difficult task in NLP. Natural language has a limited set of pronouns that have to be reassigned to refer to another entity. In code, this is easier because of the designed nature of programming languages. But scope, number and interaction of variables make grasping the semantics very difficult [HD17]. We can address this problem with static and dynamic analysis. Allamanis et al. stipulate that there will be a wave of machine learning approaches for code understanding that will leverage such semantics [ABDS18].

What are NLP modelling techniques that are ripe for carrying over?

Despite the huge differences, there are also many similar aspects that might make it possible to exploit proven concepts from NLP. The **naturalness hypothesis** states that since software code is a kind of human communication, programming language corpora have similar statistical properties to natural language corpora [ABDS18].

At least since a few years, more often than not the starting point in NLP has become some general-purpose pretrained language model. These are language models that are trained on raw data with self-supervised training objectives like masked language modelling [DCLT19, KMBS19] or replaced token detection [CLLM20, FGT$^+$20][16]. They have been hugely successful, define the state-of-the-art in NLP and have long been commodified in a comprehensive, easy to access library including code and pretrained (natural language) models [WDS$^+$19].

Code Clone Detection as a task is first in line to be addressed by a general-purpose language model as it can be approached by merely comparing representations of code fragments. Works like [WTVP16, ZWZ$^+$19, GWL$^+$19] use self-supervised training objectives to solve CCD. It seems very promising to turn the modern arsenal of language modelling techniques developed in NLP towards this task.

These general-purpose representations from pretrained language models could show their potential beyond CCD best in clearly defined shared tasks. Apart from CCD (BigCloneEval [SR16]), there exist shared tasks around code retrieval using natural language (GitHub's CodeSearchNet[17] [HWG$^+$19]) and code generation from natural

[16]This extends the idea of traditional language modelling with unidirectional Recurrent Neural Networks (next token prediction, [KJFF16, KZTC20]).

[17]`https://app.wandb.ai/github/codesearchnet/benchmark/leaderboard`

language (CoNaLa[18] [YDC+18]) generating program snippets from natural language. Other tasks could include natural language generation from code [Neu15], code summarization [FAB19], code classification tasks, grammaticality/syntax error detection, and bug localization. It would also be apt to include a general language modelling (token prediction) task.

The field of NLP has shown that such tasks can be evaluated with shallow models on top of general-purpose models. This allows to create evaluations summarized in leaderboards like GLUE[19] [WSM+18] and SuperGLUE[20] [WPN+19]. There are even models that natively cast tasks in one master task like conditioned language generation (CTRL [KMV+19]) or translation (T5 [RSR+19]). An initiative that could bundle these tasks is “Learning from ‘Big Code”’[21] proposed at Dagstuhl Seminar “Programming with ‘Big Code”’.

[18] `https://conala-corpus.github.io/`
[19] `https://gluebenchmark.com`
[20] `https://super.gluebenchmark.com/`
[21] `http://learnbigcode.github.io/challenges/`

6 Conclusion

This thesis studies matching problems that arise in the natural evolution and connection of data sources. One one hand, we approach the very generic String Matching problem by selecting from proven models with very low labelling effort. On the other, we tackle a matching problem involving very specific data (code deduplication). It cannot be solved by simple existing models and we therefore study learning a representation which allows for easy matching. We identify and study common challenges that arise in any supervised approach to matching tasks because it is concerned with learning a binary relation which comes with certain properties.

The Record Matching problem is very old (considering the pace of Computer Science) and ubiquitous. And matching of strings is an even more ubiquitous problem. We contribute to the existing canon of work in two ways. Firstly, we study string similarity measures themselves. In this, we offer a new, psychological view on string similarity measures which emphasizes the role of the human who ultimately defines and judges similarity. And we discuss for the first time the naming conflict with respect to the so-called 'Monge-Elkan similarity', which we discovered is used to refer to two very different string similarity measures.

Secondly, we study an Active Learning approach to String Matching which outputs a string similarity measure and a decision threshold. Our approach requires very low labelling effort – an order of magnitude less than existing work. The existing work is mostly on *Record* Matching, but some of the studied matching data sets are identical. This efficiency is partly due to our taking the cold start problem very seriously. We employ secondary ranking when hypotheses are still largely indiscernible and a self-regulating mechanism to counteract initial biases. There is likely potential for further improvement by logical inference. The more generic problem of Active Learning of Record Matching is still active and has recently seen an effort in standardization in the form of a benchmark framework.

The second matching task we consider is Code Clone Detection (CCD). This is a problem with much less history and it cannot benefit in the same way from the big set of simple, tested and proven similarity measures. Our approach is part of the recent trend towards learning representations as opposed to learning a model on top of bespoke features. For supervised CCD, there are no benchmarks that provide a standard split into seen and unseen data. We point out that in the context of learning binary features this split is important to get right in order to measure generalization. We found that token representation plays a very important role. It is an open question if supporting an open vocabulary can benefit supervised CCD.

We also show that there is a trend to use unsupervised pretrained models (and not just word embeddings) now standard in NLP. The impact that such models will have on Code Clone Detection is still to be seen. Also, CCD is the most basic task that lends itself best to using such representations because it can directly be used for comparison. Multitask and transfer learning which are very widespread in computer vision as well as in NLP are exciting prospects for the field of Software Engineering which are facilitated by these models.

Bibliography

[ABDS18] Miltiadis Allamanis, Earl T. Barr, Premkumar Devanbu, and Charles Sutton. A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR)*, 51(4):1–37, 2018. DOI: 10.1145/3212695. 43, 101, 102

[ACG02] Rohit Ananthakrishna, Surajit Chaudhuri, and Venkatesh Ganti. Eliminating fuzzy duplicates in data warehouses. In *Proceedings of the 28th International Conference on Very Large Databases*, VLDB 2002, pages 586–597. Elsevier, 2002. DOI: 10.1016/B978-155860869-6/50058-5. 36

[ACGK08] Arvind Arasu, Surajit Chaudhuri, Kris Ganjam, and Raghav Kaushik. Incorporating string transformations in record matching. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD 2008, pages 1231–1234. ACM, 2008. DOI: 10.1145/1376616.1376742. 36

[AGK10] Arvind Arasu, Michaela Götz, and Raghav Kaushik. On active learning of record matching packages. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD 2010, pages 783–794, 2010. DOI: 10.1145/1807167.1807252. 39

[AJW$^+$14] Ning An, Lili Jiang, Jianyong Wang, Ping Luo, Min Wang, and Bing Nan Li. Toward detection of aliases without string similarity. *Information Sciences*, 261:89–100, 2014. DOI: 10.1016/j.ins.2013.11.010. 40

[AKR09] Avi Arampatzis, Jaap Kamps, and Stephen Robertson. Where to stop reading a ranked list?: threshold optimization using truncated score distributions. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR 2009, pages 524–531. ACM, 2009. DOI: 10.1145/1571941.1572031. 36

[ALM16] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. In *International Conference on Learning Representations*, ICLR 2017, 2016. URL: `https://openreview.net/forum?id=SyK00v5xx`. 42, 100

[AMMS17] Leila Arras, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. Explaining recurrent neural network predictions in sentiment analysis. In *Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 159–168, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. DOI: 10.18653/v1/W17-5221. 101

[AP11] Josh Attenberg and Foster Provost. Inactive learning? difficulties employing active learning in practice. *ACM SIGKDD Explorations Newsletter*, 12(2):36–41, 2011. DOI: 10.1145/1964897.1964906. 22, 54

[ATGW15] Miltos Allamanis, Daniel Tarlow, Andrew Gordon, and Yi Wei. Bimodal modelling of source code and natural language. In *Proceedings of the 32nd International Conference on Machine Learning*, ICML 2015, pages 2123–2132, 2015. 43

[AvH01] Avi Arampatzis and André van Hameran. The score-distributional threshold optimization for adaptive binary classification tasks. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR 2001, pages 285–293. ACM, 2001. DOI: 10.1145/383952.384009. 36, 37, 49

[AX06] Prasanth Anbalagan and Tao Xie. Efficient mutant generation for mutation testing of pointcuts in aspect-oriented programs. In *Second Workshop on Mutation Analysis*, ISSRE 2006, pages 3–3. IEEE, 2006. DOI: 10.1109/MUTATION.2006.3. 19

[BA15] Lutz Büch and Artur Andrzejak. Approximate string matching by end-users using active learning. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM 2015, pages 93–102. ACM, 2015. DOI: 10.1145/2806416.2806453. 2, 19, 45

[BA19] Lutz Büch and Artur Andrzejak. Learning-based recursive aggregation of abstract syntax trees for code clone detection. In *Proceedings of the 26th IEEE International Conference on Software Analysis, Evolution and Reengineering*, SANER 2019, pages 95–104. IEEE, 2019. DOI: 10.1109/SANER.2019.8668039. 2, 40, 71, 74, 83

[Bak92] Brenda S Baker. A program for identifying duplicated code. *Computing Science and Statistics*, pages 49–57, 1992. 73

[BBM$^+$15] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7), 2015. DOI: 10.1371/journal.pone.0130140. 101

[BBS08] Maria-Florina Balcan, Avrim Blum, and Nathan Srebro. A theory of learning with similarity functions. *Machine Learning*, 72(1-2):89–112, 2008. DOI: 10.1007/s10994-008-5059-5. 36

[BD19] Sebastian Baltes and Stephan Diehl. Usage and attribution of stack overflow code snippets in github projects. *Empirical Software Engineering*, 24(3):1259–1295, 2019. DOI: 10.1007/s10664-018-9650-5. 71

[BDBU$^+$18] Mathilde Borg Dahl, Asker D Brejnrod, Martin Unterseher, Thomas Hoppe, Yun Feng, Yuri Novozhilov, Søren J Sørensen, and Martin Schnittler. Genetic barcoding of dark-spored myxomycetes (amoebozoa)—identification, evaluation and application of a sequence similarity threshold for species differentiation in ngs studies. *Molecular Ecology Resources*, 18(2):306–318, 2018. DOI: 10.1111/1755-0998.12725. 37

[BDD15] Charles Bettembourg, Christian Diot, and Olivier Dameron. Optimal threshold determination for interpreting semantic similarity and particularity: application to the comparison of gene sets and metabolic pathways using go and chebi. *PloS one*, 10(7):e0133579, 2015. DOI: 10.1371/journal.pone.0133579. 37

[BDW08] Sugato Basu, Ian Davidson, and Kiri Wagstaff, editors. *Constrained clustering: Advances in algorithms, theory, and applications.* Data Mining and Knowledge Discovery Series. CRC Press, 2008. 22

[BE08] Michele Banko and Oren Etzioni. The tradeoffs between open and traditional relation extraction. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, ACL 2008, pages 28–36. Association for Computational Linguistics, 2008. 18

[BFSO84] Leo Breiman, Jerome H. Friedman, Charles J. Stone, and Richard A. Olshen. *Classification and regression trees.* CRC press, 1984. DOI: 10.1201/9781315139470. 24

[BGL$^+$93] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a "siamese" time delay neural network. In *Advances in Neural Information Processing Systems 6*, NIPS 1993, pages 737–744. Curran Associates, Inc., 1993. 79, 87

[BHS13] Aurélien Bellet, Amaury Habrard, and Marc Sebban. A survey on metric learning for feature vectors and structured data. *preprint*, 2013, 1306.6709. 21, 36, 40

[BKD17] Sebastian Baltes, Richard Kiefer, and Stephan Diehl. Attribution required: Stack Overflow code snippets in GitHub projects. In *IEEE/ACM 39th International Conference on Software Engineering Companion*, ICSE 2017, pages 161–163. IEEE, 2017. DOI: 10.1109/ICSE-C.2017.99. 71

[BM03a] Mikhail Bilenko and Raymond J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 2003, pages 39–48. ACM, 2003. DOI: 10.1145/956750.956759. 19, 36

[BM03b] Mikhail Bilenko and Raymond J. Mooney. On evaluation and training-set construction for duplicate detection. In *Proceedings of the KDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pages 7–12, 2003. 40

[BMC$^+$03] Mikhail Bilenko, Raymond J. Mooney, William Cohen, Pradeep Ravikumar, and Stephen Fienberg. Adaptive name matching in information integration. *IEEE Intelligent Systems*, 18(5):16–23, 2003. DOI: 10.1109/MIS.2003.1234765. 18

[BN09] Jens Bleiholder and Felix Naumann. Data fusion. *ACM computing surveys (CSUR)*, 41(1):1–41, 2009. DOI: 10.1145/1456650.1456651. 45

[BQL$^+$18] Nikita Bhutani, Kun Qian, Yunyao Li, H.V. Jagadish, Mauricio Hernandez, and Mitesh Vasa. Exploiting structure in representation of named entities using active learning. In *Proceedings of the 27th International Conference on Computational Linguistics*, COLING 2018, pages 687–699, 2018. URL: `https://www.aclweb.org/anthology/C18-1058`. 40

[Bra97] Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997. DOI: 10.1016/S0031-3203(96)00142-2. 31

[BTR16] Paula Branco, Luís Torgo, and Rita P. Ribeiro. A survey of predictive modeling under imbalanced distributions. *ACM Computing Surveys (CSUR)*, 49(2):1–31, 2016. DOI: 10.1145/2907070. 22, 23, 27, 34, 36

[BYM$^+$98] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Anna, and Lorraine Bier. Clone detection using abstract syntax trees. In *Proceedings of the International Conference on Software Maintenance*, ICSM 1998, pages 368–377. IEEE, 1998. DOI: 10.1109/ICSM.1998.738528. 73

[BZG13] Daniel Bär, Torsten Zesch, and Iryna Gurevych. DKPro similarity: An open source framework for text similarity. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, ACL 2013, pages 121–126. Association for Computational Linguistics, 2013. URL: `https://dkpro.github.io/dkpro-similarity/`. 19

[CBHK02] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002. DOI: 10.1613/jair.953. 23, 24

[CFT93] S. Carter, R.J. Frank, and D.S.W. Tansley. Clone detection in telecommunications software systems: A neural net approach. In *Proceedings of the International Workshop on Application of Neural Networks to Telecommunications*, pages 273–287, 1993. 72

[CH13] Michelle Cheatham and Pascal Hitzler. String similarity metrics for ontology alignment. In *Proceedings of the 12th International Semantic Web Conference*, ISWC 2013, pages 294–309, Berlin, Heidelberg, 2013. Springer, Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-41338-4_19. 18, 35

[Che17] Ekaterina Chernyak. Comparison of string similarity measures for obscenity filtering. In *Proceedings of the 6th Workshop on Balto-Slavic Natural Language Processing*, pages 97–101. Association for Computational Linguistics, 2017. 35

[Chr06] Peter Christen. A comparison of personal name matching: Techniques and practical issues. In *Sixth IEEE International Conference on Data Mining-Workshops*, ICDMW 2006, pages 290–294. IEEE, 2006. DOI: 10.1109/ICDMW.2006.2. 19, 35

[Chr12] Peter Christen. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer Science & Business Media, 2012. DOI: 10.1007/978-3-642-31164-2. 11, 18, 19, 20, 23, 25

[CLK$^+$19] Jose Cambronero, Hongyu Li, Seohyun Kim, Koushik Sen, and Satish Chandra. When deep learning met code search. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE, pages 964–974, 2019. DOI: 10.1145/3338906.3340458. 43

[CLLM20] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*, ICLR 2020, 2020, 2003.10555. URL: `https://openreview.net/forum?id=r1xMH1BtvB`. 102

[CM19] Zimin Chen and Martin Monperrus. A literature study of embeddings on source code. *preprint*, 2019, 1904.03061. 43, 77

[CNC05] Sam Chapman, Barry Norton, and Fabio Ciravegna. Armadillo: Integrating knowledge for the semantic web. In *Proceedings of the Dagstuhl Seminar in Machine Learning for the Semantic Web*, 2005. 18

[Coh00] William W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems*, 18(3):288–321, July 2000. DOI: 10.1145/352595.352598. 56

[CRF03] William W. Cohen, Pradeep D. Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for namematching tasks. In *Proceedings of the 2003 International Conference on Information Integration on the Web*, IIWEB 2003, pages 73—78. AAAI Press, 2003. URL: `https://www.cs.cmu.edu/~wcohen/postscript/ijcai-ws-2003.pdf`. 11, 17, 18, 19, 20, 35, 39, 56, 57, 58, 59

[CRFR03] William W. Cohen, Pradeep Ravikumar, Stephen Fienberg, and Kathryn Rivard. Secondstring: An open source java toolkit of approximate string-matching techniques, 2003. URL: `http://secondstring.sourceforge.net`. 18, 19, 20, 39, 56, 59

[CS06] Horacio Camacho and Abdellah Salhi. A string metric based on a one-to-one greedy matching algorithm. *Research in Computer Science number*, 19:171–182, 2006. 13

[CSC04] Sam Chapman, Marco Aurélio Graciotto Silva, and Horacio Camacho. Simmetrics: A similarity metric library for strings, 2004. 18, 19

[CVW15] Peter Christen, Dinusha Vatsalan, and Qing Wang. Efficient entity resolution with adaptive and interactive training data selection. In *Proceedings of the IEEE International Conference on Data Mining*, ICDM 2015, pages 727–732. IEEE, 2015. DOI: 10.1109/ICDM.2015.63. 39, 62

[CYZ19] Long Chen, Wei Ye, and Shikun Zhang. Capturing source code semantics via tree-based convolution over API-enhanced AST. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, CF 2019, pages 174–182, 2019. DOI: 10.1145/3310273.3321560. 42

[DAC10] Danica Damljanovic, Milan Agatonovic, and Hamish Cunningham. Natural language interfaces to ontologies: Combining syntactic analysis and ontology-based lookup through the user interaction. In *Proceedings of the 7th Extended Semantic Web Conference*, ESWC 2010, pages 106–120, Berlin, Heidelberg, 2010. Springer, Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-13486-9_8. 19

[Dam64] Fred J. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964. DOI: 10.1145/363958.363994. 14

[Daw07] Richard Dawkins. Fresh Air: Richard Dawkins Explains 'The God Delusion', March 2007. URL: `https://freshairarchive.org/segments/richard-dawkins-explains-god-delusion`. 10

[DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American*

Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2019, 2019. DOI: 10.18653/v1/N19-1423. 102

[DG06] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, ICML 2006, pages 233–240. ACM, 2006. DOI: 10.1145/1143844.1143874. 32, 34

[DHI12] AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of data integration*. Elsevier, 2012. DOI: 10.1016/C2011-0-06130-6. 18, 19

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011. 89

[DKJ$^+$07] Jason V Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S Dhillon. Information-theoretic metric learning. In *Proceedings of the 24th International Conference on Machine Learning*, ICML 2007, pages 209–216, 2007. DOI: 10.1145/1273496.1273523. 43

[DLST04] Andrea De Lucia, Giuseppe Scanniello, and Genoveffa Tortora. Identifying clones in dynamic web sites using similarity thresholds. In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, ICEIS 2004, pages 391–396, 2004. DOI: 10.5220/0002597303910396. 74

[Dou17] Chenxiao Dou. *Property of Density in Entity Resolution and its Usage for Blocking and Learning*. PhD thesis, University of New South Wales, Sydney, Australia, 2017. 39

[dPAEG15] Maria del Pilar Angeles and Adrian Espino-Gamez. Comparison of methods Hamming distance, Jaro, and Monge-Elkan. In *Proceedings of the Seventh International Conference on Advances in Databases, Knowledge, and Data Applications*, DBKDA 2015, page 73, 2015. 20

[DS17] Vijay Prakash Dwivedi and Manish Shrivastava. Beyond word2vec: Embedding words and phrases in same vector space. In *Proceedings of the 14th International Conference on Natural Language Processing*, ICON 2017, pages 205–211, 2017. URL: `https://www.aclweb.org/anthology/W17-7526/`. 78

[DSLW17] Chenxiao Dou, Daniel Sun, Guoqiang Li, and Raymond K Wong. Active learning with density-initialized decision tree for record matching. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*, SSDBM 2017, pages 1–12, 2017. DOI: 10.1145/3085504.3085518. 39

[DSSOH07] Roberto Da Silva, Raquel Stasiu, Viviane Moreira Orengo, and Carlos A Heuser. Measuring quality of similarity functions in approximate data matching. *Journal of Informetrics*, 1(1):35–46, 2007. DOI: 10.1016/j.joi.2006.09.001. 35, 36, 37

[EHBG07] Seyda Ertekin, Jian Huang, Leon Bottou, and Lee Giles. Learning on the border: active learning in imbalanced data classification. In *Proceedings of the sixteenth ACM Conference on Conference on Information and Knowledge management*, CIKM 2007, pages 127–136, 2007. DOI: 10.1145/1321440.1321461. 24

[EIV07] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on knowledge and data engineering*, 19(1):1–16, 2007. DOI: 10.1109/TKDE.2007.250581. 18

[EJ07] Michael Ellsworth and Adam Janin. Mutaphrase: Paraphrasing with framenet. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 143–150. Association for Computational Linguistics, 2007. URL: `https://www.aclweb.org/anthology/W07-1424`. 9

[EM20] Charles P. Elkan and Alvaro Edmundo Monge. private correspondence, January 2020. 58

[EW19] Mathias Etcheverry and Dina Wonsever. Unraveling antonym's word vectors through a Siamese-like network. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, ACL 2019, pages 3297–3307. Association for Computational Linguistics, 2019. DOI: 10.18653/v1/P19-1319. 41

[FAB19] Patrick Fernandes, Miltiadis Allamanis, and Marc Brockschmidt. Structured neural summarization. In *International Conference on Learning Rep-*

resentations, ICLR 2019, 2019, 1811.01824. URL: `https://openreview.net/forum?id=H1ersoRqtm`. 103

[Faw06] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006. DOI: 10.1016/j.patrec.2005.10.010. 30, 31, 32, 34

[FCW16] Jeffrey Fisher, Peter Christen, and Qing Wang. Active learning based entity resolution using markov logic. In *Proceedings of the 20th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, PAKDD 2016, pages 338–349, Berlin, Heidelberg, 2016. Springer. DOI: 10.1007/978-3-319-31750-2_27. 40

[FGT$^+$20] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. CodeBERT: A pre-trained model for programming and natural languages. *preprint*, 2020, 2002.08155. 43, 102

[Fir57] John R. Firth. A synopsis of linguistic theory, 1930-1955. *Studies in Linguistic Analysis*, 1952–59:1–32, 1957. 75

[FK00] Dayne Freitag and Nicholas Kushmerick. Boosted wrapper induction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 577–583. AAAI Press, 2000. 58

[FK15] Peter Flach and Meelis Kull. Precision-recall-gain curves: Pr analysis done right. In *Advances in Neural Information Processing Systems 28*, NIPS 2015, pages 838–846. Curran Associates, Inc., 2015. 29, 31, 32, 33, 34

[FS69] Ivan P. Fellegi and Alan B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969. DOI: 10.1080/01621459.1969.10501049. 36, 45

[FS92] Carol Friedman and Robert Sideli. Tolerating spelling errors during patient validation. *Computers and Biomedical Research*, 25(5):486–509, 1992. DOI: 10.1016/0010-4809(92)90005-U. 12

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth International Conference on Artificial Intelligence and Statistics*, AISTATS 2010, pages 249–256, 2010. 89

[GCCH17] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. Semantically enhanced software traceability using deep learning techniques. In *Proceedings of the IEEE/ACM 39th International Conference on Software Engineering*, ICSE 2017, pages 3–14. IEEE, 2017. DOI: 10.1109/ICSE.2017.9. 43

[GDD$^+$14] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F. Naughton, Narasimhan Rampalli, Jude Shavlik, and Xiaojin Zhu. Corleone: hands-off crowdsourcing for entity matching. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, SIGMOD 2014, pages 601–612, New York, NY, USA, 2014. ACM. DOI: 10.1145/2588555.2588576. 19, 39

[GJM13] Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. Hybrid speech recognition with deep bidirectional LSTM. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 273–278. IEEE, 2013. DOI: 10.1109/ASRU.2013.6707742. 79

[GK96] Christoph Goller and Andreas Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of the International Conference on Neural Networks*, volume 1 of *ICNN 1996*, pages 347–352. IEEE, 1996. DOI: 10.1109/ICNN.1996.548916. 82

[GK09] Nils Göde and Rainer Koschke. Incremental clone detection. In *Proceedings of the 13th European Conference on Software Maintenance and Reengineering*, CSMR 2009, pages 219–228. IEEE, 2009. DOI: 10.1109/CSMR.2009.20. 73

[GL14] Yoav Goldberg and Omer Levy. word2vec explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *preprint*, 2014, 1402.3722. 78

[GL16] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 2016, pages 855–864, 2016. DOI: 10.1145/2939672.2939754. 42

[GMIF16] Najlah Gali, Radu Mariescu-Istodor, and Pasi Fränti. Similarity measures for title matching. In *Proceedings of the 23rd International Conference* *on Pattern Recognition*, ICPR 2016, pages 1548–1553. IEEE, 2016. DOI: 10.1109/ICPR.2016.7899857. 35

[GRC11] Jim Gemmell, Benjamin I.P. Rubinstein, and Ashok K. Chandra. Improving entity resolution with global constraints. Technical report, Microsoft Research, 2011. 25, 61

[GWL$^+$19] Yi Gao, Zan Wang, Shuang Liu, Lin Yang, Wei Sang, and Yuanfang Cai. TECCD: A tree embedding approach for code clone detection. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution*, ICSME 2019, pages 145–156. IEEE, 2019. DOI: 10.1109/ICSME.2019.00025. 42, 102

[GWP$^+$17] Chuancong Gao, Jiannan Wang, Jian Pei, Rui Li, and Yi Chang. Preference-driven similarity join. In *Proceedings of the International Conference on Web Intelligence*, WI 2017, pages 97–105, 2017. DOI: 10.1145/3106426.3106484. 37

[Hah14] Ulrike Hahn. Similarity. *Wiley Interdisciplinary Reviews: Cognitive Science*, 5(3):271–280, 2014. DOI: 10.1002/wcs.1282. 5

[HBFS01] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. *A field guide to dynamical recurrent neural networks*, chapter Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. Wiley-IEEE Press, 2001. DOI: 10.1109/9780470544037.ch14. 79

[HCL06] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2 of *CVPR 2006*, pages 1735–1742. IEEE, 2006. DOI: 10.1109/CVPR.2006.100. 87

[HD17] Vincent J. Hellendoorn and Premkumar Devanbu. Are deep neural networks the best choice for modeling source code? In *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 763–773, 2017. DOI: 10.1145/3106237.3106290. 75, 102

[HGD19] Rishi Hazra, Shubham Gupta, and Ambedkar Dukkipati. Active^2 learning: Actively reducing redundancies in active learning methods for sequence tagging. *preprint*, 2019, 1911.00234. 39

[HM82] James A. Hanley and Barbara J. McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982. DOI: 10.1148/radiology.143.1.7063747. 34

[Ho95] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1 of *ICDAR 1995*, pages 278–282. IEEE, 1995. DOI: 10.1109/ICDAR.1995.598994. 39

[HWG$^+$19] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. CodeSearchNet challenge: Evaluating the state of semantic code search. *preprint*, 2019, 1909.09436. 102

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, ICCV 2015, pages 1026–1034, 2015. DOI: 10.1109/ICCV.2015.123. 89

[IJB12] Robert Isele, Anja Jentzsch, and Christian Bizer. Active learning of expressive linkage rules for the web of data. In *Proceedings of the 12th International Conference on Web Engineering*, ICWE 2012, pages 411–418, Berlin, Heidelberg, 2012. Springer. DOI: 978-3-642-31753-8_34. 39

[Jar89] Matthew A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989. DOI: 10.2307/2289924. 14

[JBGG09] Sergio Jimenez, Claudia Becerra, Alexander Gelbukh, and Fabio Gonzalez. Generalized Mongue-Elkan method for approximate text string comparison. In *Proceedings of the 10th International conference on intelligent text processing and computational linguistics*, CICLing 2009, pages 559–570. Springer, 2009. DOI: 10.1007/978-3-642-00382-0_45. 18, 19

[JDH10] Elmar Juergens, Florian Deißenböck, and Benjamin Hummel. Code similarities beyond copy & paste. In *Proceedings of the 14th European Conference on Software Maintenance and Reengineering*, CSMR 2010, pages 78–87. IEEE, 2010. DOI: 10.1109/CSMR.2010.33. 72

[JDHW09] Elmar Juergens, Florian Deißenböck, Benjamin Hummel, and Stefan Wagner. Do code clones matter? In *Proceedings of the IEEE 31st International*

Conference on Software Engineering, ICSE 2009, pages 485–495. IEEE, 2009. DOI: 10.1109/ICSE.2009.5070547. 72

[JMSG07] Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stephane Glondu. Deckard: Scalable and accurate tree-based detection of code clones. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE 2007, pages 96–105. IEEE, 2007. DOI: 10.1109/ICSE.2007.30. 73, 74, 90

[KB19] Mahmut Kaya and Hasan Şakir Bilge. Deep metric learning: a survey. *Symmetry*, 11(9):1066, 2019. DOI: 10.3390/sym11091066. 21, 23, 40

[KBdR16] Tom Kenter, Alexey Borisov, and Maarten de Rijke. Siamese CBOW: Optimizing word embeddings for sentence representations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, ACL 2016, pages 941–951, 2016. DOI: 10.18653/v1/P16-1089. 41, 78

[KBR$^+$20] Rafael-Michael Karampatsis, Hlib Babii, Romain Robbes, Charles Sutton, and Andrea Janes. Big code!= big vocabulary: Open-vocabulary models for source code. In *Proceedings of the 42nd International Conference on Software Engineering*, ICSE 2020, 2020. DOI: 10.1145/3377811.3380342. 43, 75, 77, 100, 102

[KDSG$^+$16] Pradap Konda, Sanjib K. Das, Paul Suganthan G.C., AnHai Doan, Adel Ardalan, Jeffrey R. Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff F. Naughton, Shishir K. Prasad, Ganesh Krishnan, Rohit Deep, and Vijay Roghavendra. Magellan: Toward building entity matching management systems. *Proceedings of the VLDB Endowment*, 9(12):1197–1208, 2016. DOI: 10.14778/2994509.2994535. 19

[KFF06] Rainer Koschke, Raimar Falke, and Pierre Frenzel. Clone detection using abstract syntax suffix trees. In *Proceedings of the 13th Working Conference on Reverse Engineering*, WCRE 2006, pages 253–262. IEEE, 2006. DOI: 10.1109/WCRE.2006.18. 73

[KJ12] Purushottam Kar and Prateek Jain. Supervised learning with similarity functions. In *Advances in Neural Information Processing Systems 25*, NIPS 2012, pages 215–223. Curran Associates, Inc., 2012. 36

[KJFF16] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. In *Workshop proceedings of the First International Conference on Learning Representations (ICLR)*, 2016, 1506.02078. URL: `https://openreview.net/forum?id=71BmK0m6qfAE8VvKUQWB`. 102

[KK98] Matjaz Kukar and Igor Kononenko. Cost-sensitive learning with neural networks. In *Proceedings of the 13th European Conference on Artificial Intelligence*, volume 98 of *ECAI 1998*, pages 445–449, 1998. 88

[KKI02] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. CCFinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, 28(7):654–670, 2002. DOI: 10.1109/TSE.2002.1019480. 73

[KKP06] Sotiris Kotsiantis, Dimitris Kanellopoulos, and Panayiotis Pintelas. Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering*, 30(1):25–36, 2006. 23, 36

[KMBS19] Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. Pretrained contextual embedding of source code. *preprint*, 2019, 2001.00059. URL: `https://openreview.net/forum?id=rygoURNYvS`. 43, 102

[KMV$^+$19] Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. CTRL: A conditional transformer language model for controllable generation. *preprint*, 2019, 1909.05858. 103

[KNLH19] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey E. Hinton. Similarity of neural network representations revisited. In *Proceedings of the 36th International Conference on Machine Learning*, ICML 2019, pages 3519–3529, 2019. 16

[Köp14] Hanna Köpcke. *Object Matching on real-world problems.* PhD thesis, Universität Leipzig, 2014. 35, 37

[KOPC14] Mincheol Kim, Hyun-Seok Oh, Sang-Cheol Park, and Jongsik Chun. Towards a taxonomic coherence between average nucleotide identity and 16S rRNA gene sequence similarity for species demarcation of prokaryotes. *International Journal of Systematic and Evolutionary Microbiology*, 64(2):346–351, 2014. DOI: 10.1099/ijs.0.059774-0. 37

[KPDA09] Evangelos Kanoulas, Virgil Pavlu, Keshi Dai, and Javed A. Aslam. Modeling the score distributions of relevant and non-relevant documents. In *Proceedings of the 2nd International Conference on Theory of Information Retrieval: Advances in Information Retrieval Theory*, ICTIR 2009, pages 152–163. Springer, 2009. DOI: 10.1007/978-3-642-04417-5_14. 36

[KQG$^+$19] Jungo Kasai, Kun Qian, Sairam Gurajada, Yunyao Li, and Lucian Popa. Low-resource deep entity resolution with transfer and active learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, ACL 2019, pages 5851–5861. Association for Computational Linguistics, 2019. DOI: 10.18653/v1/P19-1586. 40

[KR10] Hanna Köpcke and Erhard Rahm. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, 69(2):197–210, 2010. DOI: 10.1016/j.datak.2009.10.003. 18, 45

[Kri01] Jens Krinke. Identifying similar code with program dependence graphs. In *Proceedings of the Eighth Working Conference on Reverse Engineering*, WCRE 2001, pages 301–309. IEEE, 2001. DOI: 10.1109/WCRE.2001.957835. 74

[KTR09] Hanna Köpcke, Andreas Thor, and Erhard Rahm. Comparative evaluation of entity resolution approaches with FEVER. *Proceedings of the VLDB Endowment*, 2(2):1574–1577, 2009. DOI: 10.14778/1687553.1687595. 35

[KTR10] Hanna Köpcke, Andreas Thor, and Erhard Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment*, 3(1-2):484–493, 2010. DOI: 10.14778/1920841.1920904. 35

[Kul13] Brian Kulis. Metric learning: A survey. *Foundations and Trends® in Machine Learning*, 5(4):287–364, 2013. DOI: 10.1561/2200000019. 21, 40

[Kus97] Nicholas Kushmerick. *Wrapper induction for information extraction.* PhD thesis, University of Washington, 1997. 58

[KZTC20] Seohyun Kim, Jinman Zhao, Yuchi Tian, and Satish Chandra. Code prediction by feeding trees to transformers. *preprint*, 2020, 2003.13848. 43, 102

[LCHY06] Chao Liu, Chen Chen, Jiawei Han, and Philip S. Yu. GPLAG: detection of software plagiarism by program dependence graph analysis. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD 2006, pages 872–881, 2006. DOI: 10.1145/1150402.1150522. 74

[LCL$^+$04] Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul M.B. Vitányi. The similarity metric. *IEEE transactions on Information Theory*, 50(12):3250–3264, 2004. DOI: 10.1109/TIT.2004.838101. 15

[Lev66] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966. 14

[LFZ$^+$17] Liuqing Li, He Feng, Wenjie Zhuang, Na Meng, and Barbara Ryder. CCLearner: A deep learning-based clone detection approach. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution*, ICSME 2017, pages 249–260. IEEE, 2017. DOI: 10.1109/ICSME.2017.46. 26, 41, 83

[LLFZ18] Omer Levy, Kenton Lee, Nicholas FitzGerald, and Luke Zettlemoyer. Long Short-Term Memory as a dynamically computed element-wise weighted sum. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, ACL 2018, pages 732–739. Association for Computational Linguistics, 2018. DOI: 10.18653/v1/P18-2116. 93

[LLMZ06] Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. CP-Miner: Finding copy-paste and related bugs in large-scale software code. *IEEE Transactions on software Engineering*, 32(3):176–192, 2006. DOI: 10.1109/TSE.2006.28. 73

[LMM$^+$17] Cristina V. Lopes, Petr Maj, Pedro Martins, Vaibhav Saini, Di Yang, Jakub Zitny, Hitesh Sajnani, and Jan Vitek. DéjàVu: a map of code duplicates on GitHub. *Proceedings of the ACM on Programming Languages*, 1:1–28, 2017. DOI: 10.1145/3133908. 71

[LO04] Chin-Yew Lin and Franz Josef Och. Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, ACL 2004, page 605. Association for Computational Linguistics, 2004. DOI: 10.3115/1218955.1219032. 13

[LSJ04] Alon Lavie, Kenji Sagae, and Shyamsundar Jayaraman. The significance of recall in automatic metrics for mt evaluation. In *Proceedings of the 6th Conference of the Association for Machine Translation in the Americas*, AMTA 2004, pages 134–143, Berlin, Heidelberg, 2004. Springer. DOI: 10.1007/978-3-540-30194-3_16. 13

[LSLH18] Lingli Li, Xiaodan Shang, Jinbao Li, and Jin Hu. Learning distance metrics for entity resolution. *IEEE Access*, 6:54900–54909, 2018. DOI: 10.1109/ACCESS.2018.2871168. 41

[Lus71] Lee B Lusted. Signal detectability and medical decision-making. *Science*, 171(3977):1217–1219, 1971. URL: `https://www.jstor.org/stable/1731167`. 31

[Lus84] Lee B Lusted. Roc recollected. *Medical Decision Making*, 4(2):131–135, 1984. DOI: 10.1177/0272989X8400400201. 31

[Mat75] Brian W. Matthews. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975. DOI: 10.1016/0005-2795(75)90109-9. 30

[MC91] George A Miller and Walter G. Charles. Contextual correlates of semantic similarity. *Language and cognitive processes*, 6(1):1–28, 1991. DOI: 10.1080/01690969108406936. 5, 9, 10, 76

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Workshop proceedings of the First International Conference on Learning Representations (ICLR)*, 2013, 1301.3781. 76

[ME95] Alvaro Edmundo Monge and Charles P. Elkan. Integrating external information sources to guide worldwide web information retrieval. In *AAAJ 1995 Fall Symposium on Knowledge Navigation and Retrieval*, pages 1–12, 1995. 12, 16, 17, 18, 59

[ME96] Alvaro Edmundo Monge and Charles P. Elkan. The field matching problem: Algorithms and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD 1996, pages

267–270. AAAI Press, 1996. URL: `http://www.aaai.org/Papers/KDD/1996/KDD96-044.pdf`. 17, 18, 19, 58

[ME97] Alvaro Edmundo Monge and Charles P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proceedings of the SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 23–29, 1997. 17, 19

[MH08] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008. 96

[MK06] Matthew Michelson and Craig A Knoblock. Learning blocking schemes for record linkage. In *Proceedings of the 21st national conference on Artificial intelligence*, volume 6 of *AAAI 2006*, pages 440–445, 2006. 23

[MK17] Joel Ruben Antony Moniz and David Krueger. Nested LSTMs. In *Proceedings of the Ninth Asian Conference on Machine Learning*, ACML 2017, pages 530–544, 2017. 79

[MLJ$^+$16] Lili Mou, Ge Li, Zhi Jin, Lu Zhang, and Tao Wang. Convolutional neural networks over tree structures for programming language processing. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI 2016, pages 1287—1293, 2016. DOI: 10.5555/3015812.3016002. 42

[MLM96] Jean Mayrand, Claude Leblanc, and Ettore Merlo. Experiment on the automatic detection of function clones in a software system using metrics. In *Proceedings of the International Conference on Software Maintenance*, volume 96 of *ICSM 1996*, page 244, 1996. DOI: 10.1109/ICSM.1996.565012. 74

[MNU00] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 2000, pages 169—178, New York, NY, USA, 2000. DOI: 10.1145/347090.347123. 58

[Mon97] Alvaro Edmundo Monge. *Adaptive Detection of Approximately Duplicate Database Records and the Database Intergration Approach to Information Discovery*. PhD thesis, University of California, San Diego, 1997. 17

[Mon00] Alvaro Edmundo Monge. An adaptive and efficient algorithm for detecting approximately duplicate database records. *International Journal on Information Systems Special Issueon Data Extraction, Cleaning, and Reconciliation*, 2000. 25

[MPS20] George Mathew, Chris Parnin, and Kathryn T. Stolee. Slacc: Simion-based language agnostic code clones. In *Proceedings of the 42nd International Conference on Software Engineering*, ICSE 2020, 2020. DOI: 10.1145/3377811.3380407. 72

[MPSS20] Venkata Vamsikrishna Meduri, Lucian Popa, Prithviraj Sen, and Mohamed Sarwat. A comprehensive benchmark framework for active learning methods in entity matching. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD 2020, page 1133–1147. ACM, 2020. DOI: 10.1145/3318464.3380597. 40

[MRB18] Ari Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *Advances in Neural Information Processing Systems 31*, NeurIPS 2018, pages 5727–5736. Curran Associates, Inc., 2018. 16

[MSC$^+$13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, NIPS 2013, pages 3111–3119. Curran Associates, Inc., 2013. 76, 77

[MT16] Jonas Mueller and Aditya Thyagarajan. Siamese recurrent architectures for learning sentence similarity. In *Proceedings of the thirtieth Conference on Artificial Intelligence*, AAAI 2016, 2016. 40

[Mun21] Robert Munro. *Human-in-the-loop machine learning.* Manning Early Access Program. Manning, estim. 2021. URL: `https://www.manning.com/books/human-in-the-loop-machine-learning`. 47

[MWGM10] David Menestrina, Steven Euijong Whang, and Hector Garcia-Molina. Evaluating entity resolution results. *Proceedings of the VLDB Endowment*, 3(1-2):208–219, 2010. DOI: 10.14778/1920841.1920871. 34

[MYC08] Erwan Moreau, François Yvon, and Olivier Cappé. Robust similarity measures for named entities matching. In *Proceedings of the 22nd In-*

ternational Conference on Computational Linguistics, Coling 2008, pages 593–600, 2008. 18, 19, 35

[MYZ13] Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL 2013, pages 746–751. Association for Computational Linguistics, 2013. URL: `https://www.aclweb.org/anthology/N13-1090/`. 76

[Nav01] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys (CSUR)*, 33(1):31—88, March 2001. DOI: 10.1145/375360.375365. 11

[Neu15] Graham Neubig. Survey of methods to generate natural language from source code, 2015. URL: `http://www.languageandcode.org/nlse2015/neubig15nlse-survey.pdf`. 103

[NH10] Felix Naumann and Melanie Herschel. An introduction to duplicate detection. *Synthesis Lectures on Data Management*, 2(1):1–87, 2010. DOI: 10.2200/S00262ED1V01Y201003DTM003. 12, 18, 19

[NL12] Axel-Cyrille Ngonga Ngomo and Klaus Lyko. EAGLE: Efficient active learning of link specifications using genetic programming. In *Proceedings of the 9th Extended Semantic Web Conference*, ESWC 2012, pages 149–163, Berlin, Heidelberg, 2012. Springer. DOI: 10.1007/978-3-642-30284-8_17. 39

[NLAH11] Axel-Cyrille Ngonga Ngomo, Jens Lehmann, Sören Auer, and Konrad Höffner. Raven – active learning of link specifications. In *Proceedings of the 6th International Conference on Ontology Matching*, OM 2011, 2011. 39

[Noa09] Andreas Noack. Modularity clustering is force-directed layout. *Physical Review E*, 79(2), 2009. DOI: 10.1103/PhysRevE.79.026102. 52

[NVR16] Paul Neculoiu, Maarten Versteegh, and Mihai Rotaru. Learning text similarity with siamese recurrent networks. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 148–157, 2016. DOI: 10.18653/v1/W16-1617. 40

[NW70] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970. 14

[OSC15] Open Science Collaboration. Estimating the reproducibility of psychological science. *Science*, 349(6251), 2015. DOI: 10.1126/science.aac4716. 21

[Ott76] Karl J. Ottenstein. An algorithmic approach to the detection and prevention of plagiarism. *ACM SIGCSE Bulletin*, 8(4):30–41, 1976. DOI: 10.1145/382222.382462. 74

[PBF54] H.W. Peterson, T.G. Birdsall, and W. Fox. The theory of signal detectability. *Transactions of the IRE Professional Group on Information Theory*, 4(4):171–212, 1954. DOI: 10.1109/TIT.1954.1057460. 31

[Pen11] Roger D. Peng. Reproducible research in computational science. *Science*, 334(6060):1226–1227, 2011. DOI: 10.1126/science.1213847. 21

[Ple18] Hans Ekkehard Plesser. Reproducibility vs. replicability: a brief history of a confused terminology. *Frontiers in Neuroinformatics*, 11(76), 2018. DOI: 10.3389/fninf.2017.00076. 21

[PNN$^+$09] Nam H Pham, Hoan Anh Nguyen, Tung Thanh Nguyen, Jafar M. Al-Kofahi, and Tien N. Nguyen. Complete and accurate clone detection in graph-based models. In *Proceedings of the IEEE 31st International Conference on Software Engineering*, ICSE 2009, pages 276–286. IEEE, 2009. DOI: 10.1109/ICSE.2009.5070528. 74

[PRWZ02] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL 2002, pages 311–318. Association for Computational Linguistics, 2002. DOI: 10.3115/1073083.1073135. 13

[PS07] Jakub Piskorski and Marcin Sydow. Usability of string distance metrics for name matching tasks in polish. In *Proceedings of the 3rd Language & Technology Conference*, 2007. 18, 19, 35

[PSPdC19] Davi Pereira-Santos, Ricardo Bastos Cavalcante Prudêncio, and André C.P.L.F. de Carvalho. Empirical investigation of active learning strategies. *Neurocomputing*, 326:15–27, 2019. DOI: 10.1016/j.neucom.2017.05.105. 47

[PWH18] Didik Dwi Prasetya, Aji Prasetya Wibawa, and Tsukasa Hirashima. The performance of text similarity algorithms. *International Journal of Advances in Intelligent Informatics*, 4(1):63–69, 2018. DOI: 10.26555/ijain.v4i1.152. 19, 21

[RBS13] Dhavleesh Rattan, Rajesh Bhatia, and Maninder Singh. Software clone detection: A systematic review. *Information and Software Technology*, 55(7):1165–1199, 2013. DOI: 10.1016/j.infsof.2013.01.008. 71, 72

[RC07] Chanchal Kumar Roy and James R. Cordy. A survey on software clone detection research. Technical Report 541, Queen's School of Computing, 2007. 72

[RC08] Chanchal Kumar Roy and James R. Cordy. NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. In *Proceedings of the 16th IEEE International Conference on Program Comprehension*, ICPC 2008, pages 172–181. IEEE, 2008. DOI: 10.1109/ICPC.2008.41. 73

[RKP$^+$19] Chaiyong Ragkhitwetsagul, Jens Krinke, Matheus Paixao, Giuseppe Bianco, and Rocco Oliveto. Toxic code snippets on stack overflow. *IEEE Transactions on Software Engineering*, 2019. DOI: 10.1109/TSE.2019.2900307. 71

[RRV13] Antonio Reyes, Paolo Rosso, and Tony Veale. A multidimensional approach for detecting irony in twitter. *Language resources and evaluation*, 47(1):239–268, 2013. DOI: 10.1007/s10579-012-9196-x. 18

[RSR$^+$19] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2019, 1910.10683. 103

[RYW$^+$19] Emily Reif, Ann Yuan, Martin Wattenberg, Fernanda B Viegas, Andy Coenen, Adam Pearce, and Been Kim. Visualizing and measuring the geometry of BERT. In *Advances in Neural Information Processing Systems 32*, NeurIPS 2019, pages 8592–8600. Curran Associates, Inc., 2019. 79

[SB02] Sunita Sarawagi and Anuradha Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 2002, pages 269–278, 2002. DOI: 10.1145/775047.775087. 39

[SB18] Luzi Sennhauser and Robert C. Berwick. Evaluating the ability of LSTMs to learn context-free grammars. In *Proceedings of the EMNLP Workshop BlackboxNLP*, 2018. DOI: 10.18653/v1/W18-5414. 82

[Set12] Burr Settles. *Active Learning.* Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012. DOI: 10.2200/S00429ED1V01Y201207AIM018. 22, 47

[SFL$^+$18] Vaibhav Saini, Farima Farmahinifarahani, Yadong Lu, Pierre Baldi, and Cristina V. Lopes. Oreo: Detection of clones in the twilight zone. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, pages 354—-365, New York, NY, USA, 2018. ACM. DOI: 10.1145/3236024.3236026. 26, 41

[SGADA18] Paul Suganthan GC, Adel Ardalan, AnHai Doan, and Aditya Akella. Smurf: self-service string matching using random forests. *Proceedings of the VLDB Endowment*, 12(3):278–291, 2018. DOI: 10.14778/3291264.3291272. 39

[SK16a] Abdullah Sheneamer and Jugal Kalita. Semantic clone detection using machine learning. In *Proceedings of the 15th IEEE International Conference on Machine Learning and Applications*, ICMLA 2016, pages 1024–1028. IEEE, 2016. DOI: 10.1109/ICMLA.2016.0185. 26, 83

[SK16b] Abdullah Sheneamer and Jugal Kalita. A survey of software clone detection techniques. *International Journal of Computer Applications*, 137(10):1–21, 2016. DOI: 10.5120/ijca2016908896. 73

[SMFM18] Rui Santos, Patricia Murrieta-Flores, and Bruno Martins. Learning to combine multiple string similarity metrics for effective toponym matching. *International Journal of Digital Earth*, 11(9):913–938, 2018. DOI: 10.1080/17538947.2017.1371253. 19, 21

[SMW15] Yufei Sun, Liangli Ma, and Shuang Wang. A comparative evaluation of string similarity metrics for ontology alignment. *Journal of Information &*

Computational Science, 12(3):957–964, 2015. DOI: 10.12733/jics20105420. 19, 35

[SN12] Tommaso Soru and Axel-Cyrille Ngonga Ngomo. Active learning of domain-specific distances for link discovery. In *Second Joint International Semantic Technology Conference*, JIST 2012, pages 97–112, Berlin, Heidelberg, 2012. Springer. DOI: 10.1007/978-3-642-37996-3_7. 39

[SPH$^+$11] Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP 2011, pages 151–161. Association for Computational Linguistics, 2011. 76

[SR15a] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10(3):e0118432, 2015. DOI: 10.1371/journal.pone.0118432. 32

[SR15b] Jeffrey Svajlenko and Chanchal Kumar Roy. Evaluating clone detection tools with BigCloneBench. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution*, ICSME 2015, pages 131–140. IEEE, 2015. DOI: 10.1109/ICSM.2015.7332459. 26, 83

[SR16] Jeffrey Svajlenko and Chanchal Kumar Roy. BigCloneEval: A clone detection tool evaluation framework with BigCloneBench. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution*, ICSME 2016, pages 596–600. IEEE, 2016. DOI: 10.1109/ICSME.2016.62. 26, 83, 102

[SSK05] Giorgos Stoilos, Giorgos Stamou, and Stefanos Kollias. A string metric for ontology alignment. In *Proceedings of the the 4th International Semantic Web Conference*, ISWC 2005, pages 624–637. Springer, Springer Berlin Heidelberg, 2005. DOI: 10.1007/11574620_45. 18

[SSS$^+$16] Hitesh Sajnani, Vaibhav Saini, Jeffrey Svajlenko, Chanchal Kumar Roy, and Cristina V. Lopes. SourcererCC: Scaling code clone detection to big-code. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE 2016, pages 1157–1168, 2016. DOI: 10.1145/2884781.2884877. 73, 74

[STSC19] Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron Courville. Ordered neurons: Integrating tree structures into recurrent neural networks. In *International Conference on Learning Representations*, ICLR 2019, 2019, 1810.09536. URL: `https://openreview.net/forum?id=B1l6qiR5F7`. 79

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*, NIPS 2014, pages 3104–3112. Curran Associates, Inc., 2014. 79

[SW81] Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981. DOI: 10.1016/0022-2836(81)90087-5. 14

[SW18] Jingyu Shao and Qing Wang. Active blocking scheme learning for entity resolution. In *Proceedings of the 22nd Pacific-Asia Conference on Knowledge Discovery and Data Mining*, PAKDD 2018, pages 350–362. Springer, 2018. DOI: 10.1007/978-3-319-93037-4_28. 23, 39

[Swe73] John A Swets. The relative operating characteristic in psychology: a technique for isolating effects of response bias finds wide use in the study of perception and cognition. *Science*, 182(4116):990–1000, 1973. DOI: 10.1126/science.182.4116.990. 31

[Tej02] Sheila Tejada. *Learning High Accuracy Rules for Object Identification.* PhD thesis, University of Southern California, 2002. 38, 61, 62

[TKM01] Sheila Tejada, Craig A. Knoblock, and Steven Minton. Learning object identification rules for information integration. *Information Systems*, 26(8):607–633, 2001. DOI: 10.1016/S0306-4379(01)00042-4. 38, 57

[TKM02] Sheila Tejada, Craig A Knoblock, and Steven Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, KDD 2002, pages 350–359, 2002. DOI: 10.1145/775047.775099. 36, 38

[TO18] Corentin Tallec and Yann Ollivier. Can recurrent neural networks warp time? In *International Conference on Learning Representations*, ICLR 2019, 2018, 1804.11188. URL: `https://openreview.net/forum?id=SJcKhk-Ab`. 89

[TSM15] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, ACL/IJCNLP 2015, pages 1556–1566, Beijing, China, 2015. Association for Computational Linguistics. DOI: 10.3115/v1/P15-1150. 40, 81, 85, 89

[Tur37] Alan Mathison Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 01 1937. DOI: 10.1112/plms/s2-42.1.230. 72

[Tve77] Amos Tversky. Features of similarity. *Psychological review*, 84(4), 1977. DOI: 10.1037/0033-295X.84.4.327. 5, 6

[TWB+18] Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. Deep learning similarities from different representations of source code. In *Proceedings of the IEEE/ACM 15th International Conference on Mining Software Repositories*, MSR 2018, pages 542–553. IEEE, 2018. DOI: 10.1145/3196398.3196431. 26, 42, 84

[UL16] Evgeniya Ustinova and Victor Lempitsky. Learning deep embeddings with histogram loss. In *Advances in Neural Information Processing Systems 29*, NIPS 2016, pages 4170–4178. Curran Associates, Inc., 2016. 89, 100

[VDK+20] Gilles Vandewiele, Isabelle Dehaene, György Kovács, Lucas Sterckx, Olivier Janssens, Femke Ongenae, Femke De Backere, Filip De Turck, Kristien Roelens, Johan Decruyenaere, Sofie Van Hoecke, and Thomas Demeester. Overly optimistic prediction results on imbalanced data: Flaws and benefits of applying over-sampling. *preprint*, 2020, 2001.06296. 26

[VN11] Tobias Vogel and Felix Naumann. Instance-based 'one-to-some' assignment of similarity measures to attributes. In *On the Move to Meaningful Internet Systems*, OTM 2011, pages 412–420, Berlin, Heidelberg, 2011. Springer. DOI: 10.1007/978-3-642-25109-2_27. 35

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, NIPS 2017, pages 5998–6008. Curran Associates, Inc., 2017. 78

[WDS+19] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *preprint*, 2019, 1910.03771. 102

[Wei04] Gary M. Weiss. Mining with rarity: a unifying framework. *ACM SigKDD Explorations Newsletter*, 6(1):7–19, 2004. DOI: 10.1145/1007730.1007734. 22, 23, 24

[Win90] William E. Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage, 1990. URL: `https://eric.ed.gov/?id=ED325505`. 14

[WK19] John Wieting and Douwe Kiela. No training required: Exploring random encoders for sentence classification. In *International Conference on Learning Representations*, ICLR 2019, 2019, 1901.10444. URL: `https://openreview.net/forum?id=BkgPajAcY7`. 93

[WL17] Huihui Wei and Ming Li. Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, IJCAI 2017, pages 3034–3040, 2017. DOI: 10.24963/ijcai.2017/423. 26, 42, 83

[WM05] Richard Wettel and Radu Marinescu. Archeology of code duplication: Recovering duplication chains from small duplication fragments. In *Proceedings of the Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, SYNASC 2005, pages 8–15. IEEE, 2005. DOI: 10.1109/SYNASC.2005.20. 73

[WPN+19] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. SuperGLUE: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems 32*, NeurIPS 2019, pages 3266–3280. Curran Associates, Inc., 2019. 103

[WRP19] Yaza Wainakh, Moiz Rauf, and Michael Pradel. Evaluating semantic representations of source code. *preprint*, 2019, 1910.05177. 43, 76, 97, 100

[WS15] Fei Wang and Jimeng Sun. Survey on distance metric learning and dimensionality reduction in data mining. *Data mining and knowledge discovery*, 29(2):534–564, 2015. DOI: 10.1007/s10618-014-0356-z. 21

[WSM$^+$18] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the EMNLP Workshop BlackboxNLP*, pages 353–355. Association for Computational Linguistics, 2018. DOI: 10.18653/v1/W18-5446. 103

[WTVP16] Martin White, Michele Tufano, Christopher Vendome, and Denys Poshyvanyk. Deep learning code fragments for code clone detection. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ASE 2016, pages 87–98. IEEE, 2016. DOI: 10.1145/2970276.2970326. 41, 42, 84, 100, 102

[WYLZ19] Jin Wang, Liang-Chih Yu, K Robert Lai, and Xuejie Zhang. Investigating dynamic routing in tree-structured LSTM for sentiment analysis. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, EMNLP-IJCNLP 2019, pages 3423–3428. Association for Computational Linguistics, 2019. DOI: 10.18653/v1/D19-1343. 93

[XAF13] Sicheng Xiong, Javad Azimi, and Xiaoli Z Fern. Active learning of constraints for semi-supervised clustering. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):43–54, 2013. DOI: 10.1109/TKDE.2013.22. 39, 62

[YDC$^+$18] Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. Learning to mine aligned code and natural language pairs from Stack Overflow. In *Proceedings of the IEEE/ACM 15th International Conference on Mining Software Repositories*, MSR 2018, pages 476–486. ACM, 2018. DOI: 10.1145/3196398.3196408. 103

[YHMH19] Carl Yang, Do Huy Hoang, Tomas Mikolov, and Jiawei Han. Place deduplication with embeddings. In *Proceedings of the World Wide Web Conference*, WWW 2019, pages 3420–3426, 2019. DOI: 10.1145/3308558.3313456. 41

[YJS07] Liu Yang, Rong Jin, and Rahul Sukthankar. Bayesian active distance metric learning. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, UAI 2007, pages 442–449, 2007. 39, 47

[YMSL17] Di Yang, Pedro Martins, Vaibhav Saini, and Cristina Lopes. Stack overflow in github: any snippets there? In *Proceedings of the IEEE/ACM 14th International Conference on Mining Software Repositories*, MSR 2017, pages 280–290. IEEE, 2017. DOI: 10.1109/MSR.2017.13. 71

[ZGH$^+$18] Dongxiang Zhang, Long Guo, Xiangnan He, Jie Shao, Sai Wu, and Heng Tao Shen. A graph-theoretic fusion framework for unsupervised entity resolution. In *Proceedings of the 34th IEEE International Conference on Data Engineering*, ICDE 2018, pages 713–724. IEEE, 2018. DOI: 10.1109/ICDE.2018.00070. 19

[ZH18] Gang Zhao and Jeff Huang. DeepSim: deep learning code functional similarity. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, pages 141–151, 2018. DOI: 10.1145/3236024.3236068. 26, 41, 83, 95

[ZKW$^+$19] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. BERTScore: Evaluating text generation with BERT. In *International Conference on Learning Representations*, ICLR 2020, 2019, 1904.09675. URL: `https://openreview.net/forum?id=SkeHuCVFDr`. 13

[ZWZ$^+$19] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Kaixuan Wang, and Xudong Liu. A novel neural source code representation based on abstract syntax tree. In *Proceedings of the IEEE/ACM 41st International Conference on Software Engineering*, ICSE 2019, pages 783–794. IEEE, 2019. DOI: 10.1109/ICSE.2019.00086. 26, 42, 83, 102