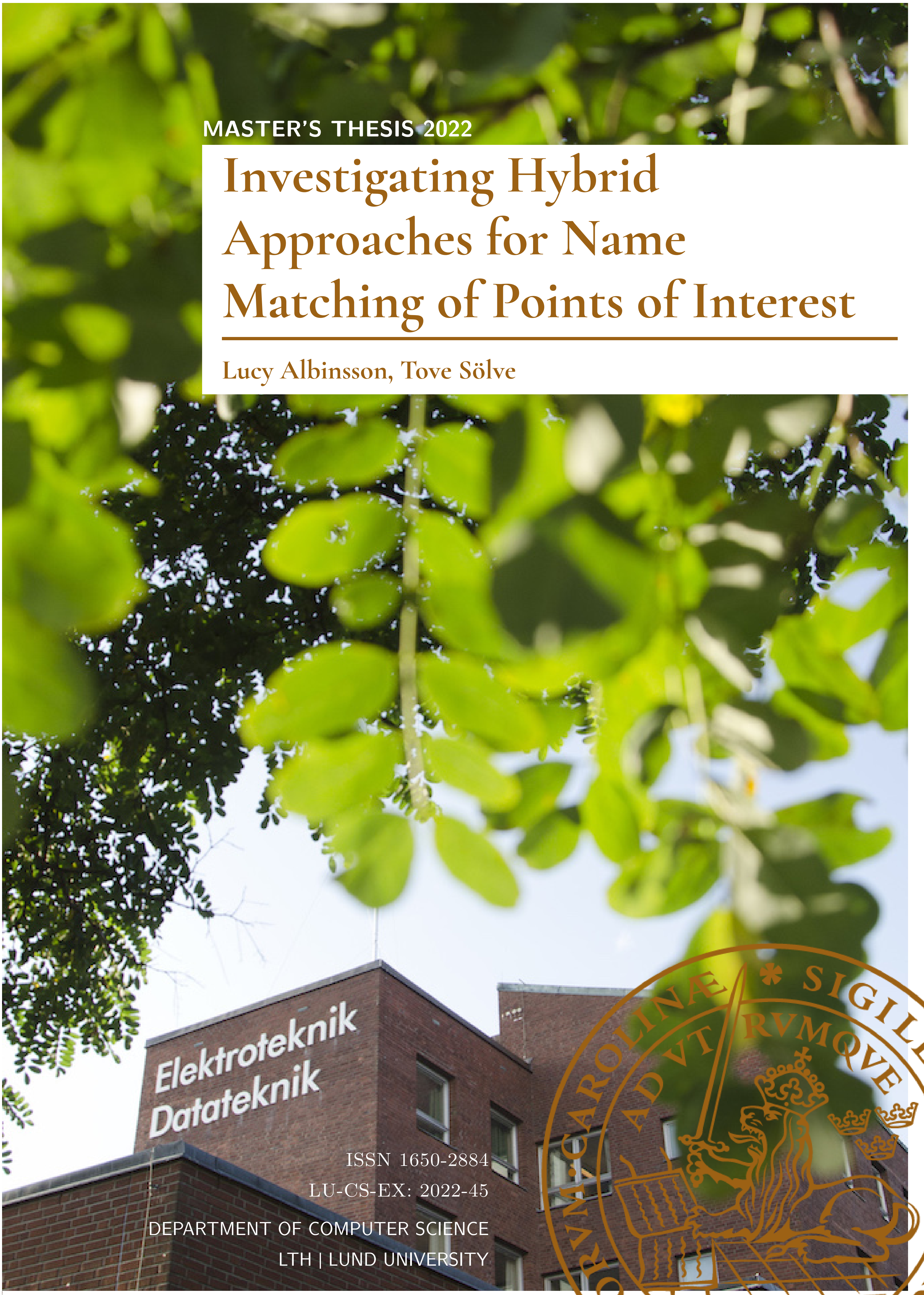


MASTER'S THESIS 2022

Investigating Hybrid Approaches for Name Matching of Points of Interest

Lucy Albinsson, Tove Sölve

ISSN 1650-2884
LU-CS-EX: 2022-45

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2022-45

Investigating Hybrid Approaches for Name Matching of Points of Interest

Undersökning av hybrida metoder för namnbaserad matching av intressepunkter

Lucy Albinsson, Tove Sölve

Investigating Hybrid Approaches for Name Matching of Points of Interest

Lucy Albinsson
lu2767al-s@student.lu.se

Tove Sölve
to5185so-s@student.lu.se

July 6, 2022

Master's thesis work carried out at AFRY.

Supervisors: Hampus Londögård, hampus.londogard@afry.com
Dennis Medved, dennis.medved@med.lu.se

Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

Determining whether or not two Points of Interest are the same entity can be a difficult task due to inconsistent information between different data sources. This can cause problems such as duplicated or removed Points of Interest in map services. This thesis investigates hybrid approaches for matching Points of Interest based on their names and coordinates. We present hybrid approaches based on state-of-the-art similarity functions, semantics and machine learning. The highest performance was achieved using a random forest classifier with features based on hybrid similarity functions, with an F_1-score of 0.986 and error reduction of 67% compared to state-of-the-art approaches.

Keywords: name matching, hybrid similarity functions, points of interest, algorithms, semantics, word embeddings, machine learning

Acknowledgements

We would like to thank Dennis Medved for continuously answering all our questions regarding the thesis in general and guidance when needed. We would like to thank Hampus Londögård for feedback and knowledge in the field. We would also like thank AFRY for giving us the opportunity to write our thesis with them. We would especially like to thank the IT Solutions department in Malmö for being supportive during our time at AFRY. Last but not least, we would like to thank everyone who provided valuable feedback on the thesis work and report.

Contents

Chapter 1

Introduction

This chapter gives an introduction to the problem investigated in the thesis. Initially, the background to the problem is introduced, followed by the purpose and problem formulation. Finally, related work, outline and work distribution is presented.

1.1 Background

Map services have become essential to everyday life. Maps require large amounts of geographical data to provide necessary functionalities such as searching for locations and navigating a city. It is not uncommon that the data is collected from multiple data sources. Since there is no guarantee of a unique identifier between data sources it can be difficult to determine how these overlap. Crowdsourced data is increasing in popularity, leading to problems such as inconsistent naming, incorrect data and diverging information. For example, in OpenStreetMap (OSM), a crowdsourced database containing geographical data, volunteers can contribute with limited regulation, leading to variations in site names, addresses, coordinates and other information. A specific location that is of interest on a map such as a restaurant, shop, library, tourist attraction or gas station, is referred to as a *Point of Interest* (POI). POIs are good examples of data that can be inconsistent. For example, "Fin's" and "Fins Sushi & Grill" represent the same POI, even though the names differ. Without a unique identifier, the information to determine similarity between POIs is limited and sometimes only the names are available, reducing it to a name matching task.

Name matching can be described as the task of determining if multiple text strings represent the same entity such as a person or a location. Strings can have two aspects of similarity: *lexical* and *semantic*. Lexical similarity is the similarity with respect to characters and words, while semantic similarity refers to similarity in terms of meaning. State-of-the-art similarity functions used for string matching typically use variants of lexical similarity.

Challenges in name matching tasks have been investigated in several NLP communities with varying results. There are several difficulties that need to be handled, such as words being placed in different order or that an added word changes the meaning of the name. Among the state-of-the-art similarity functions, no single algorithm has proven to be favorable in all name matching tasks and none of them seem to solve all difficulties.

In traditional *Natural Language Processing* (NLP), textual context is essential for determining similarity. Due to the lack of context in names, there is a need for further exploration and this thesis aims to investigate POI name matching with limited context.

1.2 Purpose

The purpose of the thesis is to contribute to the work of identifying matching and non-matching POI entities from different geographical data sources, using string comparison algorithms. The aim is to avoid duplicated POIs as well as distinguish between closely located POIs with similar names. Hopefully, the work can also be of interest for other data integration tasks where string comparison algorithms with limited context are used.

1.3 Problem Formulation

The thesis aims to investigate approaches for determining whether or not two POIs from different data sources are the same entity, which can be seen as a classification problem. The goal is to correctly classify if two closely located POIs are the same entity or not, only based on their names and coordinates. We refer to this as POI name matching. The thesis focuses on POIs located in North America with names primarily in English, as there is plenty of previous work for texts in English as well as NLP tools provided specifically for the English language. The thesis mainly focuses on hybrid approaches based on string comparison algorithms. We begin by investigating state-of-the-art similarity functions to see how these can be improved when combined in hybrid similarity functions. We continue by investigating how semantic similarity and machine learning can be used to improve the performance of hybrid approaches.

1.3.1 Research questions

The thesis aims to answer the following research questions:

1. How can hybrid approaches improve performance of state-of-the-art similarity functions for POI name matching?

2. How can semantic similarity be used to improve the performance of hybrid approaches for POI name matching?

3. How can machine learning be used to improve performance of hybrid approaches for POI name matching?

1.4 Related Work

Jiří Kysela [1] investigated string comparison algorithms with the aim to identify the best algorithm for POI name matching. Kysela compared state-of-the-art algorithms which consider two types of lexical similarity functions: *character-based* and *token-based* similarity functions. Kysela's results showed that the token-based cosine similarity achieved the best results for name matching of POIs. Even though Kysela's work did not include hybrid similarity functions specifically, it did investigate several state-of-the-art similarity functions that are used in the hybrid approaches in this thesis. The results are of particular interest for the thesis as it evaluated the similarity functions specifically on POI name matching.

Cohen et al. [2] investigate how character-based, token-based and hybrid similarity functions can be used for general string comparison. The hybrid similarity functions compared are *recursive matching scheme*, suggested by Monge and Elkan [3] and *SoftTFIDF*, suggested by the authors themselves. The SoftTFIDF is based on a combination of cosine similarity and Jaro-Winkler similarity. The evaluation was done on several data sets with various content. The results showed that cosine similarity performed the best among the token-based similarity functions and no single character-based function could be concluded as the most successful as these performed well on different tasks. The SoftTFIDF algorithm performed the best among the hybrid similarity functions, but also among all similarity functions evaluated. The authors emphasized that the performance of the algorithms is task dependent, meaning that no single similarity function is the most successful for all tasks. Even though Cohen et al. investigated string matching in general, the promising results using hybrid approaches rather than individual similarity functions indicate these might provide good performance for the purpose of POI name matching as well.

Santos et al. [4] investigated state-of-the-art similarity functions for matching names of toponyms, i.e. names of geographical places such as cities, islands etc. Their findings showed relatively small differences in performance among the algorithms and they suggested that achieving good results require careful tuning of the thresholds. They examined supervised machine learning for combining multiple similarity metrics and for avoiding manual tuning of thresholds. The results showed that ensembles of decision trees significantly outperform the individual similarity functions. Even though Santos et al. focused on matching names of toponyms and not POIs, the results are of interest when investigating the machine learning approaches in this thesis.

1.5 Outline

We begin by introducing the theory necessary for understanding the work in the thesis. We then describe the methods, data and tools used for approaching the problem, followed by explaining the implementation of the approaches. Furthermore, we present the results from evaluating the approaches followed by a discussion of these. Finally, we conclude the work done in the thesis by answering the research questions and suggesting future work.

1.6 Work Distribution

All work in this thesis has been completed in full collaboration between the authors.

Chapter 2
Theory

This chapter introduces the theory needed for understanding the work in the thesis. Fundamental concepts as similarity functions, text pre-processing and tokenization are described, followed by presenting theory related to machine learning, word embeddings and evaluation metrics.

2.1 Similarity Functions

There are many well-known algorithms for quantifying similarity between texts. These are referred to as *similarity functions*. These can be categorized into *character-based similarity functions* and *token-based similarity functions*, which will be described further in the following sections. The two types use different techniques which come with advantages and disadvantages [5, 6].

2.1.1 Character-based Similarity Functions

Character-based similarity functions use the structure of the characters in the strings to quantify similarity. These functions effectively capture typographical errors, but have problems with other types of errors, such as variations of word order in the strings [5]. As an example, consider the two strings "Florida Orthopedic Clinic" and "Orthopaedic Clinic Florida" which according to character-based similarity have a low similarity due to the word order, even though the strings are very similar. Even so, character-based similarity functions have shown promising results for string matching tasks [1]. Three state-of-the-art character-based similarity functions are described in the following sections.

Levenshtein Similarity

Levenshtein similarity is a character-based similarity function. It is based on *Levenshtein distance*, which counts the number of operations necessary to transform one text string into another. There are three types of operations: insertion, deletion and substitution, all of equal cost. The Levenshtein distance is defined in formula 2.1 [1, 6].

$$lev\_dist_{s_1,s_2}(i,j) = \begin{cases} max(i,j) & if min(i,j) = 0 \\ min\begin{cases} lev_{s_1,s_2}(i-1,j)+1 \\ lev_{s_1,s_2}(i,j-1)+1 \\ lev_{s_1,s_2}(i-1,j-1)+1_{(s_{1_i} \neq s_{2_j})} \end{cases} & otherwise. \end{cases} \tag{2.1}$$

where:

s_1 is the first string,

s_2 is the second string,

i is the index of a character in s_1,

j is the index of a character in s_2.

Levenshtein similarity can be calculated using Levenshtein distance as defined in formula 2.2 [7].

$$lev\_sim_{s_1,s_2}(i,j) = 1 - \frac{lev\_dist_{s_1,s_2}(i,j)}{max\_len(s_1,s_2)} \tag{2.2}$$

where:

s_1 is the first string,

s_2 is the second string,

i is the index of a character in s_1,

j is the index of a character in s_2.

Jaro Similarity

The idea behind *Jaro similarity* is to identify common characters between two text strings. The characters are considered common if they appear at the same position in both strings. If the characters appear in both strings but not at the same position, they are considered a transposition. Jaro similarity function is defined in formula 2.3 [1, 5, 8].

$$jaro(s_1,s_2) = 1 - \frac{1}{3} * \left(\frac{c}{|s_1|} + \frac{c}{|s_2|} + \frac{c+t}{c}\right) \tag{2.3}$$

where:

s_1 is the first string,

s_2 is the second string,

c is the number of common characters in s_1 and s_2,

t is the number of transpositions required to transform s_1 to s_2,

$|s_1|$ is the total number of characters in s_1,

$|s_2|$ is the total number of characters in s_2.

Jaro-Winkler Similarity

Jaro-Winkler similarity is a variant of Jaro similarity. Based on the assumption that the beginning of a string is often more relevant for determining similarity, Jaro-Winkler assigns higher weights to common characters in the prefixes. In Jaro-Winkler, a prefix is considered to consist of maximum four characters. Jaro-Winkler similarity is defined in formula 2.4 [1, 8].

$$jaro\_winkler(s_1, s_2) = jaro(s_1, s_2) + \frac{s}{10} * (1 - jaro(s_1, s_2)) \tag{2.4}$$

while valid for $0 \leq s \leq 4$, where:

s_1 is the first string,

s_2 is the second string,

$|s_1|$ is the total number of characters in s_1,

$|s_2|$ is the total number of characters in s_2,

s is the number of common characters in the string prefixes.

2.1.2 Token-based Similarity Functions

Token-based similarity functions use token sets of the strings to determine similarity [6, 9]. A token is a sub-sequence of characters in a string. The token sets can be generated differently depending on the *tokenization* method applied, which will be further described in section 2.3. Set operations such as intersect [9, 10] are used to calculate the similarity between the token sets. Two strings have a high similarity if their token sets have a large overlap, regard of the length of the individual tokens. [6, 10].

A disadvantage of token-based similarity functions is that they only consider exact matches of tokens. If the strings contain inconsistencies and typographical errors, their tokens will also contain these errors and therefore not be considered matching tokens. For example, consider the two strings "Florida Orthopedic Clinic" and "Orthopaedic Clinic Florida". The strings can be tokenized respectively into the token sets {"Florida", "Orthopedic", "Clinic"} and {"Orthopaedic", "Clinic", "Florida"}. The tokens "Orthopedic" and "Orthopaedic" will not be considered matching tokens and the similarity between the strings will therefore be relatively low, even though the strings are in fact very similar [6, 9].

Token-based similarity functions are not sensitive to the positions of tokens in the string, since the order of the tokens is neglected in the token sets [6, 9]. Consider again the example with the two strings "Florida Orthopedic Clinic" and "Orthopaedic Clinic Florida". The token “Florida” will be considered matching between the sets, regardless of the position in the strings.

Two token-based similarity functions that are considered state-of-the-art for string comparison [1, 6] are presented in the following sections.

Jaccard Similarity

The *Jaccard similarity* between two text strings is calculated as the size of the intersection of their token sets divided by the size of the union of their token sets [6, 11]. Non-matching tokens are ignored when comparing the token sets [1]. Jaccard similarity is defined in formula 2.5 [6].

$$jaccard(s_1, s_2) = 1 - \frac{|T_1 \cap T_2|}{|T_1 \cup T_2|} = 1 - \frac{|T_1 \cap T_2|}{|T_1| + |T_2| - |T_1 \cap T_2|} \tag{2.5}$$

s_1 is the first string,

s_2 is the second string,

T_1 is the token set of s_1,

T_2 is the token set of s_2.

Cosine Similarity

Cosine similarity is used to compute the similarity between two vectors. To use cosine similarity as a token-based similarity function on strings, the tokens need to be represented as numerical vectors. A vector representing a string could for example be created using *word embeddings*, which will be further described in section 2.5. The cosine similarity is defined as the angle between two vectors, which is their dot product divided by the product of their size [1]. Cosine similarity is defined in formula 2.6 [1].

$$cosine(s_1, s_2) = \cos\theta = \frac{A \cdot B}{|A||B|} = \frac{\sum_{i=0}^{n} A_i B_i}{\sqrt{\sum_{i=0}^{n} A_i^2}\sqrt{\sum_{i=0}^{n} B_i^2}} \tag{2.6}$$

s_1 is the first string,

s_2 is the second string,

A is the vector representation of the tokens in s_1,

B is the vector representation of the tokens in s_2.

2.2 Text Pre-processing

Text pre-processing is an important step for improving performance in NLP tasks. It involves structuring and cleaning text and is often done as a preparation before performing the main task.

Lower-casing, i.e. converting all characters to lower case, and *noise-removal*, i.e. removing special characters, are two simple steps that are commonly used in pre-processing, in order to achieve better results [12]. There are several techniques used for text pre-processing. The following sections present two of the most important techniques; *stop word removal* and *stemming* [12, 13].

2.2.1 Stop Word Removal

Words can contribute unequally to the meaning of a text. Words that do not contribute much to the meaning and occur frequently are referred to as stop words. These can be removed to avoid negative impact on the context. What words to consider as stop words can vary depending on the context. However, there are some words that are considered stop words in the general context, for example "and", "or" and "the" [14].

2.2.2 Stemming

It is not unusual that words in a text are not written in its root form. For example, the words "waited" and "waiting" are both variants of the root word "wait". Considering these words as the same can be favourable for interpreting the meaning of a text. Stemming is the process of converting variants of words into their root form to have a common representation [12, 14].

Stemming can be performed using different algorithms. The most common is the *Porter algorithm* [15], which is a rule-based algorithm that works well for the English language. The *Snowball* stemmer, also known as *Porter2*, is based on Porter with slightly different rules and minor improvements [12].

2.3 Tokenization

Tokenization is the procedure of splitting pieces of texts, such as documents, sentences or words, into shorter sequences of characters, referred to as tokens. Depending on the used tokenization method, a token could for example be a sequence of words, subwords or characters [16]. The technique behind tokenization is to use one or more delimiters to define where to split the text string. A simple tokenization method is *whitespace tokenization*, i.e. splitting the text using the whitespace character as delimiter to create tokens from each word in a text. A more advanced method is *subword tokenization*, where the words are split into shorter tokens [17]. The following sections present two types of subword tokenizations.

2.3.1 Byte-pair Encoding Tokenization

A widely used subword tokenization is a technique based on the data compression algorithm *Byte-Pair Encoding* (BPE) by Gage [18]. The idea behind the tokenization technique is to break down the text to characters and iteratively merge commonly used sequences into symbols. Starting off with all characters individually as symbols, in each iteration, the most frequent symbol pair is replaced with the combination of the merged symbols, i.e. a merge operation. There are two stopping criteria when applying BPE, the maximum number of iterations and the vocabulary size, which is the sum of the number of merge operations and the number of characters in the original vocabulary [19]. A small vocabulary size will result in fewer merge operations and shorter tokens such as unigrams, bigrams and trigrams, while a large vocabulary size will create tokens representing more frequent words. The advantage of using a small vocabulary size is that less data is needed to learn the tokenization, but it will also be less likely to be merged into meaningful tokens. In contrast, a large vocabulary size is more likely to result in tokens representing frequent words, with the down side of requiring a larger amount of data to learn from [20].

2.3.2 WordPiece

WordPiece tokenization is a subword tokenization which, similar to BPE, builds a vocabulary by iteratively adding combinations of characters or sequences. The difference from BPE is that WordPiece adds the combination that maximizes the likelihood of the training data once added, instead of the most frequent one [21].

2.4 Machine Learning

Machine learning is a field of computer science that teaches systems how to learn and improve by experience, rather than being explicitly programmed. In the context of machine learning, experience refers to information or data. The idea is for a system to repetitively learn from the data and build a model that is able to find patterns and make accurate predictions for various tasks. The data used to train the model is central for the learning process and often a large amount of data is required for good results [22].

There are three main machine learning paradigms; *supervised* learning, *unsupervised* learning and *reinforcement* learning. Supervised machine learning refers to learning from data that is labeled as the correct or incorrect outcome. In unsupervised machine learning, no labeled data is used in training. Instead, the model learns by identifying patterns and correlations within the provided data and groups similar data, referred to as *clustering*, with no knowledge of what it represents. Reinforcement learning uses an environment with parameters defining beneficial and non-beneficial activity, making it more controlled than unsupervised learning but still without labeled data [22].

Ideally, a model should learn from training data and based on this be able to perform well on unseen data. If the model is too closely aligned with the training data, it will perform well on this data specifically but not in general. This is referred to as *overfitting* and has a negative impact on performance of the models on new data [23].

2.4.1 Cross-validation

For supervised machine learning, *cross-validation* can help prevent the problem of overfitting and be used for hyperparameter tuning. Often in machine learning, the data is split into training and testing data. The cross-validation is performed on the training data by splitting it into smaller sets, referred to as *folds*. For a *k*-fold cross-validation the model is trained *k* times, using *k-1* folds for training and leaving one of the folds out for validation. For each model trained, a different fold is used for validation. The procedure is illustrated in the Figure 2.1. Cross-validation can be computationally expensive. However, it does not waste too much data in the training process, which is an advantage when there are relatively few samples in the data set [24].

Stratified *k*-fold cross-validation is an extension of the cross-validation technique, meaning that the percentage of samples of each target class in each set are the same as in the complete set. This is especially useful when working with imbalanced data [24].

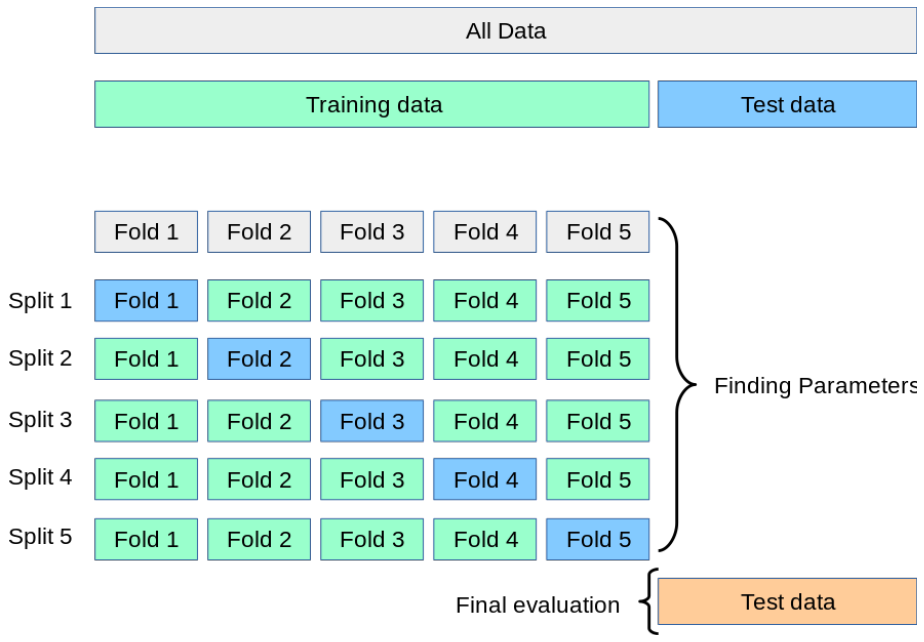


Figure 2.1: Cross-validation used on the training data [24].

2.4.2 Decision Trees

The supervised machine learning method *decision trees* can be used for both classification and regression tasks. The model learns simple decision rules from the features throughout the training. Decision trees have a tree-like structure, with internal nodes and decision nodes, the leaves. At each internal node, a decision is made according to a decision rule. The rule is usually a condition based on feature variables in the training data. The leaves are assigned one of the classes [25].

When training the trees, the algorithm needs a criterion to calculate the *information gain*, the information needed to decide how to split a node. This criterion measures the quality of a split. The measure used as criterion is often *Gini index* or *entropy* which are often similar [26].

The tree is constructed by splitting the training data according to the best split iteratively until all data points are correctly classified or a stopping criterion is reached. A stopping criterion could for example be the depth of the tree [25].

A disadvantage with decision trees is that the model can be sensitive to small variations in the training data. This problem can be solved using ensembles of trees [25].

2.4.3 Ensemble Learning of Decision Trees

General ensemble methods combine a number of models to make better predictions than a single model. Compared to a single decision tree, ensemble decision trees combine several decision trees and tend to be less data sensitive and more flexible. The method has shown successful results for a large variety of problems [25]. *Bagging* and *boosting* are two ensemble decision tree techniques. Bagging trains several models separately in parallel. Boosting, on the other hand, trains the models sequentially, where each model learns from the previous one. Since the trees are sequentially connected, the training of boosted trees can be slow [25].

Random Forest

Random forest is an ensemble learning method based on the bagging technique, that can be used for both classification and regression tasks. A random forest model consists of a collection of decision trees. Each decision tree predicts independently and the final class is selected by the majority of predicted classes from all decision trees. Using a collection of trees prevents an overfitted model and performs better in terms of accuracy. There are two aspects of randomness in random forests; each tree is trained from a bootstrap sample and the feature selection when growing the tree is also random [27]. The structure of a random forest model can be seen in Figure 2.2.

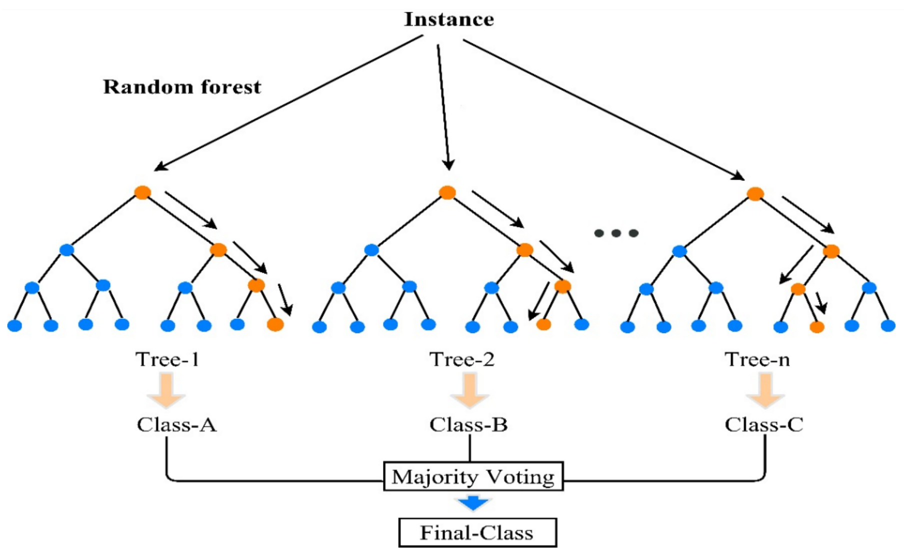


Figure 2.2: The random forest architecture [28].

Gradient Boosted Trees

Gradient boosted trees is an ensemble learning method based on the boosting technique in combination with *gradient descent*, a mathematical iterative optimization algorithm for finding a local minimum of a differential function. The idea of is to combine and improve weak learners to create a strong learner. A visualization of the gradient boosted trees architecture can be seen in Figure 2.3. The training is performed iteratively where the misclassified observations are assigned new weights in each iteration. The goal is to reduce the error in each iteration. The weights are calculated using a loss function, which can differ depending on task [25].

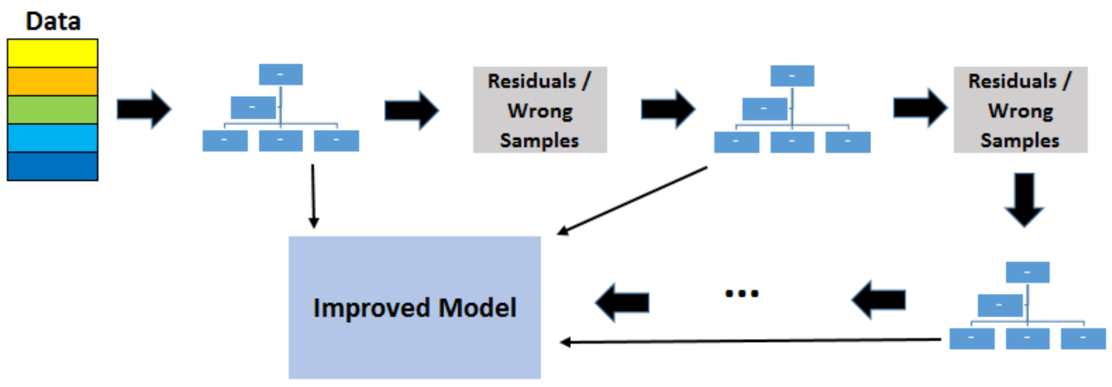


Figure 2.3: The gradient boosted trees architecture [29].

2.4.4 Artificial Neural Networks

Artificial neural networks (ANNs) are inspired by the structure and processes in a biological brain. The main building blocks of a neural network are referred to as neurons, which are simply learning units. A neuron takes an input, applies some logic to it and outputs a result. An ANN is a collection of neurons organized in different layers, where the output from one layer is passed as the input to another. A type of ANN where the information is flowing forward in the network is often referred to as *feed-forward networks* [30].

The layers in ANNs often apply different logic to the input. The number of layers in the ANN defines the depth, hence using networks with several layers is referred to as *deep learning*. The last layer in the network is referred to as the *output layer* since it is the only layer that provides the desired output, while the other layers are referred to as *hidden layers*. The width of the model corresponds to the dimensionality of the hidden layers [30].

A neural network is trained by feeding the network with data and successively adjusting weights to match the desired outcome, hence it is a supervised learning technique. To be able to adjust the weights properly, a loss function is used to determine how good the predictions are. The goal with the adjustment is to minimize the loss function, which is often done using an *optimizer*. Often, a gradient based optimizer using *back-propagation* is used. The back-propagation algorithm computes the gradient of the loss function with respect to each weight in the network [30].

Multilayer Perceptron

A *multilayer perceptron* (MLP) is a type of feed-forward ANN. It is a fully connected multilayer neural network where the weight adjustments in the training process is performed using back-propagation. MLP has one or more hidden layers and can be used for both classification and regression tasks [31].

Attention

In artificial neural networks, attention is a technique that mimics cognitive attention of neural networks. The mechanism aims to emphasize the important parts of the input data while diminishing less important parts. The importance is determined using gradient descent. The attention mechanism considers a part of the input sequence in relation to the rest of the sequence and can be context-dependent [32].

Transformers

In the paper "Attention Is All You Need" by Vaswani et al. [32], the *transformer* architecture is proposed, which is a machine learning architecture that uses attention. The original transformer model uses an encoder-decoder-based architecture consisting of encoding and decoding layers with feed-forward neural networks and multiple attention heads. The attention heads iteratively weighs relevant parts of the input using attention. Transformers are designed to handle sequential input data and have in recent years become widely used for NLP-tasks [32]. Figure 2.4 illustrates the transformer architecture.

2.4.5 Shapley Values

The *Shapley value* is used in cooperative game theory to predict outcomes. For a coalition of cooperating players, the Shapley values provide information about how much each player contributes to a final outcome of a game [33].

Lundberg and Lee [34] proposed the *SHapley Additive exPlanations* (SHAP) value as a unified framework for interpreting machine learning predictions using Shapley values. For each feature in a machine learning model, a SHAP value is assigned for a particular prediction. The SHAP value for a given feature can be defined as the average marginal contribution to the outcome, across all permutations [34].

2.5 Word Embeddings

A word embedding is a real-value vector representation of a text, that encodes its properties. The complexity of a word embedding can vary depending on the embedding dimension. Word embeddings are commonly used in NLP tasks [35].

Some word embeddings have the fascinating feature that for words with similar meaning in terms of semantic, lexical and contextual properties, their representations are closer in the vector space [35]. This can be illustrated observing simple vector operations, such as the classic example "king"-"man"+"woman". The operation between these results in a vector very

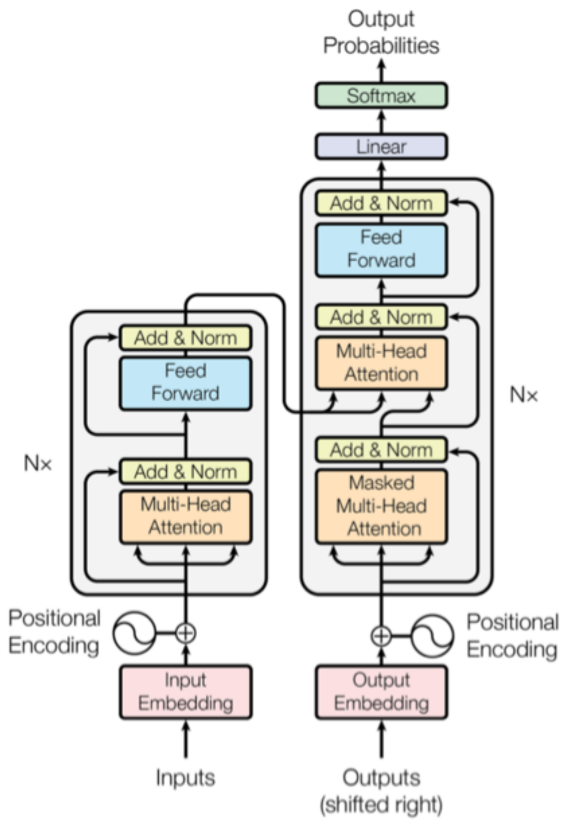


Figure 2.4: The transformer architecture [32].

similar to the vector for the word "queen", illustrating that it captures a gender relationship between the words, as showed in Figure 2.5.

Traditionally, word embeddings have been generated by considering the words statistical properties, such as occurrences and frequency. More modern techniques involve using neural networks to train word embeddings. This enhances the ability of capturing semantic meaning of words in the embeddings, however, tend to require more contextual information and large amounts of data for training.

Tokenization is often used when creating word embeddings. A common problem is that pre-trained embeddings are not able to handle out-of-vocabulary words, i.e. words not included when training and therefore with no embedding. To solve this problem, subword tokenizations, such as BPE and WordPiece, are useful as they can be used as a part of the training. In this way, out-of-vocabulary words can be split into subwords that were seen during training [37]. The following sections introduces a couple of commonly used techniques to represent text as vectors.

2.5.1 Bag-of-words

The *Bag-of-words* (BoW) algorithm is used to represent a text document, for example a sequence of words or sentences, as a "bag", i.e. multiset, of its words. The word embedding of the BoW contains the frequency with which each word in the bag occurs in the document, neglecting word order [38].

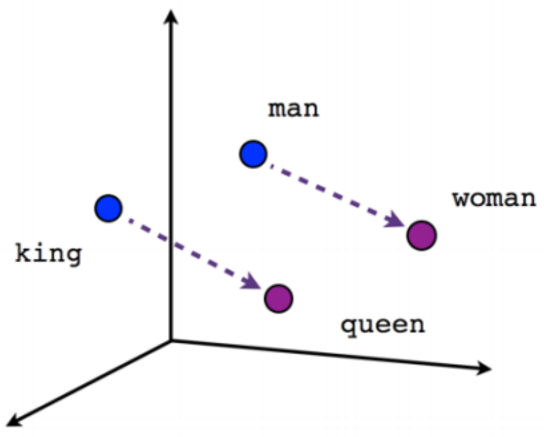


Figure 2.5: An example illustrating the relationship between word embeddings for the words "king" and "queen" [36].

2.5.2 TFIDF

The *term frequency inverse document frequency* (TFIDF) algorithm is used for converting a text document into a vector in some multi-dimensional space. For each term in a document, the algorithm compares its *inverse document frequency* (IDF), i.e. the total number of documents divided by the number of documents where the term occurs, with its *term frequency* (TF), i.e. the frequency of the term in the document [39, 40].

The calculation of the term frequency can vary due to different ways of normalizing the frequency, for example by dividing the term frequency with the number of terms in the intended document. The original, non-normalized, term frequency is defined in formula 2.7 [40].

$$TF(t, d) = \text{Number of occurrences of } t \text{ in } d \tag{2.7}$$

Where:

t is the term in the text document,

d is the text document.

The idea of the inverse document frequency is to assign a lower weight to terms that occur often, while assigning higher weights to less common terms. This gives infrequent words bigger impact on the IDF value in the TFIDF vector, which is beneficial for example when comparing strings [39]. The IDF value is defined in formula 2.8.

$$IDF(t, D) = \frac{N}{|\{d \in D : t \in d\}|} \tag{2.8}$$

Where:

t is the term in the text document,

d is the text document,

D is the corpus of text documents,

N is the total number of text documents in the corpus.

The TFIDF value for a term is calculated by multiplying the term's TF value with its IDF value, as defined in formula 2.9.

$$TFIDF(t, d, D) = TF(t, d) * IDF(t, D) \tag{2.9}$$

t is the term in the text document,

d is the text document,

D is the corpus of text documents.

Applying the TFIDF algorithm to a string results in a vector containing the TFIDF values for all terms in the string. Tokenization, described in section 2.3, is applied to each document to obtain the terms for which to calculate TFIDF values. The length of each TFIDF vector in a corpus is equivalent to the number of terms in the entire corpus [39].

Cosine similarity can be used to compute the similarity between the TFIDF vectors of two strings, as described in section 2.1.2. A higher cosine similarity score indicates that the strings are more similar. Additionally, if the overlapping terms are more rare in the corpus, the score increases even more [41].

2.5.3 SoftTFIDF

Cohen et al. [2] defined the SoftTFIDF algorithm, which is a less strict version of TFIDF [11]. The idea was to make similar terms count as the same. This way, misspellings or variants of words can be considered the same, which would increase the frequencies of those words. SoftTFIDF combines TFIDF with a character-based similarity function, to not only find exact terms, but also lexically similar terms. The character-based function is used with a threshold, which defines how similar the terms must be, to be considered the same [2].

2.5.4 GloVe

Global Vectors (GloVe) is an unsupervised learning algorithm for generating word embeddings, developed by Stanford in 2014 [42]. GloVe is trained on statistics of aggregated global word-to-word co-occurrences from a corpus, meaning that the embeddings will consider how frequent words co-occur with each other in the given corpus [42]. It is often used to find relations between words, such as synonyms. However, it is not effective in identifying homographs, i.e. words with the same spelling but different meanings [43]. GloVe is available as pre-trained models based on a pre-defined corpus of text, which introduces the risk of out-of-vocabulary tokens [44].

2.5.5 BERT

Bidirectional Encoder Representations from Transformers (BERT) is a deep learning model developed by Google in 2018 [45]. It is based on the transformer architecture described in section 2.4.4 and is used in a range of NLP applications. BERT only uses the encoder-part of the transformer architecture, meaning it uses attention and feed-forward layers, but not recurrent connections [45]. It is *bidirectional*, meaning it learns both from left to right and right to left context in all layers. BERT is pre-trained using a combination of *masked language modeling* (MLM) and *next sentence prediction* (NSP) to learn an inner representation of the English language. It is trained on a large collection of unpublished books and Wikipedia [46]. BERT uses WordPiece tokenization, which is described in section 2.3.2.

Compared to traditional unidirectional models, BERT has shown a major improvement for understanding context. This makes it effective in identifying homographs and distinguishes these as multiple vectors even though the spelling is identical [47].

A limitation of BERT is that its structure makes it difficult to derive sentence embeddings, i.e. embeddings on sentence level instead of word level. This can be bypassed by inputting single sentences through BERT and averaging the outputs of the word embeddings to sentence embeddings [48].

2.5.6 SBERT

Sentence-BERT (SBERT) is a modification of the BERT model for generating sentence embeddings. It uses a pre-trained BERT model fine-tuned with Siamese and triplet network architectures and natural language inference (NLI) data. SBERT has proven to out-perform state-of-the-art sentence embeddings on benchmark tasks [48].

2.6 Evaluation Metrics

To compare and evaluate the results of the classification algorithms, the evaluation metric *precision*, *recall*, *F_1-score* and *Matthew's correlation coefficient* (MCC) can be used. The results can also be visualized using a *confusion matrix*. The metrics and the confusion matrix are further described in the following sections.

2.6.1 Confusion Matrix

A confusion matrix, also known as an error matrix [49], is a matrix used to summarize and visualize the performance of a classification algorithm [50].

Each column in the matrix represents the instances in the predicted class while each row represents instances in the actual class. The cells show the overlap of predicted and actual classes, which makes it easy to identify the number of correct and incorrect classifications for each class. The values in the cells are used in different constellations when calculating precision, recall, F_1-score or MCC [51]. An example of a confusion matrix for binary classification can be seen in Figure 2.6.

Figure 2.6: An example of a confusion matrix.

		Predicted Label		
		n	p	Tot
Actual Label	n’	TN	FP	TN+FP
	p’	FN	TP	FN+TP
	Tot	TN+FN	FP+TP	

The cells in the matrix represent:

TP - the number of true positives,

TN - the number of true negatives,

FP - the number of false positives,

FN - the number of false negatives,

n' - the number of actual negatives,

p' - the number of actual positives,

n - the number of predicted negatives,

p - the number of predicted positives.

2.6.2 Precision and Recall

Precision and recall are two evaluation metrics that are often used together since they both measure relevance, but in slightly different ways. Precision is the relation between relevant instances, i.e. the instances that are supposed to be positive and the retrieved instances, i.e. the instances that were predicted positive, and recall is the fraction of relevant instances that were retrieved [52]. Precision is defined in formula 2.10 and recall is defined in formula 2.11 [52].

$$Precision = \frac{TP}{TP + FP} \tag{2.10}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.11}$$

Whether high precision, high recall or a balance is preferred when determining if a result is good or not is context-dependent [53].

2.6.3 F$_1$-score

F$_1$-score is the harmonic mean of precision and recall and is often used as a measure in classification problems. F$_1$-score is defined in formula 2.12 [52].

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{2.12}$$

F$_1$-score is considered the standard indication of a classifiers performance in general, since it is based on equal contribution on both precision and recall. It is widely-used when comparing the quality of a classifier as it is easy to interpret and emphasizes the positive class, which is often of interest [52].

2.6.4 Matthew's Correlation Coefficient

The Matthew's Correlation Coefficient (MCC), in statistics known as the *phi coefficient* [54], is a balanced measure of the quality of binary classification. It takes true and false positives and negatives into account and is useful when evaluating imbalanced data, meaning that the number of instances in the classes are very unequal [55]. It returns a value between -1 and +1, where +1 represents a perfect prediction, 0 an average random prediction and -1 an inverse prediction [55]. MCC is generally regarded as being one of the best measures of describing the confusion matrix [54]. MCC can be computed using formula 2.13 [54].

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.13}$$

Chapter 3
Approach

This chapter describes the methodology used when investigating approaches for POI name matching. The CRISP-DM process used for the thesis work is introduced, followed by presenting the structure of the work according to the process. Additionally, the tools used in the work are presented.

3.1 CRISP-DM

The thesis work was carried out according to the *Cross-Industry Standard Process for Data Mining* (CRISP-DM) process, which is a structured approach commonly used in data science projects [56]. The CRISP-DM process is divided into six correlating phases, as illustrated in Figure 3.1. The phases are described below.

1. Business understanding: Understanding the client's objectives, requirements and expectations from the project.

2. Data understanding: Identifying, collecting and analyzing data to gain an understanding on how to accomplish the project goal.

3. Data preparation: Preparing the data for the modeling phase, involving cleaning and formatting.

4. Modeling: Building and assessing various models, for example algorithms and approaches. This phase is often done iteratively until the best model(s) is determined.

5. Evaluation: Broader evaluation of how well the model(s) meets the project goals.

6. Deployment: Deploying the model(s) for customers to access, including monitoring and maintenance.

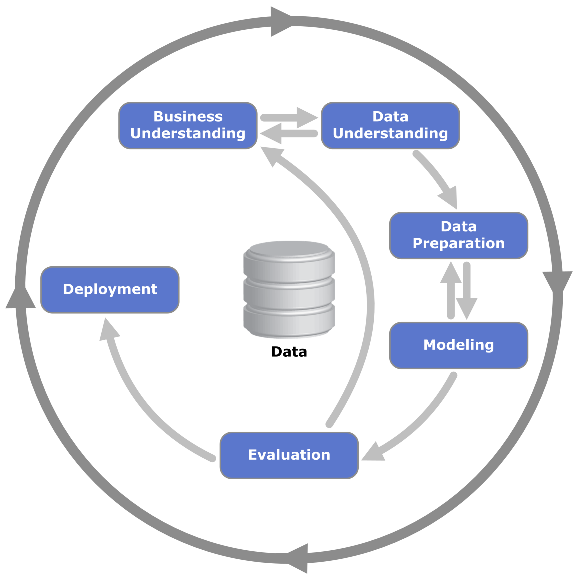


Figure 3.1: The CRISP-DM process, describing the life cycle as six phases with arrows indicating the most important dependencies between phases [57].

The phase of business understanding (1) which corresponds understanding the purpose of the task. This was described in Chapter 1. The phase of deployment (6) has been left out as it is out of the scope of this thesis. The work was carried out only according to phases 2-5.

3.2 Data Understanding

This section covers the phase of data understanding, according to the CRISP-DM process. It describes the collection and analysis of the data sets used in the work.

3.2.1 Data Sets

To address the problem of POI name matching, it was essential to find good data to work with. In order to find POIs that could potentially be the same, we needed to select data from different sources with a sufficient overlap, with respect to geographical areas and POIs. In line with the scope of the thesis, we investigated options of data sets containing English POIs. Several data sets containing geographical data from North America with POI names primarily in English were collected from four sources: OpenStreetMap (OSM) [58], Yelp [59], Data.gov [60] and Open Data Portal [61]. These sources are further introduced in the following sections.

OpenStreetMap

OpenStreetMap (OSM) is a project supported by the OpenStreetMap Foundation. The idea is to make geographical data available through crowdsourcing [58]. This leads to a huge number of contributions and the quality of the data can vary a lot, leading to problems such as incorrect or incomplete data. Data for different regions are available from their download web site [62]. For this thesis data from the regions Florida, North Carolina, Massachusetts and British Columbia was used.

Yelp

Yelp provides a platform with local business information together with reviews and other services. For this thesis we used the "Yelp Academic Business" data set [63] covering North America. The idea is to make it easy for customers to get information and get in contact with businesses. Just as for OSM, the data is crowdsourced [59].

Data.gov

Data.gov is run by the U.S. government and provides data sets with federal, state and local government information [60]. For the work in this thesis, we have used the data set "Community Points of Interest" [64], which consists of information about community centers, fire, police and EMS stations, hospitals, libraries, schools and post offices in Cary, North Carolina [65].

City of Vancouver Open Data Portal

The City of Vancouver provides public data, such as tables, maps and charts, about the city through Open Data Portal [61]. The three data sets called "Cultural Spaces" [66], "Schools" [67] and "Libraries" [68] have been used in the thesis.

3.2.2 Data Collection

The data sets collected for the thesis contain a huge amount of data, some covering all of North America and some only certain regions. To achieve a sufficient overlap between the data sets, we limited the data to specific parts of North America. When selecting the specific areas, we strived for a balance between different types of POIs and between urban and rural areas. Unfortunately, for rural areas, we found little overlap in data. Since the rural areas are not as dense and do not contain as many businesses as urban areas, the data sets do not contain much data in these areas. Therefore, urban areas were chosen of which two were central city areas and two were suburbs. We decided to collect data from several different areas in North America to ensure overlap and variety in POIs. The geographical areas used in this work are specified in Table 3.1.

Table 3.1: The geographical areas selected used in the thesis.

Region	Coordinates (Lat, Lon)	Geographical Setting
Boston	42.366625 to 42.349203, -71.068092 to -71.055022	City Central
Vancouver	49.258694 to 49.204954, -123.128227 to -123.025537	City Central
Orlando	28.616707 to 28.567949, -81.405150 to -81.315333	Suburban
Cary	35.859603 to 35.655800, -78.902602 to -78.701696	Suburban

3.2.3 Data Analysis

To make the data from the different data sets compatible for the work, we investigated their structure to understand how they could be used. When looking at names of POIs to determine whether they are the same entity, we assumed that they were located relatively close to one another in order to be considered a match. Therefore, the minimum requirement for a data point to be considered a useful POI is to have a name and a position, in terms of latitude and longitude coordinates. The coordinates are required for reducing potential matches to POIs only located within a certain distance from each other.

3.3 Data Preparation

This section covers the phase of data preparation according to the CRISP-DM process, including cleaning and formatting of the collected data. It also describes how we structured and labeled pairs from the collected data sets in order to evaluate the classification of our approaches.

3.3.1 Data Cleaning and Formatting

Clean and valid data is required in order to evaluate the various approaches used for POI name matching. To obtain valid data to be used on testing, we have manually labeled data from the areas in North America specified in Table 3.1.

Due to the slightly different characteristics of the data sets the data needed to be reformatted before merging. The goal was to create a data set of POI pairs representing potential matches of POIs. The pairs consisted of one POI from the OSM data set, since this had the most data points and would therefore generate the largest overlap, and one POI from one of the other data sets.

The data from OSM contains three types of components: nodes, ways and relations. Only node components were considered as these were the only components containing coordinates for one position. From these, only nodes containing a name tag were kept. Since Yelp contains data of businesses, all containing coordinates and names, all data points were considered to be POIs and therefore kept. All data in the Data.gov data set as well as in the data sets from Open Data Portal are POIs with coordinates and names and therefore all of these were kept.

The most essential attributes from the data sets were the name of the POI and its coordinates, latitude and longitude. Other attributes such as timestamp, version, review count etc. were dropped since these were not necessary for the task. Other information, such as category, could possibly be useful for determining if two POIs are the same. As mentioned earlier, the use of crowdsourced data introduces the risk of incorrections and incompletions, meaning that it would require identifying the correct and matching category for each POI. While it might be beneficial for the task, it would make it more complex and is out of the scope of this project.

Based on the assumption that matching POIs are likely to be closely located to each other, we only consider potential matches to be POIs within a certain distance. When using crowdsourced data, we cannot ensure that the coordinates are identical for the POIs between data sets, however, we chose a distance to allow for some incorrections of position. The selection of distance was based on wanting to identify as many matching POIs as possible, while keeping the number of non-matching POIs relatively low, to avoid having a too imbalanced data set. After some exploration we concluded that 0.0002 degrees in both latitude and longitude was suitable for the task. This corresponds to approximately 25 meters. With a larger distance than 0.0002 degrees we did not observe many more matches, while the number of non-matching pairs increased significantly, making the data set more imbalanced. On the other hand, when using a distance smaller than 0.0002 we observed that too many matching POIs were filtered out.

We filtered the data sets to contain only the information we were interested in for each pair of POIs: the name, longitude and latitude of the POI from OSM, the name, longitude and latitude of the POI from the other data set.

3.3.2 Labeling

To evaluate the approaches on the data, we first needed to define whether or not the POIs in a pair refer to the same entity. This was done by labeling the pairs as matches or non-matches. The POIs were paired and labeled by implementing a basic labeling script. The script first created pairs of POIs by calculating the distance between POIs from different data sets. The pairs with a distance less than 0.0002 degrees were kept. The pairs were then manually labeled based on the information provided in the names. The following labels were used:

- 0 - no match
- 1 - match
- 2 - difficult to determine
- 3 - not a POI

The result of a labeled pair of POI names consisted of their names in the different sources, their coordinates, distance from each other and the label. An example of a labeled pair is as follows:

```
["Starbucks", "Haymarket", 42.3573, -71.0582, 42.3572, -71.0582, 7.8187, 0]
```

After labeling the data, all pairs with label 3 were removed, since they do not address the problem of POI name matching. Since the pairs with label 2 could not be determined as a match or not, even by a human, there was no way to evaluate whether or not an algorithm could classify it correctly. Therefore, the 46 pairs with label 2 were removed from the data set.

The remaining pairs were used for the experiments in this thesis. In total, the used data set consisted of 5,567 pairs. Out of these, 4,721 were non-matches and 846 were matches of which 504 were exact matches. An exact match is where the names of the POIs are lexically identical. Table 3.2 gives an overview of the labeled data used in the work. Figure 3.2 illustrate the distribution of matches and non-matches in the data set.

Table 3.2: Overview of the labeled data used in the thesis.

Region	Data Sets	Number of Pairs	Number of Matches	Number of Exact Matches
Boston	OSM, Yelp	2,478	274	166
Vancouver	OSM, Yelp	2,718	491	295
Vancouver	OSM, Open Data Portal	30	4	1
Orlando	OSM, Yelp	332	71	39
Cary	OSM, Data.gov	9	6	3
Total:		5,567	846	504

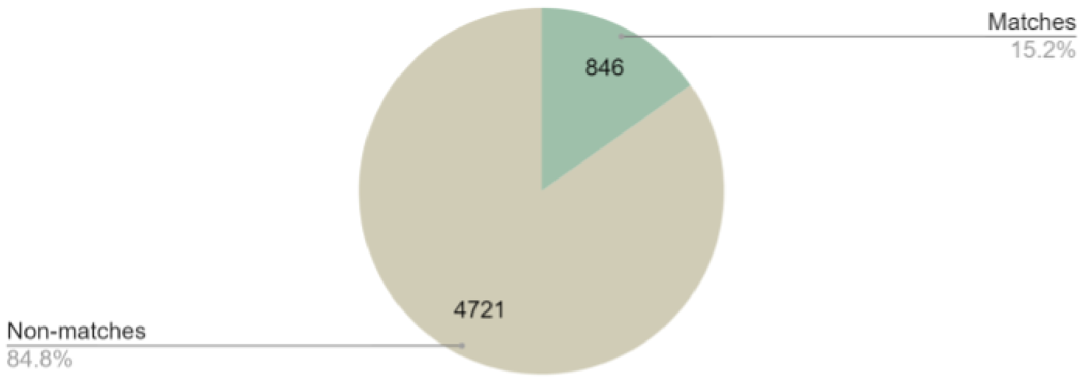


Figure 3.2: The distribution of matches and non-matches in the labeled data set.

3.4 Modeling

This section covers the modeling phase in the CRISP-DM process. The phase consisted of iteratively implementing and assessing algorithms and approaches used to investigate the research questions of the thesis. The phase was divided into four major parts; the first aiming to establish a baseline, the second aiming to investigate hybrid similarity functions, the third aiming to investigate improvements using semantic similarity and the fourth aiming to investigate machine learning using similarity functions. Since the modeling phase is comprehensive, the implementations in each of the four parts are presented and described in Chapter 4.

3.5 Evaluation

This section covers the evaluation phase in the CRISP-DM process and describes how the performance of the approaches in the modeling phase have been evaluated.

To determine how well our approaches perform for matching POIs, we evaluated all of our approaches, except the last part using machine learning, with two variants of the labeled data set, as presented in section 3.3: one containing the entire data set and one containing only the pairs with non-exact names, which we refer to as the reduced data set. The purpose of testing the approaches on the first data set was to see how the approaches perform on data that represents reality. The purpose of using the second data set was to see how the approaches perform specifically on non-trivial tasks, which is where the state-of-the-art algorithms often fail. We made no evaluation of the machine learning approaches on the second data set. The reason for this was to maximize the number of data points when training the models.

In order to compare the approaches, we first needed to define how to measure good performance for the task. Since the goal of the approaches is to classify the data set as similarly as possible to what we as humans would, we compared the predictions to the labels in the data set. The quality of the predictions was evaluated using the metrics precision, recall, F_1-score and MCC. Since F_1-score is regarded as the standard for evaluating classifiers, we primarily used this to compare the results. Furthermore, confusion matrices were used to obtain additional information about the performance. Above this, the results were evaluated by manually analyzing the classified pairs, to identify strengths and weaknesses of the approaches and mitigate these. Along with the results we show examples of how the approaches managed to classify pairs of POIs.

To answer the research questions regarding how the approaches improve performance of the task, we compared the results to state-of-the-art algorithms. To do this, we established a baseline to which we compared the approaches using the above-mentioned metrics. We also measured the error reduction rate compared to the baseline. This measure shows the amount of errors that was not managed by the baseline, but managed by the algorithms. The error reduction rate was investigated for the best performing approach in each phase of the work.

Several algorithms used in this work return a similarity score instead of a direct classification. All scores were normalized to values between 0 and 1, where 0 is completely different and 1 is completely similar. In order to classify the pairs, a threshold is needed to convert the score to a prediction. Depending on the thresholds selected, the predictions can differ which makes the selection of thresholds difficult. We used a simple grid search to find the optimal thresholds for the different algorithms. Since all scores were normalized to values 0 and 1, we used an interval of 0.05 between the thresholds in the grid search for all values between 0 and 1.

3.6 Tools

This section introduces the software tools used throughout the thesis and describes in which parts they were implemented.

3.6.1 NLTK

Natural Language Toolkit (NLTK) [69] is an open-source platform providing several libraries for NLP-related tasks such as tokenization, classification and stemming.

In this work, we use the *nltk.metrics.distance* module for computing the Jaro, Jaro-Winkler, Levenshtein and Jaccard similarity functions. The module *nltk.stem.snowball* is used for stemming of words when text pre-processing. We also use the *nltk.corpus* package, from where we use a collection of English stop words used for stop word removal when text pre-processing.

3.6.2 Unidecode

Unidecode [70] is a Python module for transforming text strings to ASCII representations. The module is used for text pre-processing in the work.

3.6.3 Scikit-learn

Scikit-learn [71] is a widely used machine learning library for Python, that provides various tools for machine learning and statistical modeling.

Scikit-learn's *sklearn.metrics.pairwise* submodule is used for calculating cosine similarity between word embeddings. The classifiers *RandomForestClassifier* and *MLPClassifier* from the library are used when evaluating supervised machine learning approaches. Additionally, Scikit-learn is used for calculating all evaluation metrics and for applying cross-validation in the machine learning approaches.

3.6.4 BPEmb

BPEmb [20] is a library containing pre-trained subword embeddings in several different languages. The embeddings in the library are GloVe implementations trained on Wikipedia articles using BPE tokenization [20]. The BPEmb library contains word embeddings trained with different vocabulary sizes and embedding dimensions.

We use BPEmb's pre-trained embedding for English to embed POI names and tokens in the hybrid algorithms. From now on, we refer to the English embedding from BPEmb as BPEmb.

3.6.5 Hugging Face

Hugging Face [72] provides Python-based open-source libraries for transformer architectures, mainly focusing on NLP tasks. It contains a wide range of tools and models based on widely used architectures, such as BERT and SBERT.

From Hugging Face's Transformers library, we use the pre-trained BERT model *bert-base-uncased* [73] and from the Sentence Transformers library, we use the pre-trained SBERT model *all-mpnet-base-v2* [74]. The models are used to embed POI names.

3.6.6 SHAP

SHAP [75] is a Python library providing tools for interpreting the predictions of machine learning models using Shapley values.

SHAP is used in the machine learning approaches to compute and interpret the feature importance for the predictions.

3.6.7 XGBoost

XGBoost [76] is an open-source library for machine learning algorithms using gradient boosting techniques.

The *XGBClassifier* is an optimized implementation of scikit-learn's gradient boosted tree classification, which we use when evaluating supervised machine learning approaches.

Chapter 4
Implementation

This chapter covers the modeling phase in the CRISP-DM process and describes the implementation of the approaches investigated in the thesis. Implementations and evaluations have been done iteratively to thoroughly address the research questions. The following sections represent the four major parts of the modeling phase: establishment of the baseline, investigation of hybrid similarity functions, investigation of improvements using semantic similarity and investigation of machine learning approaches using similarity functions. The three latter directly corresponds to each of the research questions. Furthermore, the implementations in the different parts are closely related, since the evaluations can affect decisions made in the following implementations.

4.1 Baseline

To be able to evaluate if the performance of the approaches investigated in the thesis improved compared to state-of-the-art similarity functions, a baseline was established. Similarity functions have been used in previous work to solve various name matching tasks. However, no single algorithm has proved to perform the best for all task domains. When defining the baseline, we needed to evaluate which state-of-the-art similarity function were suitable for POI name matching. The similarity functions were evaluated on the data sets described in section 3.3.2 so that we could select the best performing similarity functions as the baseline.

The character-based similarity functions evaluated were Levenshtein similarity, Jaro similarity and Jaro-Winkler similarity. All of these were implemented using the *nltk.metrics.distance* module from NLTK. No text pre-processing was done before applying the algorithms.

The token-based similarity functions evaluated were Jaccard similarity and cosine similarity. Jaccard similarity was implemented using the *nltk.metrics.distance* module from NLTK. The cosine similarity was implemented by creating a BoW vector for each POI name in the pairs and then calculating the cosine similarity between the vectors using the *sklearn.metrics.pairwise* submodule from scikit-learn. For the token-based similarity functions the POI names were tokenized using whitespace tokenization and no pre-processing was done.

As will be presented further in the result section 5.1, the evaluation of the five state-of-the-art algorithms showed that Jaro-Winkler performed best among character-based similarity functions and cosine similarity performed the best among the token-based similarity functions. Therefore these two were established as the baseline. As these proved to be the best performing similarity functions these were used in several of the following implementations.

4.2 Hybrid Similarity Functions

This section describes the implementations for investigating how hybrid approaches can improve performance for POI name matching, corresponding to research question 1. An important aspect when investigating hybrid similarity functions is to understand the similarity functions can be combined to compensate for each other's weaknesses, which has been suggestion in previous work.

The implementations of hybrid approaches described in this section are SoftTFIDF and RestrictedSoftTFIDF. They are both based on the TFIDF algorithm and therefore TFIDF was implemented as a first step. Even though TFIDF is a non-hybrid approach, it was evaluated for comparison with SoftTFIDF and RestrictedSoftTFIDF.

The implementation of text pre-processing as well as each approach is described further in the following sections. All experiments in these sections were performed on both variants of data set; the entire data set and the reduced data set. Each variant was also performed with and without text pre-processing, to be able to determine if this affected the performance.

4.2.1 Text Pre-processing

For the baseline the POI names were tokenized using whitespace tokenization, but no text pre-processing was done. Suffixes, variants of words, different usage of special characters and other noise are examples of difficulties that lexical similarity functions do not perform well on. This issue was addressed by using pre-processing together with whitespace tokenization, before calculating any similarity.

The following techniques were used in the text pre-processing, in the given order:

1. Lower-casing.
2. Noise removal. This was done by transforming all special characters into ASCII representation, using the Unidecode Python module, and removing the remaining characters that were not letters or numbers.
3. Whitespace tokenization.
4. Stop word removal. This was done using a collection of English stop words from the NLTK library. All stop words with a length less than or equal to three was removed from the tokenized names. The same noise removal as described above, was done on the stop word collection before applying it to the tokenized POI names.

5. Stemming. This was done using NLTK's *nltk.stem.snowball* module for English.

The text pre-processing in the thesis from now on refers to the process consisting of these steps. As an example, applying the pre-processing to the name "17°C Bubble Tea & Dessert Café" will result in the token set {"17degc", "bubble", "tea", "dessert", "cafe"}.

4.2.2 TFIDF

From manually analyzing the baseline evaluation we identified a pattern, that common business-related words, such as "café", "restaurant" and "shop" were being used inconsistently among the POI names. We found that this made it difficult for the lexical similarity functions. In many of the names business-related words were places in the end of the name, while in others they were not included at all. For example, "Anthem" and "Anthem Kitchen & Bar" are names of the same POI, the latter with the additional business-related words "Kitchen" and "Bar". Both Jaro-Winkler and cosine similarity achieved low scores for these types of pairs. To address this problem, we investigated how assigning weights to certain tokens could improve the performance. For example, assigning a larger weight to the word "Anthem" could make it more important than "Kitchen" and "Bar", leading to higher similarity between the names. Since the TFIDF algorithm is used to assign weights to tokens that occur frequently in a given corpus, we investigated how TFIDF could be used for handling common business-related words in the POI names.

We implemented the TFIDF algorithm described in section 2.5.2. The corpus used in the algorithm was created by extracting all tokens from all POI names in the given data set. The vectors created from the TFIDF values are dependent on the corpus and will have low values for tokens occurring frequently. The similarity between the POI names was calculated by using cosine similarity between the TFIDF vectors.

4.2.3 SoftTFIDF

From analyzing the performance of the TFIDF implementation, we found that the algorithm could not handle variants of similar parts of the names. Instead, it considered these parts as completely different tokens. To solve this a variant of the SoftTFIDF algorithm inspired by Cohen et al. [2] was implemented. The main idea of the Cohen et al.'s algorithm was to use a character-based function to determine if two similar tokens can be considered the same, when calculating TFIDF values in the TFIDF algorithm. This could make the algorithm more robust for variants of words while still weighting words by occurrence.

Our implementation of the SoftTFIDF algorithm is a hybrid between TFIDF with cosine similarity and a character-based similarity function. Similar to Cohen et al.'s implementation, we chose Jaro-Winkler as the character-based similarity function, since the baseline indicated that it performs well on POI name matching. While Cohen used a set threshold of 0.90 for the Jaro-Winkler similarity score, we evaluated our implementation using several thresholds.

The corpus used in SoftTFIDF is created the same way as for TFIDF. For each POI pair, each token from the first name is compared with each token from the second name. The Jaro-Winkler similarity score is calculated for each token combination. After considering all combinations for a token, only the token combination with the highest Jaro-Winkler similarity score, that is also above the defined similarity threshold, is stored in a similarity map as a one-to-one relation. The tokens in the similarity map are considered the same when calculating frequencies for the TFIDF values. The frequency for a token in the similarity map is the sum of the token's own frequency and the similar token's frequency. This resulted in a lower TFIDF value for the token in the SoftTFIDF vector. Finally, the similarity between the POI names was calculated using cosine similarity between their SoftTFIDF vectors.

4.2.4 RestrictedSoftTFIDF

From manually analyzing the performance of the TFIDF and SoftTFIDF, we could see an improvement for names where the impact of common business-related words was low. Words such as "restaurant" received a lower similarity score and was therefore considered less important. For example, the algorithm was able to correctly determine "Mooo..." and "Mooo Restaurant" as a match. A problem that was identified was that the algorithms were not able to determine what parts of the name to consider important for other cases than business-related words. Deciding what words that are important in a name can be difficult since it is context-dependent, meaning it can be important in some cases but not in others. We could especially identify this pattern for geographical names, such as names of cities or neighborhoods. An example of this is the POIs "Cary" and "Cary Train Station", both located in the Cary area. Both the SoftTFIDF and the TFIDF incorrectly classified these as matching, see the results in Table 5.15, since the tokens "train" and "station" are more common in the corpus and therefore are considered less important. In this context the geographical area "Cary" should not be considered more important than the rest of the tokens.

Based on this observation we restricted the corpus by the geographical area. The idea was that tokens with geographical meaning occurs more often in names of POIs located in the area, and therefore would have less impact on POIs in that area. For example, the token "Cary" would be less important when occurring in POIs names in the Cary area. We decided to create the corpus with all the POIs that are less or equal to 0.0002 degrees, which is approximately 25 meters, from one of the POIs in the pair we are considering. With this technique, the corpus was created from a significantly less amount of POIs compared to the corpus in TFIDF and SoftTFIDF, which changes the token weights significantly.

4.3 Semantic Similarity Approaches

This section describes the implementations for investigating how semantic similarity can be used to improve performance of hybrid approaches for POI name matching, corresponding to research question 2.

A disadvantage with the lexical similarity functions is their lack of ability to capture the meaning of the words. For example, when comparing the POI names "Pine House Bread & Cake shop" and "Pine House Bakery", the last part of the names, "Bread & Cake Shop" and "Bakery" are semantically similar. Both character-based and token-based similarity functions, and therefore also SoftTFIDF, would achieve a low similarity score for the example. The aim of this part of the thesis is to identify and compare the semantic similarity between the POIs in order to add an aspect to the previous presented hybrid approaches.

As described in section 2.5, there are several learned word embeddings aiming to capture the semantic similarity between pieces of text. To begin with, we investigated how pre-trained word embeddings used with cosine similarity performed for POI name matching. Then we integrated the word embeddings into the SoftTFIDF implementation as this performed the best in earlier evaluations, presented in the Results section 5.2.

The implementation of the approaches using semantic similarity is described further in the following sections. All experiments in this section were performed on both variants of data sets; the entire data set and the reduced data set.

4.3.1 Pre-trained Word Embeddings

We investigated how well pre-trained word embeddings could capture semantic similarity between the POI names, since the structure of names can differ from more standard sentences or short texts. Training a word embedding requires large amounts of data, more than was feasible to collected for the work. To evaluate the performance, we used three pre-trained word embeddings; BPEmb, BERT and SBERT. The word embeddings create vector representations of the names by considering the words both lexically and semantically. To calculate the similarity between the embedded vectors, cosine similarity was used.

The names of the POIs were input as raw data meaning no pre-processing or tokenization was used, since the embeddings contain their own tokenization. For BERT the *BertTokenizer* from Hugging Face was used. SBERT and BPEmb need no additional tokenization since it is a part of the actual models.

BPEmb can be used with different vocabulary sizes and embedding dimensions. To find the most suitable hyperparameters for the task, we compared the results when using various hyperparameters. The comparison was done using only English models. We observed that the vocabulary size of 50.000 and embedding dimension of 300 had the highest performance and therefore these were used in the following implementations. The BERT and SBERT models used were provided from Hugging Face.

4.3.2 SemanticSoftTFIDF

When comparing the results between the SoftTFIDF algorithm and the pre-trained word embeddings with cosine similarity, we observed that they each failed on different types of pairs. The word embeddings seemed to better handle POI names specifically containing words with similar semantic meaning, while not performing as well in general for other cases. In contrast, the SoftTFIDF performed well in general, however did not manage to recognize semantic similar words. For example, the non-matching pair "Cary" and "Cary Train Station" was incorrectly classified as a match by SoftTFIDF, but correctly classified as a non-match by SBERT and BPEmb, see results in Table 5.14 and Table 5.15.

To mitigate this issue we investigated a hybrid that could both identify these semantic similarities while still not failing in the general cases, by combining hybrid similarity functions with semantics. We implemented a hybrid consisting of SoftTFIDF and various pre-trained word embeddings, which we refer to as SemanticSoftTFIDF. The idea of the hybrid was that the pre-trained word embedding would improve SoftTFIDF by identifying semantically similar tokens.

The SemanticSoftTFIDF algorithm is an extension of SoftTFIDF. While SoftTFIDF only calculates the Jaro-Winkler similarity between tokens, SemanticSoftTFIDF calculates both the Jaro-Winkler similarity as well as the cosine similarity between the tokens' word embeddings to determine which tokens are similar enough to add to the similarity map. This allows for tokens being considered similar either by lexical similarity, for tokens with high Jaro-Winkler similarity scores, or by semantic similarity, for tokens with high cosine similarity score between their word embeddings. For the implementation of SemanticSoftTFIDF, we evaluated the algorithm using each of the pre-trained word embeddings in combination with the SoftTFIDF. The pre-trained word embeddings implemented are BPEmb, BERT and SBERT, used together with cosine similarity, as described in section 4.3.1.

Pre-trained word embeddings are meant to be used on entire text strings. However, the nature of the TFIDF algorithm requires tokenization. Since earlier results for the SoftTFIDF algorithm, see section 5.2.2, indicated better results using pre-processing, this was also done for SemanticSoftTFIDF. This means that SemanticSoftTFIDF measures the semantic similarity between tokens instead of the entire POI names.

4.4 Machine Learning Approaches

This sections describes the implementations for investigating how machine learning can be used to improve performance of hybrid approaches for POI name matching, corresponding to research question 3. The implementations were inspired by Santos et al. [4] and used scores from multiple similarity functions as input to machine learning models. The purpose was to combine different hybrid similarity functions to benefit from their different types of advantages. It also allowed us to avoid manual tuning of thresholds.

For example, the non-matching pair "Bao Bakery" and "Bamboo Café", is more likely to be classified correctly by algorithms considering lexical similarity but not semantic similarity, since the words "Bakery" and "Café" have a high semantic similarity but low lexical similarity. In contrast, algorithms considering semantic similarity but not lexical similarity are more likely to incorrectly consider these as a match. The idea of using various similarity functions as input to the models was to address issues where different algorithms considered different aspects of similarity, by classifying the pairs based on several similarity functions instead of only a single similarity function.

The investigation of machine learning approaches consisted of two parts. First we investigated machine learning approaches using only similarity functions. Then we investigated how the models were affected when using additional features that were more closely related to the raw data of the POIs. All of the similarity functions used in the machine learning approaches are evaluated and explained in detail previously in the work.

The machine learning models were evaluated by splitting the entire data set into training (80%) and test (20%) data. The split was seeded to make the results from the models comparable with each other. To select the hyperparameters for the models, stratified 5-fold cross-validation was used on the training data. The models were then trained on all training data, using the most suitable hyperparameters for each model, and evaluated on the test data. To make the machine learning results comparable to the earlier algorithms, we also trained and evaluated the approaches according to Santos et al.'s k-fold cross-validation methodology. We performed a stratified 5-fold cross-validation on the entire data set to get an average of each evaluation metric by letting all pairs be used as test data at some point in the folds. This method was performed for best performing variants when evaluated on the test set.

Additionally, we analyzed the feature importance for these to evaluate the level of importance of the similarity functions in the variants for the models' predictions. This was done by calculating and plotting the SHAP values of the models.

We evaluated three machine learning models: random forest, gradient boosted trees and a multilayer perceptron. We used scikit-learn's implementation of random forest and XGBoost's implementation of gradient boosted trees. The models have 300 and 100 trees respectively. For the random forest classifier, entropy was chosen as the splitting criterion. Additionally, we used scikit-learn's implementation of a MLP-classifier for a neural network. The neural network used in the experiments consisted of four layers with 100, 50, 30 and 20 neurons each.

Unlike the other approaches investigated in the thesis, the machine learning approaches were only evaluated using the entire data set and not the reduced data set, to maximize the data for training the models and to have data that is a good representation of reality.

4.4.1 Machine Learning Using Similarity Functions

For the machine learning approaches only using similarity functions as input we created several variants of feature vectors consisting of different combinations of similarity functions evaluated earlier in this work. The selection of variants aimed to include similarity functions with different strengths that were observed in earlier experiments. The evaluated variants of similarity functions used as feature inputs are presented in Table 4.1.

Variant 1 was selected to see how well the models would perform on the same information as the baseline. Variants 2 and 3 were selected to have input from the baseline, a word embedding and a hybrid algorithm. For these variants BPEmb was selected as this performed best in earlier evaluations. SoftTFIDF and SemanticSoftTFIDF were selected as these were the two hybrids performing best in earlier evaluations, with the latter considering semantics. Variant 4 consisted of all similarity functions investigated in the work, to find out if the models would benefit from having as much information as possible even if some of them, for example the state-of-the-art similarity functions, showed poor performance on the task. Variant 5 was selected to see if the models could benefit from only using one variant of the SemanticSoftTFIDF as these showed very similar performance.

The scores were computed for all the pairs in the training data. For SoftTFIDF, RestrictedSoftTFIDF and SemanticSoftTFIDF, the thresholds for the inner Jaro-Winkler similarity and the cosine similarity of the word embeddings needed to be defined before calculating the similarity scores input to the models. Therefore we used the thresholds that achieved the highest performance in the earlier evaluations. The corpora used in TFIDF, SoftTFIDF and SemanticSoftTFIDF were calculated using all data points to make the weighting scheme as representative as possible.

Table 4.1: Feature variants 1-5 containing the feature inputs as indicated with "X".

Feature input	**1**	**2**	**3**	**4**	**5**
Cosine	X	X	X	X	X
Jaro-Winkler	X	X	X	X	X
Levenshtein				X	X
Jaccard				X	X
Jaro				X	X
TFIDF				X	X
SoftTFIDF (Jaro-Winkler threshold: 0.85)			X	X	X
RestrictedSoftTFIDF (Jaro-Winkler threshold: 0.90)				X	X
SBERT				X	X
BERT				X	X
BPEmb		X	X	X	X
SemanticSoftTFIDF with SBERT (Jaro-Winkler threshold: 0.85, embedding threshold: 0.75)		X		X	X
SemanticSoftTFIDF with BERT (Jaro-Winkler threshold: 0.85, embedding threshold: 0.95)				X	
SemanticSoftTFIDF with BPEmb (Jaro-Winkler threshold: 0.85, embedding threshold: 0.70)				X	

4.4.2 Machine Learning With Additional Features

For the second machine learning approach we evaluated how information closely coupled to the raw data of the POI names could have an impact on the performance in machine learning models, in combination with the similarity functions.

The additional features evaluated were the distance in meters between the POIs in each pair, as well as the ratio between the number of tokens in the POI names in each pair. For both of the additional features selected, the idea was to use information closely related to the raw data of the POIs' names and coordinates. This information is different to what is considered in the similarity functions, but could still be relevant. It could be useful for a human to make a decision and for that reason the model could also benefit from it. For example, a matching pair would ideally be more likely to have a shorter distance than a non-match. The two additional features were added separately to the best performing variant from the previous evaluation, which proved to be variant 5, as presented in section 5.4.1. The evaluated variants of similarity functions used as feature inputs are presented in Table 4.2.

Table 4.2: Feature variants 6-7 containing the feature inputs as indicated with "X".

Feature input	**6**	**7**
Cosine	X	X
Jaro-Winkler	X	X
Levenshtein	X	X
Jaccard	X	X
Jaro	X	X
TFIDF	X	X
SoftTFIDF (Jaro-Winkler threshold: 0.85)	X	X
RestrictedSoftTFIDF (Jaro-Winkler threshold: 0.90)	X	X
SBERT	X	X
BERT	X	X
BPEmb	X	X
SemanticSoftTFIDF with SBERT (Jaro-Winkler threshold: 0.85, embedding threshold: 0.75)	X	X
SemanticSoftTFIDF with BERT (Jaro-Winkler threshold: 0.85, embedding threshold: 0.95)	X	X
SemanticSoftTFIDF with BPEmb (Jaro-Winkler threshold: 0.85, embedding threshold: 0.70)	X	X
Distance	X	
Token-length ratio		X

Chapter 5

Results

This chapter presents the results of the implemented approaches according to the evaluation described in section 3.5. While all approaches have been evaluated using different thresholds according to the grid search method, only the results for the threshold(s) resulting in the highest F_1-score is presented for each approach, unless stated otherwise.

5.1 Baseline

This section presents the results used when establishing the baseline. Table 5.1 shows the evaluation metrics for each of the similarity functions investigated when establishing the baseline, for the entire data set. Jaro-Winkler similarity with a threshold of 0.80 achieved the highest F_1-score, 0.957, among the character-based similarity functions, but also among all similarity functions. Cosine similarity with a threshold of 0.40 achieved the highest F_1-score, 0.930, among the token-based similarity functions. Table 5.2 shows the evaluation metrics for each similarity function performed on the reduced data set. Jaro-Winkler with the threshold 0.80 and cosine with the threshold 0.40 achieved the highest F_1-scores here as well with 0.893 and 0.827, respectively. Therefore, Jaro-Winkler and cosine were established as the baseline. The F_1-scores for Jaro-Winkler and cosine for all thresholds in the grid search, on the entire data set, is presented in Figure 5.1. Corresponding F_1-scores for the reduced data set are presented in Figure 5.3. Figure 5.2 shows the confusion matrices for the baseline, using the thresholds resulting in the highest F_1-scores respectively, for the entire data set. Corresponding confusion matrices for the reduced data set are shown in Figure 5.4.

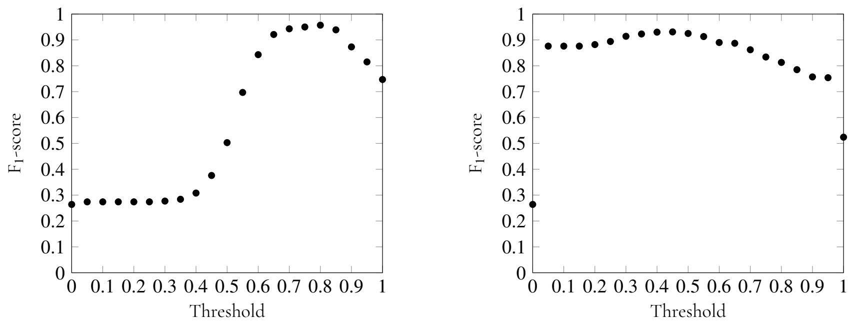
Table 5.1: Evaluation metrics for the investigated similarity functions for establishing a baseline using the entire data set.

(a) Character-based similarity functions.

Similarity Function	Threshold	Precision	Recall	F_1	MCC
Levenshtein	0.45	0.926	0.863	0.894	0.876
Jaro	0.70	0.950	**0.957**	0.954	0.945
Jaro-Winkler	0.80	**0.964**	0.950	**0.957**	**0.950**

(b) Token-based similarity functions.

Similarity Function	Threshold	Precision	Recall	F_1	MCC
Jaccard	0.25	0.930	0.927	0.928	0.916
Cosine	0.40	**0.931**	**0.929**	**0.930**	**0.918**



(a) Jaro-Winkler similarity. **(b)** Cosine similarity.

Figure 5.1: F_1-scores for the baseline similarity functions for all investigated thresholds using the entire data set.

(a) Jaro-Winkler similarity.

		Predicted Label n	Predicted Label p	Tot
Actual Label	n’	4691	30	**4721**
	p’	42	804	**846**
	Tot	**4733**	**834**	

(b) Cosine similarity.

		Predicted Label n	Predicted Label p	Tot
Actual Label	n’	4663	58	**4721**
	p’	60	786	**846**
	Tot	**4723**	**844**	

Figure 5.2: Confusion matrices for the baseline similarity functions using the entire data set.

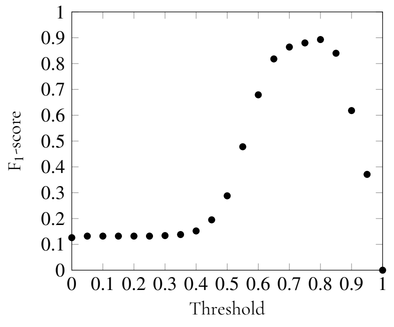
Table 5.2: Evaluation metrics for the investigated similarity functions using the reduced data set.

(a) Character-based similarity functions.

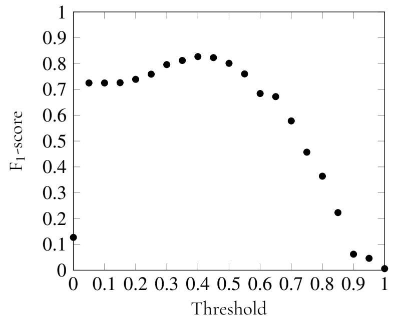
Similarity Function	Threshold	Precision	Recall	F_1	MCC
Levenshtein	0.35	0.683	0.781	0.728	0.709
Jaro	0.70	0.877	**0.895**	0.886	0.877
Jaro-Winkler	0.80	**0.909**	0.877	**0.893**	**0.885**

(b) Token-based similarity functions.

Similarity Function	Threshold	Precision	Recall	F_1	MCC
Jaccard	0.25	0.826	0.819	0.822	0.806
Cosine	0.40	**0.829**	**0.825**	**0.827**	**0.814**



(a) Jaro-Winkler similarity.



(b) Cosine similarity.

Figure 5.3: F_1-scores for the baseline similarity functions for all investigated thresholds using the reduced data set.

		Predicted Label n	Predicted Label p	Tot
Actual Label	n'	4691	30	**4721**
	p'	42	300	**342**
	Tot	**4733**	**330**	

(a) Jaro-Winkler similarity.

		Predicted Label n	Predicted Label p	Tot
Actual Label	n'	4663	58	**4721**
	p'	60	282	**342**
	Tot	**4723**	**340**	

(b) Cosine similarity.

Figure 5.4: Confusion matrices for the baseline similarity functions using the reduced data set.

5.2 Hybrid Similarity Functions

This section presents the results used when evaluating how hybrid approaches can improve the performance in POI name matching, for which the implementations are described in section 4.2.

5.2.1 TFIDF

Table 5.3 shows the evaluation metrics for the TFIDF implementation applied on both variants of the data set. The highest score was achieved using text pre-processing, resulting in an F_1-score of 0.968 with the threshold 0.50 for the entire data set and 0.924 with the threshold 0.45 for the reduced data set. Figure 5.5 shows the confusion matrices for the TFIDF on both variants of the data set, using pre-processing.

Table 5.3: Evaluation metrics for TFIDF with and without text pre-processing.

(a) The entire data set.

Text Pre-processing	Cosine Threshold	Precision	Recall	F_1	MCC
Without	0.40	0.966	0.915	0.940	0.930
With	0.50	**0.975**	**0.962**	**0.968**	**0.963**

(b) The reduced data set.

Text Pre-processing	Cosine Threshold	Precision	Recall	F_1	MCC
Without	0.35	0.891	0.810	0.848	0.839
With	0.45	**0.926**	**0.921**	**0.924**	**0.918**

		Predicted Label		
		n	**p**	**Tot**
Actual Label	**n'**	4700	21	**4721**
	p'	32	814	**846**
	Tot	**4732**	**835**	

(a) The entire data set.

		Predicted Label		
		n	**p**	**Tot**
Actual Label	**n'**	4696	25	**4721**
	p'	27	315	**342**
	Tot	**4723**	**340**	

(b) The reduced data set.

Figure 5.5: Confusion matrices for TFIDF with text pre-processing for both variants of the data set.

5.2.2 SoftTFIDF

Table 5.4 shows the evaluation metrics for the SoftTFIDF implementation applied on both variants of the data set. The best results were achieved using text pre-processing, resulting in an F_1-score of 0.976 on the entire data set and 0.939 on the reduced data set. Both used a Jaro-Winkler threshold of 0.85 and cosine threshold of 0.40. Figure 5.6 shows the confusion matrices for SoftTFIDF on both variants of the data set using pre-processing.

Table 5.4: Evaluation metrics for SoftTFIDF with and without text pre-processing.

(a) The entire data set.

Text Pre-processing	Jaro-Winkler Threshold	Cosine Threshold	Precision	Recall	F_1	MCC
Without	0.80	0.40	0.967	0.954	0.961	0.954
With	0.85	0.40	**0.974**	**0.978**	**0.976**	**0.971**

(b) The reduced data set.

Text Pre-processing	Jaro-Winkler Threshold	Cosine Threshold	Precision	Recall	F_1	MCC
Without	0.80	0.35	0.913	0.889	0.901	0.894
With	0.85	0.40	**0.936**	**0.941**	**0.939**	**0.934**

		Predicted Label n	p	Tot
Actual Label	n'	4699	22	**4721**
	p'	19	827	**846**
	Tot	**4732**	**835**	

(a) The entire data set.

		Predicted Label n	p	Tot
Actual Label	n'	4699	22	**4721**
	p'	20	322	**342**
	Tot	**4719**	**344**	

(b) The reduced data set.

Figure 5.6: Confusion matrices for SoftTFIDF with text pre-processing for both data set variants.

5.2.3 RestrictedSoftTFIDF

Table 5.5 shows the evaluation metrics for the RestrictedSoftTFIDF implementation on both variants of the data set. The best performance was achieved with text pre-processing, resulting in an F_1-score of 0.969 with a Jaro-Winkler threshold of 0.95 and a cosine threshold of 0.25 for the entire data set. For the reduced data set the highest F_1-score, 0.913, was achieved with a Jaro-Winkler threshold of 0.90 and a cosine threshold of 0.25. Figure 5.7 shows the confusion matrices for the RestrictedSoftTFIDF on both variants of the data set using pre-processing.

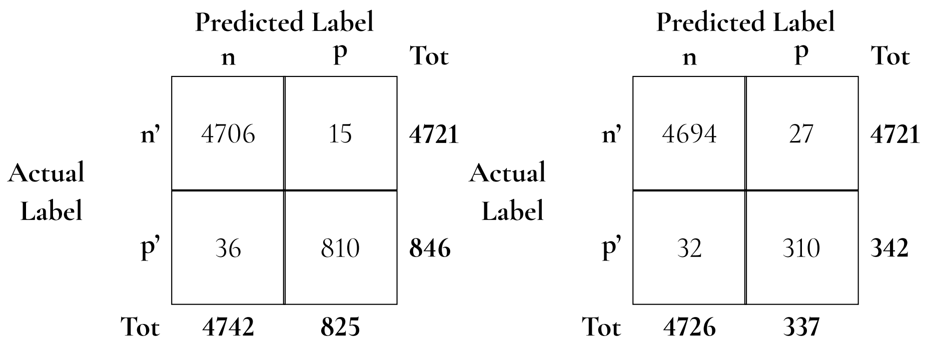
Table 5.5: Evaluation metrics for RestrictedSoftTFIDF with and without text pre-processing.

(a) The entire data set.

Text Pre-processing	Jaro-Winkler Threshold	Cosine Threshold	Precision	Recall	F_1	MCC
Without	0.90	0.25	0.967	0.927	0.946	0.937
With	0.95	0.25	**0.982**	**0.957**	**0.969**	**0.964**

(b) The reduced data set.

Text Pre-processing	Jaro-Winkler Threshold	Cosine Threshold	Precision	Recall	F_1	MCC
Without	0.90	0.25	0.912	0.819	0.863	0.855
With	0.90	0.25	**0.920**	**0.906**	**0.913**	**0.907**



(a)

Actual Label		Predicted Label n	Predicted Label p	Tot
	n’	4706	15	**4721**
	p’	36	810	**846**
	Tot	**4742**	**825**	

(b)

Actual Label		Predicted Label n	Predicted Label p	Tot
	n’	4694	27	**4721**
	p’	32	310	**342**
	Tot	**4726**	**337**	

(a) The entire data set. **(b)** The reduced data set.

Figure 5.7: Confusion matrices for RestrictedSoftTFIDF with text pre-processing for both data set variants.

5.3 Semantic Similarity Approaches

This section presents the results used when evaluating how semantic similarity can improve hybrid similarity functions, for which the implementations are described in section 4.3.

5.3.1 Pre-trained Word Embeddings

Table 5.6 shows the evaluation metrics of the pre-trained word embedding implementations with cosine similarity, using both variants of the data set. The highest score was achieved using BPEmb with a threshold of 0.60, resulting in an F_1-score of 0.947 for the entire data set and 0.869 for the latter. The F_1-scores when using a variation of hyperparameters for BPEmb with a cosine threshold of 0.60, are presented in Table 5.7. The results show that the highest F_1-score was achieved using a model with 300 embedding dimensions and a vocabulary size of 50,000.

Table 5.6: Evaluation metrics for pre-trained word embeddings with cosine similarity.

(a) The entire data set

Word Embedding	Cosine Threshold	Precision	Recall	F_1	MCC
BPEmb	0.60	0.950	**0.944**	**0.947**	**0.938**
BERT	0.80	**0.957**	0.832	0.890	0.875
SBERT	0.65	0.943	0.934	0.938	0.927

(b) The reduced data set.

Word Embedding	Cosine Threshold	Precision	Recall	F_1	MCC
BPEmb	0.60	**0.875**	**0.863**	**0.869**	**0.860**
BERT	0.80	0.862	0.585	0.697	0.694
SBERT	0.65	0.856	0.836	0.846	0.835

Table 5.7: F_1-scores for BPEmb with a cosine similarity threshold of 0.60, using different hyperparameters using the entire data set.

		Embedding Dimension		
		25	**100**	**300**
Vocabulary Size	**3000**	0.541	0.792	0.930
	10,000	0.688	0.864	0.946
	50,000	0.778	0.920	**0.947**
	200,000	0.731	0.906	0.937

The confusion matrices in Figure 5.8 and Figure 5.9 show the number of correctly and incorrectly classified pairs for each embedding respectively, for both variants of the data set.

Actual Label	Predicted Label: n	Predicted Label: p	Tot
n’	4679	42	**4721**
p	47	799	**846**
Tot	**4726**	**841**	

(a) BPEmb.

Actual Label	Predicted Label: n	Predicted Label: p	Tot
n’	4689	32	**4721**
p	142	704	**846**
Tot	**4831**	**736**	

(b) BERT.

Actual Label	Predicted Label: n	Predicted Label: p	Tot
n’	4673	48	**4721**
p’	56	790	**846**
Tot	**4729**	**838**	

(c) SBERT.

Figure 5.8: Confusion matrices for the pre-trained word embeddings using the entire data set.

		Predicted Label		
		n	p	Tot
Actual Label	n’	4679	42	**4721**
	p’	47	295	**342**
	Tot	**4716**	**337**	

(a) BPEmb.

		Predicted Label		
		n	p	Tot
Actual Label	n’	4689	32	**4721**
	p’	142	200	**342**
	Tot	**4831**	**232**	

(b) BERT.

		Predicted Label		
		n	p	Tot
Actual Label	n’	4673	48	**4721**
	p’	56	286	**342**
	Tot	**4729**	**334**	

(c) SBERT.

Figure 5.9: Confusion matrices for the pre-trained word embeddings using the reduced data set.

5.3.2 SemanticSoftTFIDF

Table 5.8 show the evaluation metrics for the SemanticSoftTFIDF algorithm for both variants of the data set. For both variants of data sets, the versions with BERT and SBERT achieved the same and the highest F$_1$-scores, 0.977 for the entire data set and 0.942 for the reduced data set. The thresholds used for BERT with both data set variants were a cosine threshold of 0.40, Jaro-Winkler threshold of 0.85 and cosine threshold of 0.95 for the word embeddings. The thresholds used for SBERT with both data set variants were a cosine threshold of 0.40, Jaro-Winkler threshold of 0.85 and cosine threshold of 0.75 for the word embeddings. The confusion matrices in Figure 5.10 and Figure 5.11 show the number of correctly and incorrectly classified pairs for each embedding respectively, for both variants of the data set.

Table 5.8: Evaluation metrics for SemanticSoftTFIDF.

(a) Evaluation metrics using the entire data set.

Cosine Threshold	Jaro-Winkler Threshold	Embedding Threshold	Embedding	Precision	Recall	F$_1$	MCC
0.40	0.85	0.70	BPEmb	0.973	**0.980**	0.976	0.972
0.40	0.85	0.95	BERT	**0.974**	**0.980**	**0.977**	**0.973**
0.40	0.85	0.75	SBERT	**0.974**	**0.980**	**0.977**	**0.973**

(b) Evaluation metrics using the reduced data set.

Cosine Threshold	Jaro-Winkler Threshold	Embedding Threshold	Embedding	Precision	Recall	F$_1$	MCC
0.40	0.85	0.70	BPEmb	0.934	**0.947**	0.940	0.936
0.40	0.85	0.95	BERT	**0.936**	**0.947**	**0.942**	**0.938**
0.40	0.85	0.75	SBERT	**0.936**	**0.947**	**0.942**	**0.938**

		Predicted Label		
		n	**p**	**Tot**
Actual Label	**n'**	4698	23	**4721**
	p	17	829	**846**
	Tot	**4715**	**852**	

(a) SemanticSoftTFIDF with BPEmb.

		Predicted Label		
		n	**p**	**Tot**
Actual Label	**n'**	4699	22	**4721**
	p	17	829	**846**
	Tot	**4716**	**851**	

(b) SemanticSoftTFIDF with BERT.

		Predicted Label		
		n	**p**	**Tot**
Actual Label	**n'**	4699	22	**4721**
	p'	17	829	**846**
	Tot	**4716**	**851**	

(c) SemanticSoftTFIDF with SBERT.

Figure 5.10: Confusion matrices for SemanticSoftTFIDF with the pre-trained word embeddings using the entire data set.

		Predicted Label		
		n	**p**	**Tot**
Actual Label	**n'**	4698	23	**4721**
	p'	18	324	**342**
	Tot	**4716**	**347**	

(a) SemanticSoftTFIDF with BPEmb.

		Predicted Label		
		n	**p**	**Tot**
Actual Label	**n'**	4699	22	**4721**
	p'	18	324	**342**
	Tot	**4717**	**346**	

(b) SemanticSoftTFIDF with BERT.

		Predicted Label		
		n	**p**	**Tot**
Actual Label	**n'**	4699	22	**4721**
	p'	18	324	**342**
	Tot	**4717**	**346**	

(c) SemanticSoftTFIDF with SBERT.

Figure 5.11: Confusion matrices for SemanticSoftTFIDF with the pre-trained word embeddings using the reduced data set.

5.4 Machine Learning Approaches

This section presents the results of the evaluation metrics when investigating how machine learning can improve hybrid approaches, for which the implementations are described in section 4.4.

5.4.1 Machine Learning Using Similarity Functions

This section presents the results for the machine learning models using only similarity functions as input. The results are presented using the hyperparameters chosen to achieve the best results for each model. Table 5.9 shows evaluation metrics for random forest, gradient boosted trees and the MLP-classifier. The highest F_1-score, 0.988, was obtained using the gradient boosted tree classifier with the similarity functions in variant 5. Figure 5.12 shows the feature importance for the classifiers for variant 5. The most important similarity function for all classifiers was one of the hybrid approaches. Table 5.10 shows the average evaluation metrics for the classifiers when using cross-validation on variant 5. The highest results were achieved with the random forest classifier and the gradient boosted tree classifier, with an F_1-score of 0.984. Figure 5.13 shows the confusion matrices for the classifiers' averages when using cross-validation on the features in variant 5.

Table 5.9: Evaluation metrics for the models.

(a) Random forest.

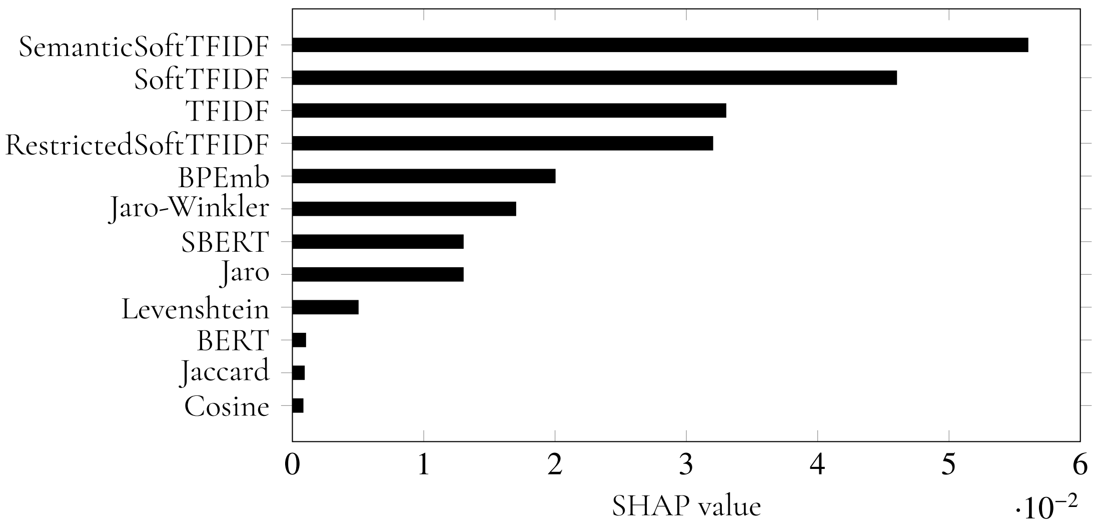
Variant	Precision	Recall	F_1	MCC
1	0.948	0.957	0.953	0.944
2	0.986	**0.981**	**0.984**	**0.981**
3	0.986	**0.981**	**0.984**	**0.981**
4	**0.991**	0.977	**0.984**	**0.981**
5	**0.991**	0.977	**0.984**	**0.981**

(b) Gradient boosted trees.

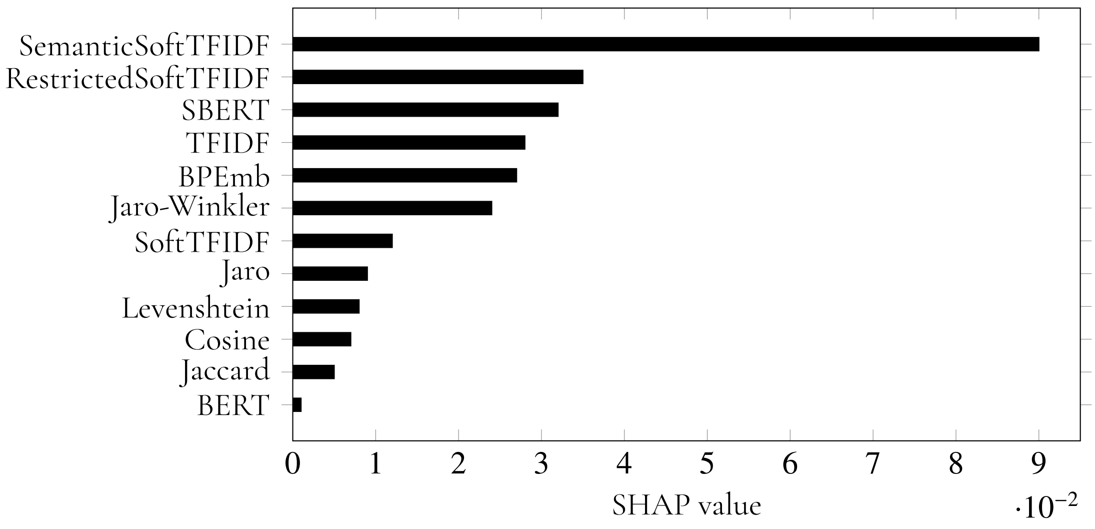
Variant	Precision	Recall	F_1	MCC
1	0.962	0.958	0.960	0.953
2	0.986	0.981	0.984	0.981
3	0.981	0.977	0.979	0.975
4	**0.991**	0.977	0.986	0.981
5	**0.991**	**0.986**	**0.988**	**0.986**

(c) MLP-classifier.

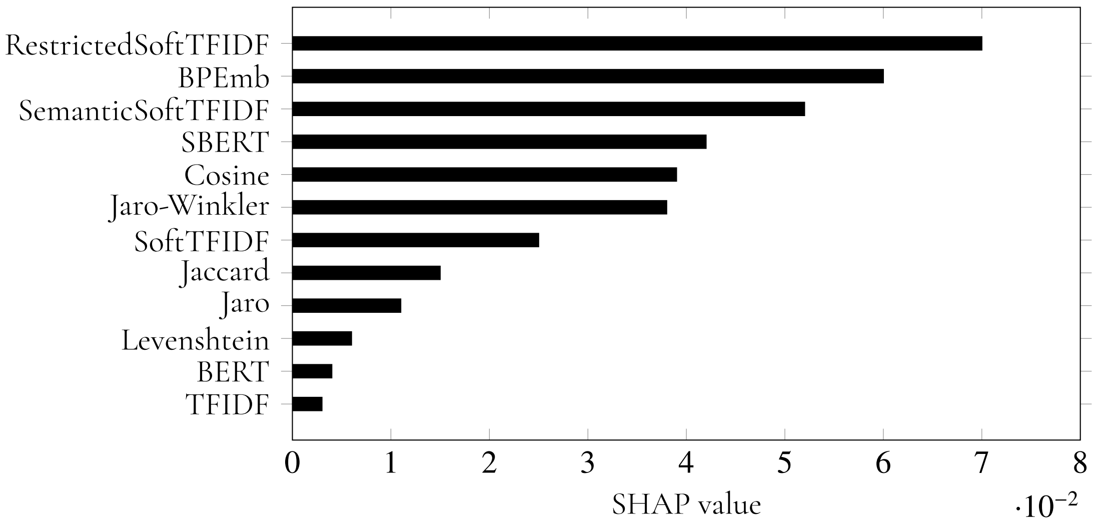
Variant	Precision	Recall	F_1	MCC
1	0.948	0.962	0.955	0.947
2	0.991	**0.972**	0.981	0.978
3	0.986	**0.972**	0.979	0.975
4	**0.995**	**0.972**	**0.984**	**0.981**
5	**0.995**	**0.972**	**0.984**	**0.981**



(a) Random forest.



(b) Gradient boosted trees.



(c) MLP-classifier.

Figure 5.12: Feature importance for the models with the similarity functions in variant 5.

Table 5.10: Average evaluation metrics using cross-validation with variant 5.

Classifier	**Average Precision**	**Average Recall**	**Average F_1**	**Average MC**
Random forest	0.988	**0.981**	**0.984**	**0.982**
Gradient boosted trees	0.984	0.979	0.981	0.978
MLP-classifier	**0.989**	0.977	0.983	0.980

		Predicted Label n	**Predicted Label** p	**Tot**
Actual Label	n'	4705	16	**4721**
Actual Label	p'	10	836	**846**
	Tot	**4715**	**852**	

(a) Random forest.

		Predicted Label n	**Predicted Label** p	**Tot**
Actual Label	n'	4703	18	**4721**
Actual Label	p'	13	833	**846**
	Tot	**4716**	**851**	

(b) Gradient boosted trees.

		Predicted Label n	**Predicted Label** p	**Tot**
Actual Label	n'	4702	19	**4721**
Actual Label	p'	9	837	**846**
	Tot	**4711**	**856**	

(c) MLP-classifier.

Figure 5.13: Confusion matrices for cross-validation of the models with feature vector variant 5.

5.4.2 Machine Learning With Additional Features

This section presents the results for the machine learning approaches with additional features. Table 5.11 shows the evaluation metrics for the models applied on feature variants 6 and 7. All classifiers achieved F_1-score of 0.981 for variant 6. The gradient boosted tree and MLP achieved the highest F_1-score of 0.986 for variant 7. Figure 5.14 shows the feature importance of the features for the classifiers for variant 7. Just as for the variant without additional features, the most important feature for all classifiers was one of the hybrid approaches. The feature importance of the token length was quite low, although the results show a minor positive impact on the classifiers. Table 5.13 shows the average evaluation metrics for the classifiers when using cross-validation on variant 7. The highest result was achieved with the random forest classifier, with an F_1-score of 0.986. Figure 5.15 shows the confusion matrices for the classifiers averages when using cross-validation on feature variant 7.

Table 5.11: Evaluation metrics for the models on variants 6 and 7.

(a) Random forest.

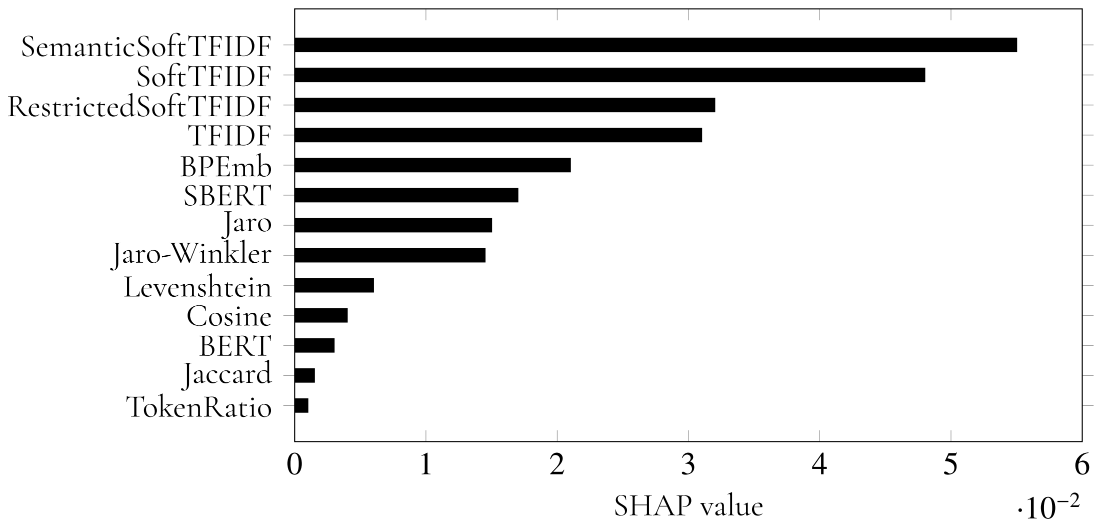
Variant	Precision	Recall	F_1	MCC
6	**0.991**	0.972	0.981	0.978
7	**0.991**	**0.977**	**0.984**	**0.981**

(b) Gradient boosted trees.

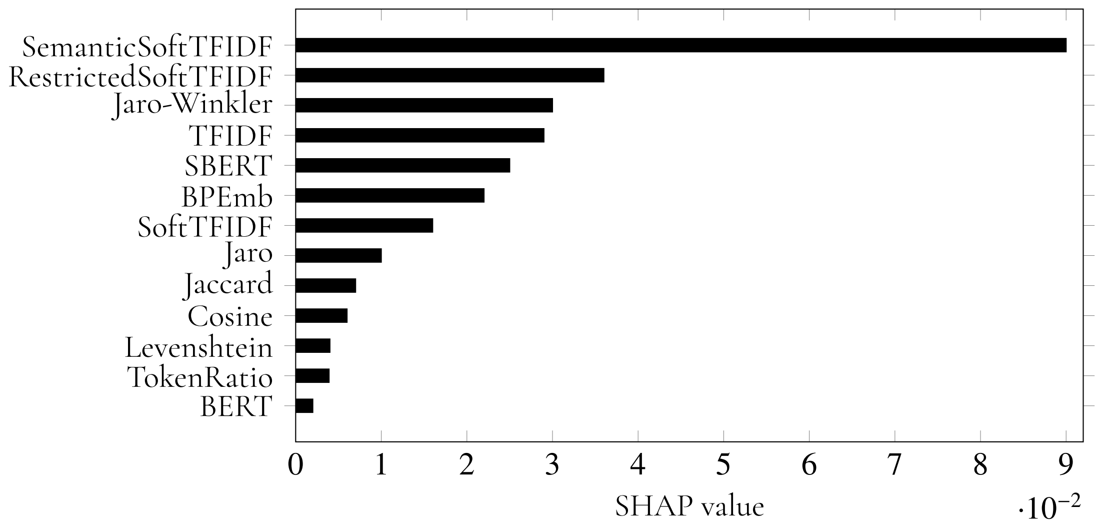
Variant	Precision	Recall	F_1	MCC
6	0.986	0.977	0.981	0.978
7	**0.996**	**0.981**	**0.986**	**0.983**

(c) MLP-classifier.

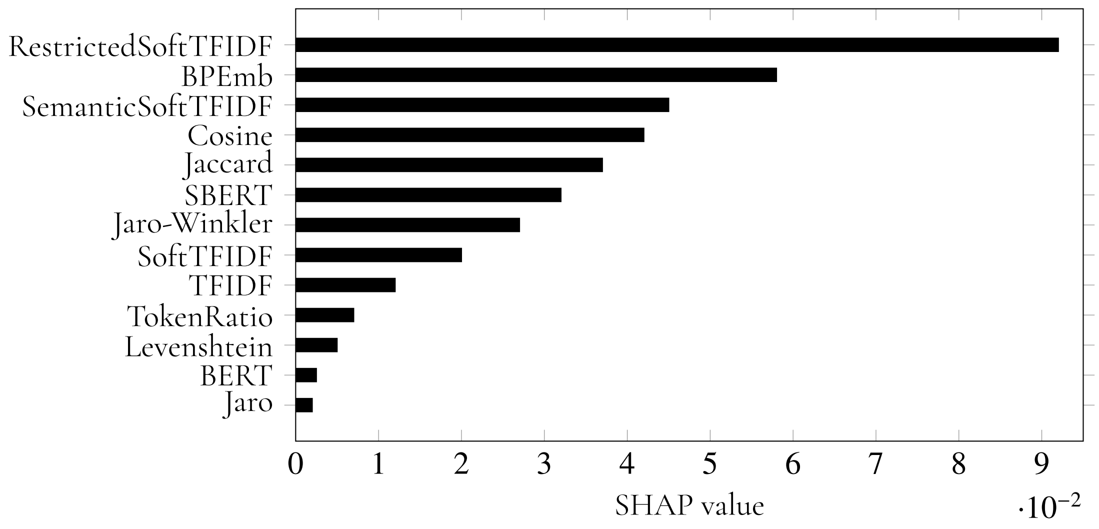
Variant	Precision	Recall	F_1	MCC
6	0.991	0.972	0.981	0.978
7	**0.995**	**0.977**	**0.986**	**0.983**



(a) Random forest.



(b) Gradient boosted trees.



(c) MLP-classifier.

Figure 5.14: Feature importance for the models with the similarity functions in variant 7.

Table 5.12: Average evaluation metrics for cross-validation of the classifiers using feature variant 7.

Classifier	**Average Precision**	**Average Recall**	**Average F_1**	**Average MC**
Random forest	0.990	0.981	**0.986**	**0.983**
Gradient boosted trees	0.984	**0.983**	0.983	0.980
MLP-classifier	**0.992**	0.976	0.984	0.981

Table 5.13: Average evaluation metrics for cross-validation of the classifiers using feature variant 7.

Classifier	**Average Precision**	**Average Recall**	**Average F_1**	**Average MC**
Random forest	0.990	0.981	**0.986**	**0.983**
Gradient boosted trees	0.984	**0.983**	0.983	0.980
MLP-classifier	**0.992**	0.976	0.984	0.981

Actual Label		**Predicted Label** n	**Predicted Label** p	**Tot**
	n’	4705	16	**4721**
	p’	8	838	**846**
	Tot	**4713**	**854**	

(a) Random forest.

Actual Label		**Predicted Label** n	**Predicted Label** p	**Tot**
	n’	4707	14	**4721**
	p’	14	832	**846**
	Tot	**4721**	**846**	

(b) Gradient boosted trees.

Actual Label		**Predicted Label** n	**Predicted Label** p	**Tot**
	n’	4701	20	**4721**
	p’	7	839	**846**
	Tot	**4708**	**859**	

(c) MLP-classifier.

Figure 5.15: Confusion matrices for cross-validation of the models with feature variant 7.

5.5 Examples of Classifications

Tables 5.14 and 5.15 show how the algorithms investigated in this thesis classify several example pairs of POIs. All examples are taken from the data set used in the evaluation. The table shows the algorithms' performance when using the best thresholds and parameters, according to the results presented earlier in this chapter. The algorithms compared in the table, with corresponding indices, are:

1. Jaro-Winkler similarity
2. Cosine similarity
3. TFIDF
4. SoftTFIDF
5. RestrictedSoftTFIDF
6. BPEmb with cosine similarity
7. SBERT with cosine similarity
8. BERT with cosine similarity
9. SemanticSoftTFIDF using BPEmb
10. SemanticSoftTFIDF using SBERT
11. SemanticSoftTFIDF using BERT
12. Random forest classifier for feature variant 7
13. Gradient boosted trees for feature variant 7
14. MLP-classifier for feature variant 7

Table 5.14 contains examples of matching pairs. The checkmark, ✓, indicates the algorithms that correctly classified the pair as a match and the X indicates the algorithms that incorrectly classified the pair. Table 5.15 contains examples of non-matching pairs and the checkmark indicates the algorithms that correctly classified the pair as a non-match and the X indicates the algorithms that incorrectly classified the pair.

Table 5.14: Examples of classifications on matching POI pairs for the algorithms.

Names	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
HI Boston Hostelling International	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Spagnulos La Famiglia Spagnuolo's	X	X	X	✓	✓	X	X	X	✓	✓	✓	✓	✓	✓
Fin's Fins Sushi and Grill	X	X	✓	✓	✓	X	X	X	✓	✓	✓	✓	✓	✓
Boston Beer Works BEERWORKS No. 3 Canal	X	X	X	✓	X	X	X	X	✓	✓	✓	X	X	X
Pro Optical ProOptical Boston	✓	X	X	✓	X	X	X	X	✓	✓	✓	✓	✓	✓
MIJA Cantina Mija Cantina & Tequila Bar	X	X	✓	✓	✓	✓	✓	X	✓	✓	✓	✓	✓	✓
Penang Malaysian Cuisine Penang	✓	✓	✓	X	✓	✓	✓	X	X	X	X	✓	✓	✓
Mooo.... Mooo Restaurant	X	X	✓	✓	✓	✓	X	X	✓	✓	✓	✓	✓	✓
TD Canada Trust Td Bank Financial Group	X	X	X	X	X	✓	✓	X	X	X	X	X	X	✓
Biercraft Bier Craft Bistro	✓	X	X	✓	X	X	✓	X	✓	✓	✓	✓	✓	✓
Bâtard Batard Boulangerie	X	X	✓	✓	✓	X	X	X	✓	✓	✓	✓	✓	✓
Veggie Bowl Veggiebowl	✓	X	X	X	X	✓	✓	✓	X	X	X	✓	X	✓
Florida Hospital Cancer Institute Cancer Institute of Florida	X	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Ba-Le Deli & Bakery Ba Le Sandwich Shop	X	X	X	✓	X	X	✓	X	✓	✓	✓	✓	✓	✓
Qdoba QDOBA Mexican Eats	X	X	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 5.15: Examples of classifications on non-matching POI pairs for the algorithms.

Names	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Winter Park Historical Museum Winter Park Farmers' Market	X	X	✓	✓	✓	X	X	✓	✓	✓	✓	✓	✓	✓
Tufts Medical Center Tufts Medical Center MBTA Station	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Boston Common Coffee Co. Boston Bean Stock Coffee	X	X	✓	✓	✓	X	X	X	✓	✓	✓	✓	✓	✓
Roche Bros. Sabroso Taqueria	✓	✓	✓	X	✓	✓	✓	✓	X	X	X	✓	✓	✓
Haymarket Haymarket Center Garage	X	X	X	X	✓	X	X	✓	X	X	X	X	X	X
Sushi TonTon Sushi Aria	X	X	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Pho Long Restaurant Pho Van	X	X	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Green Leaf Bubble Tea Green Leaf Natural Food	X	X	X	X	✓	X	✓	X	X	X	X	X	X	X
Collingwood Neighbourhood School (Bruce Annex) Collingwood Neighbourhood School	X	X	X	✓	X	X	X	X	✓	✓	✓	X	X	X
Champlain Mall Champlain Square	X	X	✓	✓	X	X	X	✓	✓	✓	✓	✓	✓	✓
Green Thumb Theatre Green Thumb Theatre for Young People - Studio 1	X	X	X	X	X	X	X	✓	X	X	X	✓	✓	X
Cary Cary Train Station	X	X	X	X	✓	✓	X	✓	X	X	X	X	X	X

5.6 Error Reduction

Table 5.16 presents the error reduction achieved with the best performing approach from each part of the thesis. The error reduction is calculated from the F_1-score of 0.957, which was the best result in the baseline. The largest error reduction was achieved by the random forest model using feature variant 7, with an error reduction of 67% from the baseline.

Table 5.16: Error reduction from the baseline in percent.

Approach	Error Reduction
SoftTFIDF	44%
SemanticSoftTFIDF (SBERT)	47%
Random Forest (feature variant 7)	**67%**

Chapter 6

Discussion

In this chapter we discuss the results presented in Chapter 5 and the evaluation of the approaches. Interesting findings and aspects that affected the results are analyzed. The first section covers important aspects in the baseline. The following three sections discuss the results for the hybrid approaches, improvements when using semantics and the machine learning approaches. The evaluation methods and limitations of the work are also discussed.

6.1 Baseline

As presented in section 5.1, Jaro-Winkler similarity and cosine similarity achieved the highest F_1-scores, for character-based and token-based respectively among the investigated state-of-the-art similarity functions. Therefore, these were established as the baseline for this project. The F_1-score for Jaro-Winkler similarity was 0.957, which can be considered high for a similarity function that only considers lexical aspects. This can be interpreted in such a way that considering lexical similarity can be beneficial for POI name matching.

Cosine similarity achieved an F_1-score of 0.930, which was lower than Jaro-Winkler similarity. Since cosine similarity is calculated between vectors, the score is highly dependent on how the vectors are constructed. The vectors in the baseline were created using BoW. With more complex information in the vectors, the performance of cosine similarity could possibly have been higher, which is also what we found later in the work.

Another interesting aspect related to the two variants of data set is the relation between the performance and the selected threshold, which is shown in Figure 5.1 and Figure 5.3. For both Jaro-Winkler and cosine the F_1-scores were in general lower for the same thresholds when using the reduced data set compared to using the entire data set. For Jaro-Winkler similarity, the trend was relatively similar for the two data set variants. More interesting is the difference in trends for cosine similarity between the data set variants. For the reduced data set the negative gradient was larger when the thresholds exceeded 0.5. The difference in trends between the data sets is an indicator that the selection of thresholds is related to the characteristics of the data set.

6.2 Hybrid Similarity Functions

In this section we discuss the results presented in section 5.2, for evaluating how hybrid approaches can improve performance of state-of-the-art similarity functions for POI name matching.

6.2.1 Analysis of Results

In Table 5.3 it can be seen that the TFIDF algorithm achieved higher scores in all evaluation metrics compared to the baseline. The highest F_1-score for the TFIDF for the entire data set was 0.968 and 0.924 for the reduced data set. The results show that without pre-processing, TFIDF achieved an F_1-score of 0.940 using the entire data set and 0.848 using the reduced data set, which is an improvement from cosine similarity in the baseline with F_1-scores of 0.930 and 0.827. This is interesting since these are both token-based similarity functions using cosine similarity without any pre-processing. The difference is that in the baseline, cosine similarity is computed for the names BoW vectors, compared to TFIDF vectors as in this approach. This indicates that the TFIDF vectors provide more useful information about the names than the BoW vectors, when determining the similarity.

The evaluation metrics for evaluating the SoftTFIDF algorithm, presented in Table 5.4, show improved performance compared to both the baseline and TFIDF. The algorithm achieved an F_1-score of 0.976 on the entire data set with pre-processing, which is a slight improvement from 0.968, achieved with TFIDF. This is not a major improvement, however, looking at the results when using the reduced data set, the SoftTFIDF achieves an F_1-score of 0.939 compared to 0.924 with TFIDF. This indicates that the SoftTFIDF does perform better for the non-trivial cases, which is a positive find.

For the RestrictedSoftTFIDF the F_1-scores of 0.969 and 0.913 were achieved using pre-processing on the entire data set and the reduced data set, as shown in Table 5.3. When the entire data set is used, the results show improvements from the TFIDF. For the reduced data set there is a slight loss in performance, where the F_1-score is decreased to 0.913 compared to 0.924. However, when not using pre-processing for this data set, RestrictedSoftTFIDF performs better than TFIDF, with an F_1-score of 0.863 compared to 0.848. The results indicate that there is an improvement using RestrictedSoftTFIDF from TFIDF, however not an improvement from SoftTFIDF. Additionally, SoftTFIDF has a larger improvement from TFIDF. This can be interpreted in such way that the "soft" part, implemented in both algorithms, is the main reason for the improved performance from TFIDF. This indicates that considering approximately similar tokens as matching has a positive impact on the performance of POI matching.

It is interesting to understand why RestrictedSoftTFIDF performed worse than SoftTFIDF. This could partly be explained by the fact that the corpus used for calculating the TFIDF values was much smaller than for the SoftTFIDF. Even though the corpus contained more tokens that were important for the geographical location relative to the corpus size, it also had the disadvantage of not being able to lower the importance of business-related words, since they did not occur as frequently in the smaller corpus.

An interesting observation from the results of the confusion matrices, see Figures 5.5, 5.6 and 5.7, is that when altering the data set used for each of the implementations, all values in the confusion matrices were affected. This differs from what we saw in the baseline where only the true positives were affected. This is due to the hybrid approaches using a corpus to calculate and create the TFIDF vectors, which is based on the other POIs in the data set and therefore introducing a dependency.

From the evaluation of the hybrid similarity functions investigated in this part of the thesis, we found that the SoftTFIDF using pre-processing performed the best for POI name matching.

6.2.2 Thresholds

We observed that the similarity thresholds achieving the best performance differed among the hybrid similarity function, both when altering the pre-processing and the data set used. The thresholds for cosine similarity in the algorithms are not directly comparable to the thresholds for cosine similarity in the baseline. This is because the vectors contain different information and therefore the similarity scores are calculated based on slightly different information. It is however interesting that for the different data set variants within each algorithm the thresholds differ. This indicates that the thresholds need to be tuned differently depending on the evaluated data, which supports the argument of not manually tuning thresholds. The results also show that the thresholds are slightly higher when the data is pre-processed, which is not surprising since there is less noise that could make the names more different and therefore make the algorithm achieve higher similarity scores in general, even for non-matches.

6.2.3 Text Pre-processing

We observed that for all algorithms evaluated for this part, the results show better performance when text pre-processing is used. For TFIDF, the F_1-score improved from 0.940 to 0.968 when using the entire data set, and from 0.848 to 0.924 when using the reduced data set. Similarly, we observed that pre-processing improved the performance of the SoftTFIDF. For the entire data set, the text pre-processing improved the F_1-score from 0.961 to 0.976, and for the reduced data set it improved from 0.901 to 0.939. Even for the RestrictedSoft-TFIDF the text pre-processing shows improved performance, with the F_1-score increasing from 0.946 to 0.969, when using the entire data set, and increasing from 0.863 to 0.913 when using the reduced data set. For all algorithms, these are rather large improvements in F_1-scores which indicates that pre-processing has a positive impact on the performance.

The reason why pre-processing improves the performance can be explained by the fact that the data that is compared is cleaner. In the pre-processing procedure, described in section 4.2.1, all noise is removed by for example lower-casing and removing special characters. This eliminates small variations, insignificant to the meaning, between the names and therefore makes them more lexically similar. Based on the evaluations, we could see a clear indication that using pre-processing improved the performance for the hybrid approaches used for POI name matching.

6.2.4 Finding Approximate Tokens

A positive find when evaluating the results was that introducing the "soft" part of the hybrid algorithms indicated that approximate tokens were successfully identified as identical tokens. This helped solve the problem of variants of tokens not being considered similar when applying token-based similarity functions, such as calculating the TFIDF values. As an example, consider the matching POIs "Spagnulos" and "La Famiglia Spagnuolo's", in Table 5.14. The pair was incorrectly classified by the baseline as well as TFIDF, but correctly classified by SoftTFIDF and RestrictedSoftTFIDF. After pre-processing, these resulted in the token sets {"spagnulos"} and {"la", "famiglia", "spagnuolos"}. Due to the variation of spelling in "spagnulos" and "spagnuolos", these were not counted as the same token when calculating the TFIDF values and therefore the cosine similarity between the vectors was low. Introducing the Jaro-Winkler in the hybrids to identify approximate tokens, as in SoftTFIDF and RestrictedSoftTFIDF, seemed to mitigate the problem. For these algorithms, the variants of "spagnulos" and "spagnuolos" achieved a high Jaro-Winkler similarity and was therefore counted as the same token when calculating the TFIDF values, resulting in vectors with higher similarity. The observations indicate that using a character-based similarity function to find approximate tokens has a positive impact on performance when combined with token-based similarity functions.

6.2.5 Weighting Words

The idea behind using TFIDF was to investigate how weighting of tokens by occurrence could make certain parts of the names more or less important. For the task of POI name matching, we found that there is often a difference in how business-related words, for example "restaurant", were used. When evaluating the results, we saw that by assigning a lower weight to these tokens when calculating their TFIDF values, they seemed to be less important when determining the similarity between the vectors. Several examples of this can be seen in Table 5.14, such as the matching POI names "Mooo..." and "Mooo Restaurant" as well as "Bâtard" and "Batard Boulangerie". The business-related words "Restaurant" and "Boulangerie" were more likely to occur often for POI names, assigning them a lower TFIDF value. As a result, the vectors' similarity was determined mainly by the similarity between the tokens "Mooo..." and "Mooo" as well as "Bâtard" and "Batard", with higher similarity scores. Similarly, Table 5.15 shows that the non-matching POI pair "Sushi TonTon" and "Sushi Aria" were correctly classified by the TFIDF-based algorithms. Since "Sushi" was more frequently occurring in the corpus than "TonTon" and "Aria", the fact that these are different from each other was more important than the fact that "Sushi" was a common token. The similarity between the vectors of the names was therefore determined mainly from the similarity between the tokens "TonTon" and "Aria", which was low and therefore the pair is correctly classified. Another benefit from using weights is that it helps solve the problem of differing numbers of tokens between the names in a pair. Lowering the importance of the additional tokens can make the POIs' vectors more similar.

The above-mentioned cases were not correctly classified by the baseline, but correctly classified by the hybrid approaches where TFIDF was introduced. The observations indicate that weighting of tokens depending on their importance for the name can help improve performance for POI name matching. For the case of POI names used in this work, we see that many POIs were businesses and therefore for this context, it was beneficial that business-related words have less of an impact than the actual name of the business. Although, which parts of the names to be considered more or less important for the name is highly context-dependent and is correlated to the corpus used when calculating the TFIDF values. The selection of corpus and weighting scheme could be further adapted for the context and the task at hand, and is something that could be further investigated for further improved performance on specific contexts.

6.2.6 Weighting Words by Location

Another interesting find when analyzing the classifications, was that for some POI pairs RestrictedSoftTFIDF does fulfill the desired outcome of assigning a lower weight to location-related words. For example, the pair "Haymarket" and "Haymarket Center Garage" are both located in the Haymarket area. As can be seen in Table 5.15, none of the baseline, TFIDF and SoftTFIDF were able to correctly classify the pair, while the RestrictedSoftTFIDF classified it correctly. Similarly, the pair "Cary" and "Cary Train Station" in the Cary neighborhood, was only classified correctly by the RestrictedSoftTFIDF and not by the baseline, TFIDF or SoftTFIDF. This is an indication that the algorithm did in fact manage to solve this problem to some extent. However, the results clearly show a poorer performance in general. As mentioned earlier, this was likely to be caused by the small corpus and lack of making business-related words in POI names less important.

For this approach, we chose to consider the POIs in the surrounding area within a distance of 25 meters. Adjusting the area surrounding the POIs, from which the corpus is to be created, could possibly improve the results and would be an interesting investigation. For example, for POI pairs that are not located close to any other POI, which is common in rural areas, choosing an area that is too small would introduce the risk of the corpus being created only from that pair, meaning that the TFIDF weighting would not be very useful. Depending on the location, the optimal size of the surrounding area to be considered in the corpus can vary, which requires further exploration on how to select the areas for different POIs within the same data set. For some POIs it would be interesting to create the corpus from the entire city in which it is located, while for others, it would be more beneficial to only consider POIs located in a neighborhood or street. There is potential to further investigate solutions to this problem, for example by creating combinations of multiple corpora and then being able to adjust the weights both according to the area, as in RestrictedSoftTFIDF, and the business-related words in the entire corpus, as in SoftTFIDF.

6.3 Semantic Similarity Approaches

This section aims to discuss the results presented in section 5.3, evaluating how semantic similarity can be used to improve performance of hybrid approaches for POI name matching.

6.3.1 Analysis of Results

Using pre-trained word embeddings together with cosine similarity did not show impressive results. As seen in Table 5.6, BPEmb achieved the highest F_1-scores among the embeddings, 0.947 for the entire data set and 0.869 for the reduced data set. These scores were lower than the scores achieved in the baseline with Jaro-Winkler. The scores were slightly higher than for cosine similarity in the baseline, which once again highlights how cosine similarity depends on the embeddings.

Table 5.7 presents the obtained F_1-scores from alternating hyperparameters, embedding dimension and vocabulary size, for BPEmb. We compared all possible combinations of hyperparameters provided by the BPEmb package for the English language. This shows varying results, where a small vocabulary size in combination with a low dimensionality received an F_1-score of 0.541, which is much lower than any of the state-of-the-art similarity functions performed during the process of establishing the baseline. The results indicate that a higher dimensionality seems to work better for the task but implies more complexity.

Semantic similarity was also used in one of the implemented hybrids functions, namely SemanticSoftTFIDF. The best F_1-scores using SemanticSoftTFIDF was achieved with BERT and SBERT, 0.977 for the entire data set and 0.942 for the reduced data set. These scores are marginally higher than for SoftTFIDF. By looking at the SemanticSoftTFIDF confusion matrices in Figure 5.10 and 5.11 and compare the classifications with the confusion matrices in Figure 5.6, corresponding to the results of SoftTFIDF, we can see that only one or two false negatives or positives differs between the matrices. This observation makes it difficult to draw any conclusion about the impact of semantic similarity in the hybrid approaches. Also, seeing that BPEmb achieved the highest F_1-score among the pre-trained word embeddings used with cosine, we expected it to be the best performing embedding in SemanticSoftTFIDF as well. BERT's result when used with cosine similarity was not impressive at all as it did not even exceed the baseline. As seen for the SemanticSoftTFIDF results in Table 5.8, the difference between the three word embeddings was minimal.

6.3.2 Embeddings and Context

Modern word embedding techniques are constructed to capture complex information about the meaning of words. Usually, the context of the word is a major contributor to capture that information. POI names are relatively short and due to the limited context, it is difficult to capture useful contextual information. This certainly affected the performance of using the pre-trained word embeddings in our approaches, especially in SemanticSoftTFIDF where tokens in the names were embedded separately with even less contextual information.

The fact that the embedded tokens were pre-processed may also have had a negative effect on the ability to capture contextual information as intended. This could be explained by the fact that the embeddings can benefit from the usage of special characters or capital letters in training to identify structure and meaning, and that this help is removed by pre-processing the names.

Worth noting is that the pre-trained embeddings used in this work were not trained on data specific for the task. Fine-tuning the model with POI data could possibly lead to the embeddings containing more domain-specific information and in that way be more useful for determining similarity between POIs. Even though this thesis covers POIs from North America, there were several cases in the data set where the names were not in English or where the names were made up. This made the task more complex since the pre-trained word embeddings were all trained on texts in English. However, the latter mentioned issue of made-up words would not be solved with training on other languages.

6.3.3 Semantics

Since name matching tasks traditionally handle lexical similarity, the goal of using pre-trained word embeddings in the thesis was to introduce another aspect, semantics, that could possibly help in the process of determining similarity. Even though using semantics did not significantly improve the results, we could see an effect of the semantics when comparing the incorrectly classified pairs of the TFIDF-based approaches with the approaches using pre-trained word embeddings with cosine similarity.

POIs within a pair can imply different semantic meaning, even though they are the same entity. Examples of this can be seen in Table 5.14 and 5.15. For the matching pair "Fin's" and "Fins Sushi and Grill", the semantic meaning of the entity changes from a name, in the first POI name, to a business when adding "Sushi and Grill" in the second POI name. Another example of a pair where the semantic meaning differs is the non-matching pair "Cary" and "Cary Train Station", where the first refers to a geographical area and the second to a train station within that same area. Because of the semantic differences between the POIs, both examples were classified as non-matching pairs when using pre-trained word embeddings with cosine, which means that the embeddings seem to capture semantic meaning as expected. This demonstrates the complex issue that semantic similarity can both be helpful and not helpful, depending on the context. As discussed earlier, differing length of POIs in a pair is a problem, which also seemed to be the case when semantics is considered. Mostly because the additional words in one of the POIs can be important and change the semantic meaning, which is only beneficial in some cases. However, it is not easy to determine those cases where semantic similarity contributes to or complicates the task when little context is provided.

The way semantic similarity was added to SoftTFIDF in SemanticSoftTFIDF did not seem to contribute to determining similarity. Although the inner character-based algorithm together with the weighted frequency values seemed to be the important parts of the SemanticSoftTFIDF, we cannot conclude semantic similarity as useless for the POI name matching. As explained earlier, we could see cases where embeddings contribute to the classification by capturing the semantic meaning. Since the embeddings are constructed using context to create meaningful embeddings, it could perhaps be beneficial to embed the entire name at once rather than token-wise and combine this with the hybrid approach instead.

6.4 Machine Learning Approaches

In this section we discuss the results presented in section 5.4, used to evaluate how machine learning approaches can improve performance of similarity functions for POI name matching. Since the machine learning approaches were only evaluated on the entire data set, in order to maximize the amount in training, the analysis will only be based on this data set variant and not compared with results obtained from using only the reduced data set in the earlier approaches.

6.4.1 Analysis of Results

Table 5.9 presents the results of the machine learning models using various feature variants, on the test data, i.e. data that was not seen during training of the models. It should be noted that for these results the models are trained on 80% of the entire data set and evaluated by predicting the remaining data. This means that the evaluation metrics in the tables are not directly comparable to the evaluation metrics presented for earlier approaches in this work. It does however indicate that the models do perform well on unseen data, meaning they perform well on general data for the task. Even though not evaluated on the exact same data as the earlier approaches, the results also give an indication of how well the models perform for POI name matching using various feature vectors. Since the data were seeded, the results in Table 5.9 are all trained and tested on the same data, which make them directly comparable to each other.

The results show that for feature variant 1, where only the baseline algorithms are used in the feature vector, all three machine learning models perform poorly and have lower F_1-scores than the other feature variants. This is an indication that even with machine learning, there is not enough valuable information only in the baseline for the models to determine the similarity of the POIs correctly. For variant 3 we can see that F_1-scores are slightly lower than for variants 2, 4 and 5, for all three machine learning models. This is interesting as variant 3 includes the baseline, SoftTFIDF and the pre-trained word embedding BPEmb with cosine, but, unlike variants 2, 4 and 5, variant 3 does not include a SemanticSoftTFIDF algorithm. This indicates that the SemanticSoftTFIDF is important for the models' predictions, which is discussed further in section 6.4.3. For these variants, the three machine learning models have similar F_1-scores, but overall variant 5 achieved the highest F_1-score of all models, the highest being 0.988 with the gradient boosted trees classifier.

The results when adding the features of distance, variant 6, and token length ratio, variant 7, can be seen in Table 5.11. The results indicate that there is a slight increase in performance for variant 7 compared to the results from the models' predictions when not using these additional features. For variant 6 there was a slight decrease for all three models. For variant 7, the random forest achieves the same F_1-score of 0.984, gradient boosted trees a slightly lower F_1-score of 0.986 and the MLP-classifier a slightly higher F_1-score of 0.986.

Since variants 5 and 7 performed the best, these were evaluated further using stratified 5-fold cross-validation on the entire data set. Table 5.10 presents the average evaluation metrics of the models on feature variant 5. This means that every pair has been used for both training and validation in some part of the cross-validation. As mentioned earlier by Santos et al.'s [4], this makes the results comparable to the earlier approaches since all data is evaluated. The results show that the highest F_1-score of 0.984 was achieved using random forest, which is higher than all previous approaches in the work when evaluated on the entire data set. These results show an improvement from the previous best performance of an F_1-score of 0.977, achieved by SemanticSoftTFIDF using SBERT and BERT, as can be seen in Table 5.8. The similar scores among the models for the variants indicate that it is not of major importance which of the models that is used, but more important which algorithms are included in the feature vectors.

Table 5.13 presents the average evaluation metrics of the classifiers when using stratified 5-fold cross-validation on the entire data set, on feature vector variant 7. The best performance was achieved by the random forest model, with an F_1-score of 0.986, which is slightly higher than what the performance for variant 5.

The confusion matrices for the random forest classifier using feature vector variant 5, as can be seen in Figure 5.13a, and feature vector variant 7, as can be seen in Figure 5.15a, only differ by two less false negatives in feature vector variant 7. This is a minor improvement and there is little data from which to draw any conclusion, however the results indicate that the random forest does seem to perform slightly better when adding the feature of token length ratio.

6.4.2 Thresholds

The data fed to the machine learning models was the same data used in earlier approaches to determine the similarity, namely different similarity functions. It is interesting to understand why the machine learning models achieved higher results than the other approaches. This could be explained by the fact that for all other approaches, thresholds were used in order to classify the pairs based on the similarity scores, which made their performance highly dependent on the defined threshold. This was not the case for the machine learning models as no pre-defined thresholds were used for the classifications. Instead, the models based their predictions on the raw similarity scores in the feature vectors. Santos et al. [4] suggested that machine learning would improve performance by avoiding manual threshold tuning and therefore avoid having an uncertainty in performance due to the threshold. This is also what we observed from the results as we saw an improvement when not having pre-defined thresholds.

An interesting consideration is that for several of the algorithms used to calculate the similarity scores fed to the models, inner thresholds were defined as part of the algorithm. This was the case in for example SemanticSoftTFIDF, where inner thresholds were used to determine Jaro-Winkler similarity and semantic similarity from the pre-trained word embeddings. These inner thresholds were still manually defined even when used in the machine learning models in order to obtain a similarity score.

It is interesting to consider the results when SemanticSoftTFIDF was not used in the feature vector, but SoftTFIDF, Jaro-Winkler, cosine and BPEmb with cosine were used, corresponding to variant 3 in the results. For this variant, the models have been fed information of semantic similarity, lexical similarity using the baseline functions and SoftTFIDF, which was all the same information that SemanticSoftTFIDF was based on, but not combining them with defined thresholds. The results show that this variant did not perform as well as the variants where the SemanticSoftTFIDF was included. This is an interesting observation as the algorithm consisted of more inner thresholds than the other algorithms.

6.4.3 Feature Importance

For the machine learning models evaluated, the results indicated that the selection of similarity functions included in the feature vector was an important factor for the performance. It is therefore interesting to consider the feature importance for the models, which was evaluated using SHAP values, shown in Figure 5.12 for variant 5 and Figure 5.14 for variant 7. The figures show that for all three models, the hybrid approaches achieve highest SHAP values compared to the other similarity functions in the feature vectors. This indicates that the hybrid approaches works well for the task and provides the models with valuable information. What other algorithms had high SHAP values differed among the classifiers. Interesting to note is that the state-of-the-art similarity functions Jaccard, cosine, Jaro and Levenshtein did not obtain high feature importance in any of the models, while Jaro-Winkler had an average feature importance in all of the models.

None of the features in any of the models had negative SHAP values, meaning that none of the features had a negative impact on the prediction. In relation to SemanticSoftTFIDF using SBERT, the other SHAP values for the other algorithms were relatively even which makes it difficult to draw any conclusions.

Interesting to note is that for feature variant 7, the token length ratio had a very low feature importance. This indicated that adding the token length did not add a major value to the prediction but did show a slight benefit over not being added.

6.4.4 Optimization

Even though it has not been included in the evaluation, it is interesting to consider the optimization of the approaches. Machine learning models can require a lot of time and memory, especially when increasing the amount of data. In these approaches, calculating the similarity functions is time-consuming and requires memory. The evaluation shows an improvement in performance using the machine learning approaches, however the improvement was not significant. There is a trade-off between how important the slight increase in performance is in relation to the time and memory required by the machine learning approaches. Depending on the scalability and objective of the POI name matching, it could be more relevant, for example, to use the SemanticSoftTFIDF instead of the machine learning approach, as it achieves high results but requires a lot less computations.

Feature importance could be used to further optimize the machine learning approaches by removing the similarity functions in the feature vectors that have a low feature importance, as this would require less computations. For example, the results show that the difference between the models' performance on feature vector variant 2, using only four features, was not much worse than feature vector variant 7, which uses nearly all features. While the performance is better with variant 7, it comes at a higher computational cost.

6.5 Evaluation Methods

The main metric used for evaluating the approaches was the F_1-score, as this is considered the standard metric for the performance of classifiers. Above the F_1-score, the metrics precision, recall and MCC were presented. As these are highly correlated to the F_1-score, for most of the results of the approaches we observed that this was also the case. In general, the approaches achieving high F_1-scores also achieved high precision, recall and MCC. For some results we found that it was not always the approach achieving the highest F_1-score that also achieved the highest precision and recall. However, for all approaches we saw that the approach achieving the highest F_1-score also achieved the highest MCC. The highest MCC of 0.983 was achieved by the random forest classifier on variant 7, as can be seen in Table 5.13. The machine learning approaches in general achieved high MCC scores. Furthermore, the SemanticSoftTFIDF achieved a MCC of 0.973 and SoftTFIDF achieved a MCC of 0.971. Since MCC is an accurate metric for an imbalanced data set, it is a positive indication that the approaches that achieved high F_1-scores also achieved high MCC scores. This indicated that the approaches perform well on the data set used in the work, as it was rather imbalanced.

The evaluation was done using two variants of the data set. In general, the relation between the results for the two data sets were quite constant. For the implementations achieving high results on the entire data set, the results were also high for the reduced data set. This was not surprising as exact pairs were easily correctly classified by all investigated approaches.

Depending on the objective of POI name matching, different metrics might be of more interest than others. A common concern when using geographical data from multiple sources is avoiding duplicates, it could therefore be of higher importance to minimize the number of false negatives, and therefore high recall is to prefer.

An aspect observed when evaluating the results was that the closer the evaluation metrics were to a prediction of 1.0, the improvements made to the classification approaches showed relatively small increases in the evaluation metrics. To more easily compare the improvements of the approaches compared to the baseline, we considered the percentage error reduction. Table 5.16 shows improvement in terms of error reduction, where the machine learning approach using random forest achieved the highest reduction of 67%.

Worth noting about the evaluation is the manual analysis of the classified pairs, that was done by attempting to observe trends in what types of POIs were incorrectly classified. This method gave good indications of issues for the classifiers. However, since the data set was rather small and the number of trivial cases classified incorrectly was also rather small, it was difficult to make any assumptions from the observations, since there was no way to determine what was a trend and what was a coincidence.

6.6 Limitations

The data set used for evaluation is relatively small and it would have been beneficial to evaluate the algorithms on a larger data set. Especially when using machine learning, a large data set for training is preferred. Another aspect of the data is the distribution of POI types. Ideally, a broad representation of different types of POIs would be considered, with a distribution corresponding to the real world. Restaurants, shops and cafes might have been over-represented as the Yelp data set mainly contains POI of businesses. For example, this could affect the TFIDF-values by giving lower scores to words related to these types of POIs.

During the labeling process we encountered another limitation, namely pairs that were not possible to determine as a match or non-match. These pairs were labeled as a separate class and not used for the evaluation, which is unfortunate since they clearly highlight the problem of the task. Nevertheless, it is a difficult evaluate a prediction without knowing the correct answer.

As briefly mentioned earlier in this chapter, using frameworks based on the English language can have varying results when working with POIs. The location of a POI does not guarantee the language of the name. In this work, both the stop word collection and the stemming library were based on English, which means that the usage of these tools did not work as expected for non-English names. For example, with another language or additional languages the word "La" in "La Famiglia Spagnuolo's" could have been considered a stop word and therefore removed.

Chapter 7
Conclusions

To conclude the work done in this thesis, we aim to answer the three research questions stated in section 1.3.1.

We found that hybrid approaches can improve performance for POI name matching compared to state-of-the-art similarity functions. Combining similarity functions can compensate for each other's disadvantages by solving different parts of the task. More specifically, our approaches demonstrated several techniques that have a positive impact on the performance: pre-processing, matching approximate tokens and weighting parts of the POI name improved the results.

We found that semantic similarity did not improve the performance of the hybrid approaches investigated in this thesis. The evaluation indicates that semantics can be identified, but the lack of context in the names and the fact that the word embeddings are not suited for POI names makes it difficult to use for this task.

We found indications that machine learning can improve the performance by combining multiple hybrid approaches and avoiding manual tuning of thresholds. Furthermore, the evaluation of the machine learning approaches supports the conclusion that hybrids improve the performance of state-of-the-art similarity functions.

Overall, the investigated approaches perform well for POI name matching, with the best machine learning approach reducing the error by 67% compared to state-of-the-art similarity functions. The evaluation indicates several techniques that could potentially be beneficial if used in more favorable ways, for example the usage of semantics and composition of the corpora, which can be further investigated.

Chapter 8

Future work

In this chapter we propose future improvements of the work done in the thesis.

8.1 Weighted Words

An issue we faced was determining which parts of the names were important when comparing POIs. In the hybrid approaches we investigated how corpora can be used for weighting tokens differently depending on their importance. We found that selecting a corpus based on an area can have a positive impact for POIs located in that area, however making the corpus too small has a negative impact on weighting other tokens. Selecting a large corpus based on all POIs in the data set had a positive impact on weighting business-related tokens but does not completely solve the problem of names having different numbers of tokens. Determining What parts of the names are to be considered more or less important is highly context dependent. How to select and combine corpora based on the context could be further investigated to improve the performance when used in hybrid approaches. It could also be interesting to investigate a deep learning approach to determine the importance of different parts of the name.

8.2 Categorization of POIs

In this thesis, we evaluated similarity of POIs by using only their names and coordinates. It would also be of interest to investigate an approach to categorize the POIs based on types such as restaurants, parks, libraries etc., using machine learning. The categories could perhaps be helpful for determining similarity.

8.3 Fine-tuning Embeddings

When investigating semantics in the hybrid approaches, existing pre-trained embeddings, not trained on POIs or other geographical information, were used. However, it would be interesting to evaluate the performance of a word embedding trained specifically for POI name matching. By making the embeddings more suitable for POIs, they would be more likely to handle task specific problems. Fine-tuned embeddings could possibly perform better and improve the determination of semantics between POI pairs.

8.4 Optimization

For the scope of the thesis, we evaluate the performance of hybrid approaches for POI name matching. In the evaluation of the performance, optimization such as time and memory complexity is not considered. However, these aspects would be relevant when applying the approaches on larger scale and could therefore be of interest to investigate further.

References

[1] J. Kysela, "A comparison of text string similarity algorithms for poi name harmonisation," in *International Conference on Articulated Motion and Deformable Objects*. Springer, 2018, pp. 121–130.

[2] W. W. Cohen, P. Ravikumar, S. E. Fienberg *et al.*, "A comparison of string distance metrics for name-matching tasks," in *IIWeb*, vol. 3. Citeseer, 2003, pp. 73–78.

[3] A. Monge and C. Elkan, "An efficient domain-independent algorithm for detecting approximately duplicate database records." Citeseer, 1997.

[4] R. Santos, P. Murrieta-Flores, and B. Martins, "Learning to combine multiple string similarity metrics for effective toponym matching," *International journal of digital earth*, vol. 11, no. 9, pp. 913–938, 2018.

[5] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE Transactions on knowledge and data engineering*, vol. 19, no. 1, pp. 1–16, 2006.

[6] J. Wang, G. Li, and J. Fe, "Fast-join: An efficient method for fuzzy token matching based string similarity join," in *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 2011, pp. 458–469.

[7] S. Kulkarni, "Jaro winkler vs levenshtein distance," Available at https://srinivas-kulkarni.medium.com/jaro-winkler-vs-levenshtein-distance-2eab21832fd6 (2022-06-12), Medium, March 2021.

[8] W. W. Cohen, P. Ravikumar, and S. Fienberg, "A comparison of string metrics for matching names and records," in *Kdd workshop on data cleaning and object consolidation*, vol. 3, 2003, pp. 73–78.

[9] N. Gali, R. Mariescu-Istodor, and P. Fränti, "Similarity measures for title matching," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 1548–1553.

[10] N. Augsten and M. H. Böhlen, “Similarity joins in relational database systems,” *Synthesis Lectures on Data Management*, vol. 5, no. 5, pp. 1–124, 2013.

[11] T. El-Shishtawy, “A hybrid algorithm for matching arabic names,” *arXiv preprint arXiv:1309.5657*, 2013.

[12] K. Ganesan, “All you need to know about text preprocessing for nlp and machine learning,” Available at http://www.blowinglotsofweirdstuffup.com/guide.html (2022-06-01), KDuggets, April 2019.

[13] K. Jayakodi, M. Bandara, and D. Meedeniya, “An automatic classifier for exam questions with wordnet and cosine similarity,” in *2016 Moratuwa engineering research conference (MERCon)*. IEEE, 2016, pp. 12–17.

[14] S. Kannan, V. Gurusamy, S. Vijayarani, J. Ilamathi, M. Nithya, S. Kannan, and V. Gurusamy, “Preprocessing techniques for text mining,” *International Journal of Computer Science & Communication Networks*, vol. 5, no. 1, pp. 7–16, 2014.

[15] D. M. Porter and R. Boulton, “Snowball,” Available at https://snowballstem.org/ (2022-04-14), 2001-2002.

[16] G. Grefenstette, *Tokenization.* Dordrecht: Springer Netherlands, 1999, pp. 117–133. [Online]. Available: https://doi.org/10.1007/978-94-015-9273-4_9

[17] C. Khanna, “Word, subword, and character-based tokenization: Know the difference,” Available at https://towardsdatascience.com/word-subword-and-character-based-tokenization-know-the-difference-ea0976b64e17 (2022-03-09), Towards Data Science, July 2021.

[18] P. Gage, “A new algorithm for data compression,” *C Users Journal*, vol. 12, no. 2, pp. 23–38, 1994.

[19] R. Sennrich, B. Haddow, and A. Birch, “Neural machine translation of rare words with subword units,” *arXiv preprint arXiv:1508.07909*, 2015.

[20] B. Heinzerling and M. Strube, “Bpemb: Tokenization-free pre-trained subword embeddings in 275 languages,” *arXiv preprint arXiv:1710.02187*, 2017.

[21] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, “Google’s neural machine translation system: Bridging the gap between human and machine translation,” *arXiv preprint arXiv:1609.08144*, 2016.

[22] T. M. Mitchell *et al.*, *Machine learning.* McGraw-hill New York, 1997.

[23] T. Dietterich, “Overfitting and undercomputing in machine learning,” *ACM computing surveys (CSUR)*, vol. 27, no. 3, pp. 326–327, 1995.

[24] “Cross-validation,” Available at https://scikit-learn.org/stable/modules/cross_validation (2022-05-18), Scikit-learn.

[25] V. A. Dev and M. R. Eden, “Formation lithology classification using scalable gradient boosted decision trees,” *Computers & chemical engineering*, vol. 128, pp. 392–404, 2019.

[26] B. P. Roe, H.-J. Yang, J. Zhu, Y. Liu, I. Stancu, and G. McGregor, "Boosted decision trees as an alternative to artificial neural networks for particle identification," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 543, no. 2-3, pp. 577–584, 2005.

[27] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[28] A. Verma and V. Ranga, "On evaluation of network intrusion detection systems: Statistical analysis of cidds-001 dataset using machine learning techniques," *Pertanika Journal of Science & Technology*, vol. 26, no. 3, pp. 1307–1332, 2018.

[29] S. Yildirim, "Gradient boosted decision trees-explained," Available at https://towardsdatascience.com/gradient-boosted-decision-trees-explained-9259bd8205af (2022-06-12), Feb 2020.

[30] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[31] H. Ramchoun, Y. Ghanou, M. Ettaouil, and M. A. Janati Idrissi, "Multilayer perceptron: Architecture optimization and training," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 4, no. 1, pp. 26–30, 2016.

[32] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[33] A. E. Roth, *The Shapley value: essays in honor of Lloyd S. Shapley*. Cambridge University Press, 1988.

[34] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in neural information processing systems*, vol. 30, 2017.

[35] D. Jurafsky and J. Martin, "Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition." Prentice Hall, 2000, vol. 2.

[36] "Embeddings: Translating to a lower-dimensional space," Available at https://developers.google.com/machine-learning/crash-course/embeddings/translating-to-a-lower-dimensional-space (2022-06-12), Google Courses, Feb 2022.

[37] J. Zhao, S. Mudgal, and Y. Liang, "Generalizing word embeddings using bag of subwords," *arXiv preprint arXiv:1809.04259*, 2018.

[38] P. Huilgol, "Quick introduction to bag-of-words (bow) and tf-idf for creating features from text," Available at https://www.analyticsvidhya.com/blog/2020/02/quick-introduction-bag-of-words-bow-tf-idf (2022-06-12), Analytics Vidhya, Feb 2020.

[39] S. Tata and J. M. Patel, "Estimating the selectivity of tf-idf based cosine similarity predicates," *ACM Sigmod Record*, vol. 36, no. 2, pp. 7–12, 2007.

[40] C. S. Unnikrishnan, “How sklearn’s tfidfvectorizer calculates tf-idf values,” Available at https://www.analyticsvidhya.com/blog/2021/11/how-sklearns-tfidfvectorizer-calculates-tf-idf-values/ (2022-04-13), Analytics Vidhya, Nov 2021.

[41] A. R. Lahitani, A. E. Permanasari, and N. A. Setiawan, “Cosine similarity to determine similarity measure: Study case in online essay assessment,” in *2016 4th International Conference on Cyber and IT Service Management*. IEEE, 2016, pp. 1–6.

[42] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation,” in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[43] P. Wenig, “Creation of sentence embeddings based on topical word representations,” Available at https://towardsdatascience.com/creation-of-sentence-embeddings-based-on-topical-word-representations-d325d50f99e (2022-04-13), Towards Data Science, 01 2019.

[44] S. Theiler, “Basics of using pre-trained glove vectors in python,” Available at https://medium.com/analytics-vidhya/basics-of-using-pre-trained-glove-vectors-in-python-d38905f356db (2022-04-13), Medium, Sep 2019.

[45] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.

[46] “Bert overview,” Available at https://huggingface.co/docs/transformers/model_doc/bert (2022-04-13), Hugging Face.

[47] L. McQuillan, “Word2vec vs bert,” Available at https://www.saltdatalabs.com/blog/word2vec-vs-bert (2022-06-12), Salt Data Labs, Mar 2022.

[48] N. Reimers and I. Gurevych, “Sentence-bert: Sentence embeddings using siamese bert-networks,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. [Online]. Available: https://arxiv.org/abs/1908.10084

[49] “Confusion matrix,” Available at https://en.wikipedia.org/wiki/Confusion_matrix (2022-03-09), Wikipedia, March 2022.

[50] A. Kulkarni, D. Chong, and F. A. Batarseh, “5 - foundations of data imbalance and solutions for a data democracy,” in *Data Democracy*, F. A. Batarseh and R. Yang, Eds. Academic Press, 2020, pp. 83–106. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128183663000058

[51] S. Narkhede, “Understanding confusion matrix,” Available at https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62 (2022-03-09), Towards Data Science, May 2018.

[52] J. Korstanje, “The f1 score,” Available at https://towardsdatascience.com/the-f1-score-bec2bbc38aa6 (2022-03-09), Towards Data Science, August 2021.

[53] J. Wang, G. Li, J. X. Yu, and J. Feng, "Entity matching: How similar is similar," *Proceedings of the VLDB Endowment*, vol. 4, no. 10, pp. 622–633, 2011.

[54] "Phi coefficient," Available at https://en.wikipedia.org/wiki/Phi_coefficient (2022-03-09), Wikipedia, March 2022.

[55] "Matthew correlation coefficient," Available at https://scikit-learn.org/stable/modules/generated/sklearn.metrics.matthews_corrcoef.htmlt (2022-03-09), Scikit-learn, March 2022.

[56] Z. Bosnjak, O. Grljevic, and S. Bošnjak, "Crisp-dm as a framework for discovering knowledge in small and medium sized enterprises' data," 06 2009, pp. 509 – 514.

[57] "Crisp-dm process diagram," Available at https://sv.wikipedia.org/wiki/Fil:CRISP-DM_Process_Diagram.png (2022-04-04), Wikimedia Commons, April 2012.

[58] "Openstreetmap," Available at https://www.openstreetmap.org/about (2022-03-09), 2022.

[59] "Yelp," Avaialable at https://www.openstreetmap.org/about (2022-03-09), 2022.

[60] "data.gov," https://data.gov/about/ (Accessed 2022-03-30), 2022.

[61] "City of vancouver data set," Available at https://opendata.vancouver.ca/pages/home/ (2022-03-30), 2022.

[62] G. GmbH and O. Contributors, "Openstreetmap data extracts," Available at https://download.geofabrik.de/ (2022-02-16), 2018.

[63] Y. Inc., "Download yelp dataset," Available at https://www.yelp.com/dataset/download (2022-02-16, 2004–2022.

[64] Data.gov, "Community points of interest," Available at https://catalog.data.gov/dataset/community-points-of-interest (2022-03-25), 2022.

[65] "data.gov, community points of interest," Available at https://catalog.data.gov/dataset/community-points-of-interest (2022-03-30), 2022.

[66] C. of Vancouver, "Cultural spaces," Available at https://opendata.vancouver.ca/explore/dataset/cultural-spaces/information/?disjunctive.type&disjunctive.primary_use&disjunctive.ownership (2022-03-25), 2019.

[67] ——, "Schools," Available at https://opendata.vancouver.ca/explore/dataset/schools/information/ (2022-03-25), 2019.

[68] ——, "Libraries," Available at https://opendata.vancouver.ca/explore/dataset/libraries/information/ (2022-03-25), 2019.

[69] "Nltk documentation," Available at https://www.nltk.org/ (2022-04-27), NLTK Project.

[70] T. Šolc and S. M. Burke, "Unidecode project description," Available at https://pypi.org/project/Unidecode/ (2022-04-27).

[71] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[72] "Hugging face," Available at https://huggingface.co/ (2022-05-28), 2022.

[73] "Bert base model (uncased)," Available at https://huggingface.co/bert-base-uncased (2022-05-18), Hugging Face.

[74] "all-mpnet-base-v2," Available at https://huggingface.co/sentence-transformers/all-mpnet-base-v2 (2022-05-18), Hugging Face.

[75] S. Lundberg, "Shap," Available at https://shap.readthedocs.io/ (2022-05-30), 2018.

[76] "Xgboost," Available at https://xgboost.ai/ (2022-05-18).

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2022-06-20

EXAMENSARBETE Investigating Hybrid Approaches for Name Matching of Points of Interest
STUDENT Lucy Albinsson, Tove Sölve
HANDLEDARE Dennis Medved (LTH), Hampus Londögård (AFRY)
EXAMINATOR Jacek Malec (LTH)

Ser kartan dubbelt?

POPULÄRVETENSKAPLIG SAMMANFATTNING **Lucy Albinsson, Tove Sölve**

För att karttjänster ska vara pålitliga är det viktigt att kunna avgöra om två närliggande intressepunkter med liknande namn är samma plats eller inte. Examensarbetet undersöker metoder för att avgöra detta genom att titta på olika aspekter av likheter mellan namnen.

Karttjänster har blivit en viktig del för att underlätta människors vardag. För att kartan ska vara användbar och representera den verkliga världen krävs det stora mängder geografisk information. Därför används ofta flera olika källor och att kombinera dessa kan leda till utmaningar. Tänk dig att du är på väg till universitetssjukhuset i Lund och tar fram din karttjänst för att hitta till rätt avdelning. Kartan visar två markörer som båda verkar representera avdelningen du letar efter, trots att de har något olika namn och ligger i olika delar av byggnaden. Att karttjänsten visar dubbletter av en och samma plats, är ett av problemen som kan uppstå när information hämtas från flera källor. Det finns ingen försäkran om att en plats har samma namn och geografisk information i olika databaser. Likaså är det svårt att avgöra om två närliggande platser med liknande namn faktiskt är olika, vilket gör det svårt att slå ihop rätt information.

Cancer Institute of Florida
Florida Hospital Cancer Institute

I vårt examensarbete har vi undersökt om det går att avgöra om två närliggande platser är samma eller inte utifrån deras namn och geografiska position. Vi undersöker två aspekter av likhet mellan text, dels struktur och förekomst av tecken och dels semantisk betydelse. Våra metoder är baserade på kombinationer av traditionella algoritmer, vektorrepresentationer av semantisk likhet och maskininlärning.

Resultaten visar att det är positivt att kombinera algoritmer då de kan täcka upp för varandras brister. Dock visade det sig svårt att dra nytta av den semantiska likhethet mellan namnen eftersom att det ofta saknas sammanhang i namn. Maskininlärningsmetoderna var baserade på flera av de kombinerade algoritmerna och visade sig ge bäst resultat. Överlag visade resultaten att det kan vara positivt att kombinera metoder som tar hänsyn till olika aspekter av likhet för att avgöra om två intressepunkter representerar samma plats.