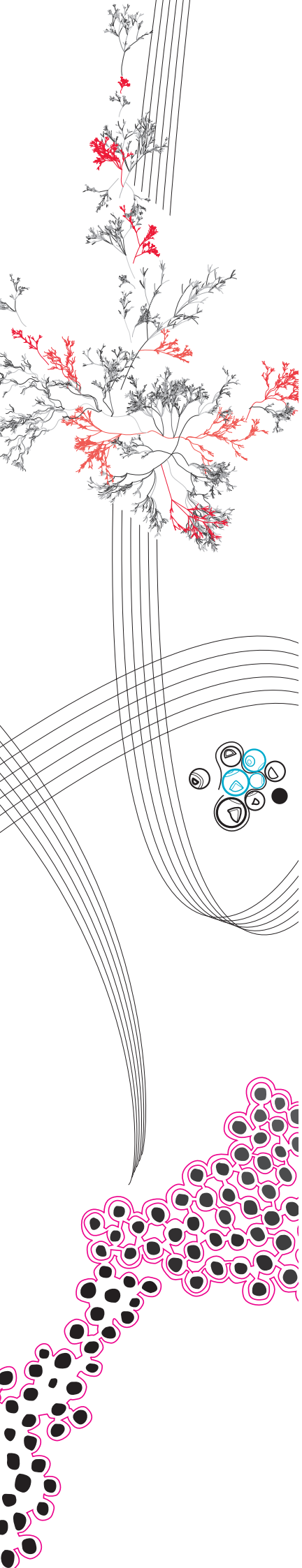


MASTER THESIS

Reducing labeled data usage in duplicate detection using deep belief networks

Stefan C. Janssen
s.c.janssen@student.utwente.nl

Faculty of Electrical Engineering, Mathematics and Computer Science
Human Media Interaction Research Group

Committee:
dr. Mannes Poel
dr.ing. Gwenn Englebienne
drs. Manuel van den Berg

July 1, 2016
Version 1.00

UNIVERSITY OF TWENTE.

Abstract

Modern duplicate detection systems typically use supervised machine learning algorithms to create duplicate detection models. These algorithms require a large amount of manually labeled data to train on. Using semi-supervised deep learning techniques would allow the training to use not only labeled data, but also unlabeled data, which is easily available. The expectation is that this will allow models with less manually labeled data to achieve similar or better accuracy as traditional supervised algorithms.

To determine the effect of semi-supervised learning, we created several variations of existing duplicate detection algorithms, all focused on incorporating semi-supervised learning. For the matching phase this resulted in a deep belief network instead of a traditional neural network as classifier. Additional variants were also created with word embedding and word hashing as extra features, as both are new developments in natural language processing and often used in combination with deep learning. For the indexing phase semantic hashing was used as a way to encode these additional features.

All these algorithms have been evaluated against several traditional duplicate detection algorithms on two different datasets. Both datasets are samples of real-life datasets and contain contact data. A separate dataset was used for tuning. The different algorithms, covering both the indexing and matching steps in duplicate detection, are compared using a 5x2 K-fold evaluation to determine their performance on unseen data.

The results for the matching step show significant differences between supervised and semi-supervised in only one scenario (improving the F1-score from 0.806 to 0.821). There are no significant differences for the variants which combine semi-supervised learning with the additional features. For the indexing algorithms semi-supervised training does improve the results significantly in several cases (e.g. improving the F1-score from 0.702 to 0.710 in one of cases).

The required modifications in structure have been evaluated separately. The additional auto-encoder layer in matching improves the quality on one of the datasets (e.g. improving the F1-score from 0.919 to 0.935 in one of the cases). For the indexing algorithms the additional semantic hashing reduces the quality (e.g. from F1-score of 0.720 to 0.702 in one case).

The addition of extra features, word embedding and word hashing, improve the results in matching on one of the datasets, but only for the larger training sizes (e.g. improving the F1-score from 0.872 to 0.899 in one instance). For the indexing algorithms the additional features also improved the results (e.g. from a 0.705 F1-score to 0.720 in one case). In both cases there was no differences between newly developed features, such as word embedding and word hashing, and bag-of-words, the additional feature baseline.

Contents

List of Tables

1 Introduction

Modern organizations maintain large databases as a result of their core business. A key part of the data inside these databases, especially in the service industry, will consist of contact data. Organizations have to track the contact information of their suppliers, employees and customers. Over many years, these databases have gathered many duplicate entries. For example, internal departments might move to a single database system and organizations can be bought or merged. Finding and solving these duplicate entries helps organizations to be both more effective and more efficient. However, finding duplicate entries is not always straightforward. Because of the fuzzy nature of contact data, duplicate entries of the same contact will likely differ slightly. Both names and addresses are subject to many variations: one entry might contain an initial and the other the entire given name. Phone numbers and email addresses are often more easily standardized, but might not always be present.

While the problem of *duplicate detection*[1] is not always straightforward, there are numerous techniques that give good results. A typical approach is to calculate a number of string distance measures between the different fields of the two records and use these distances to determine whether two records are duplicate or not. Historically, doing so involved domain experts skillfully creating matching rules, but today it is more common to derive these rules using machine learning techniques.

Typically the machine learning algorithms used to create these matching rules are supervised algorithms, meaning that they require annotated data to train on. Models created by supervised algorithms have the potential of producing high-quality results, but large sets of labeled data are required to do so. The creation of labeled data is time consuming as a domain expert has to manually classify pairs as either duplicate or non-duplicate.

A potential solution to reduce the dependency on labeled data is provided by recent advances in deep learning. In semi-supervised learning, such as certain deep learning techniques, not only the

[1] The problem of duplicate detection is also referred to as record linkage, entity resolution, merge/purge or identity resolution.

labeled data is used for training, but also the unlabeled data. An example of a semi-supervised deep learning technique is a deep belief network.

Using unlabeled data in addition to labeled has the potential to achieve comparable results with less labeled data, as large amounts of unlabeled data take the place of some of the labeled data. Replacing the role of labeled data with unlabeled data will reduce the human effort required to train models: because in contrast to labeled data, unlabeled data is easily available in the form of the dataset to be processed.

1.1 Research questions

The goal of the proposed research is to investigate the effects of deep learning, and specifically deep belief networks, when applied to the problem of duplicate detection. Being semi-supervised, deep belief networks can make use of both labeled and unlabeled data. The expectation is that this allows for more advanced models than traditional machine learning techniques, which can only make use of labeled data. The following research questions will be addressed:

Research question 1 *How does the use of unlabeled data in addition to labeled data influence duplicate detection quality?*

The main question that this research aims to answer is about the influence of unlabeled data in addition to the labeled data. To make semi-supervised training possible, a belief network is used for matching. Using unlabeled data is expected to improve the performance of the model and increase the quality of the matching results, especially as the amount of labeled data gets lower.

Research question 2 *How does the use of layers such as auto-encoders to create a belief network influence duplicate detection quality?*

While deep belief networks offer the possibility of training networks in a semi-supervised way, they make certain requirements on the structure of the neural network, for example the use of auto-encoder layers. By evaluating these changes separately, it becomes possible to distinguish between the effect of the added unlabeled data and the required structural changes.

Research question 3 *How does the use of other techniques commonly used in natural language processing with deep learning, such as word embedding and semantic hashing, influence duplicate detection quality?*

The use of semi-supervised learning with deep belief networks is not the only development within deep learning. A lot of other new developments within deep learning and natural language processing also have the potential to be useful within the context of duplicate detection, where they can be used as additional features for the used algorithms.

2

Background

The algorithms described in this paper employ a number of machine learning techniques. This chapter aims to give a short overview of some of the basic techniques and concepts in machine learning. While these techniques are described in general terms, the focus is on those relevant to this research. Those who are already familiar with these techniques could consider skipping this chapter and going to Chapter 3, where related work is discussed in relation with this research.

2.1 Machine learning

Machine learning is the study of algorithms that can learn from and make predictions on data. These algorithms build models from examples, rather than following strictly static instructions. Not only are several of these models capable of higher accuracy than manually created models, they are also capable of adapting to new circumstances without intervention as both model creation and evaluation can be automated. It is therefore not surprising to see machine learning algorithms being adopted in a variety of scenarios.

There are a lot of different machine learning algorithms and they are typically placed in several different categories, based on which kind of problems they are suitable for. For example, machine learning algorithms are categorized as either supervised or unsupervised, depending on whether or not the algorithm requires the data to be annotated with a target value. Although uncommon, there are also algorithms that strictly do not fall under either supervised or unsupervised learning: semi-supervised algorithms, for example, are capable of using both labeled and unlabeled data.

Within supervised learning, a distinction is usually made between classification and regression. In classification the target value is one of a fixed set of values. This type of value is more commonly known as an enumeration in other fields of computer science. For example, an email filtering system would classify new mails as either spam or non-spam. In regression the target value is

numeric. Classification and regression machine learning algorithms are referred to as respectively classifiers or regressors.

2.1.1 Evaluation of classifiers

The evaluation of different classifiers plays a large role in the application of machine learning. During the evaluation, different classifiers are compared to each other and to trivial baselines. An evaluation of a number of classifiers consists of several key elements. The different classifiers are trained on the data and the resulting models are in turn applied to the data. Finally the output is compared to the golden standard and the results of the comparison are summarized using some metric, for example error count. The results of the comparison give insight in which classifiers are better suited to a certain problems, and if a classifier is an improvement over baseline solutions at all.

A typical evaluation is more complicated than simply training on the whole dataset and looking at the results. This kind of evaluation is known as the *training set evaluation*, as the model is evaluated on the same data it is trained on. The training set evaluation is not typically used to compare classifiers against each other. The most important reason is that training set evaluation does not properly reflect the generalization error of a classifier, or how a classifier performs on unseen data.

The limitations of the training set evaluation can be illustrated by the nearest neighbour classifier. Unlike most classifiers, this classifier does not create a model of the input upon training. The entire training set is simply stored in memory. When classifying, the nearest neighbour classifier [8] looks for the training sample nearest to the input, and outputs the class of that sample. During a training set evaluation, the training sample closest to the input is simply the input itself, resulting in the illusion of a zero error rate[1].

Error metrics

In practice errors are not directly combined into a error metric. The results are first summarized in a *confusion matrix*. A confusion matrix is a matrix in which the rows represent the true classes and the columns represent the classes predicted by the model. In case of only two classes, the confusion matrix is a two-by-two matrix. Each cell in the matrix contains the number of samples that belong to a certain class and have been classified as a certain class. This includes samples that have been classified as the actual class.

Another difference in practice is the used error metric. The error rate or related metrics such as the percentage of samples classified correctly are problematic in certain scenarios, for example when the distribution between the classes is skewed. In the case of duplicate detection, classifying pairs as either duplicate or non-duplicate, the odds are heavily skewed towards two random records being non-duplicate. In such a scenario, a classifier that always returns the majority class will have a very low error rate.

In practice it is relevant to find instances of the rare class between many cases of the common class. In these cases other evaluation metrics, such as precision and recall, are typically used. Precision is defined as the part of retrieved elements that is relevant, while recall is the part of

[1] In case the training set contains multiple entries that are completely equals except for the target value, the nearest neighbour classifier will not have an zero error rate when evaluated in a training evaluation.

the relevant elements that have been retrieved. As most problems require a balance between both precision and recall, they are commonly combined in the F1-score.

Another error metric that is commonly used is the *Matthews correlation coefficient* (MCC) [37]. Like precision and recall, the MCC takes the distribution of the classes into account. An advantage of MCC over these other metrics is that MCC uses the number of true negatives, which precision and recall do not. The MCC also does not differentiate between a desirable class and a undesirable class. When using precision and recall selecting a certain class as desirable could create a bias [45]. Similar to the F1-score, the MCC is the combination of two error metrics known as informedness and markedness.

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.1}$$

K-fold evaluation

A training set evaluation does give not give good insight in the generalization error, but there are other evaluation techniques that do. In general, these evaluation techniques focus on evaluating the classifier on data different from the data that it is trained on. A common technique that does this is called a K-fold evaluation [54].

In a K-fold evaluation the available data is split in K parts. Each of the K parts is referred to as a fold, hence the name K-fold. In each round of the evaluation, one of the folds is set aside for evaluation. The algorithms are trained on the other folds and then evaluated on the fold that has been set aside for evaluation. This process is repeated so that each fold is used in evaluation once.

2.1.2 Genetic algorithms

Genetic algorithms are a flexible machine learning technique inspired by natural selection. Initially a population of random candidates is created. Every cycle of the genetic algorithm all candidates in this population are evaluated using some metric. A selection is then made of the higher scoring candidates, from which the population for the next generation is generated. There are different ways for generating this next generation: existing candidates can be mutated, two candidates can be combined using cross-over and a small number of new random candidates can be introduced. After a number of cycles the population will start to consist of better performing candidates than the initial population. Eventually the best performing candidate is picked from the population.

If the metric used to evaluate the population is static, the genetic algorithm will result in a solution that is optimized for the used metric, but performs worse on closely related evaluations. For example, if the used metric evaluates on basis of a set of labeled training samples, the solution might have good results on this set, but generalizes badly to new data. To prevent overfitting in the genetic algorithm, a common approach is to only use part of the labeled training set in the evaluation each generation [20]. This way all candidates in the same generation are evaluated on the same training data selection, to ensure a fair comparison, but the selection changes with every generation.

One of the things that sets genetic algorithms apart from other techniques is their capability to work with different structures. While other classifiers might be more efficient, genetic algorithms can also be used to optimize structures for which there are no specialized algorithms.

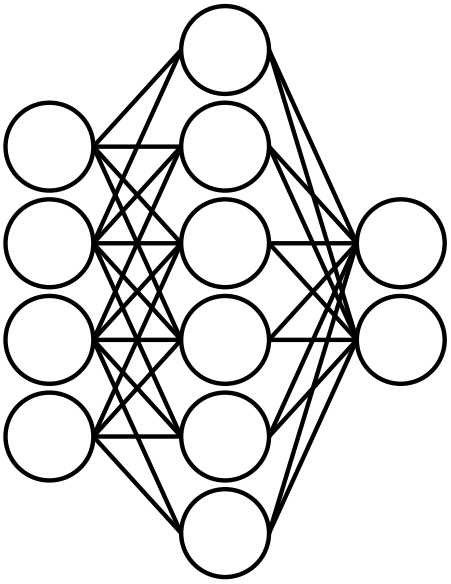


Figure 2.1: Neural networks are networks of neurons, which are arranged in layers. Input values are propagated through the network.

2.1.3 Neural networks

Neural networks are a type of classifiers based loosely on the working of the human neural system. A neural network (see Figure 2.1) is a network of neurons, which are typically arranged in layers. With exception of the input layer, each layer of neurons takes the output of the previous layer and processes it further. During the processing, each neuron calculates the weighted sum of outputs of the previous layer and transforms it using an activation function. Typically the weighted sum also includes a bias value, which is added regardless of input values. Although each neuron in a layer uses the same input, the weights are different for the different neurons, resulting in different outputs.

There are a wide variety of neural network configurations, with either a different network structure or different activation functions. It is also possible to combine different activation functions in the same network. For example, when performing regression, it is common to replace the activation function of the output layer with the identity function to create a simple weighted sum. A single neuron with the identity activation function is conceptually identical to a linear regression model.

2.2 Deep learning

Deep learning is the term used for algorithms used within machine learning that focus on learning higher level abstractions. By learning higher level abstractions, these models tend to perform significantly better on a number of tasks that traditionally were considered difficult in machine learning, such as image recognition.

The classic approach to machine learning problems is to construct a set of features manually and then train a machine learning algorithm on that. For example, in image recognition features

might be edge detection, some kind of shape indication, color information and some description of the texture. Because not every set of features is equally good, a lot of time in machine learning is spent on optimizing the features to train on. As deep learning is capable of learning higher level abstractions, these features do not need to be constructed manually, but can instead be learned.

By learning higher level features there are a number of advantages. First of all, automatically learning features saves the time that would otherwise go into designing these features. Secondly, features created by deep learning are often better than hand-made features. There are a number of underlying reasons for this: errors in the network can be back-propagated into the features, and features learned this way are specific to the problem at hand.

2.2.1 Deep neural networks

A deep forward neural network is the deep variant of the common shallow feedforward neural network. A deep forward neural network consists of an input layer, multiple hidden layers and an output layer. Connections only exist in one direction: from the input layers through the hidden layers to the output layer.

Where shallow neural network rely mostly on backpropagation for training, in networks with more layers there are a number of problems that result in only using backpropagation not being practical [19]. A common solution is to pretrain the layers in the network. While there are different ways of pretraining, a common technique is to treat all layers as either auto-encoders or restricted Boltzmann machines. The layers can then be trained one layer at the time. This process avoids the problems that occur when using only backpropagation on a large number of layers. Networks of this type are called deep belief networks. After the pretraining the full network is further optimized using backpropagation.

In addition to pretraining there are a number of other techniques that intend to make very large deep neural networks more feasible. Most of these are not only applicable to feedforward neural networks, but also to the other variants.

One of the noticeable changes is the use of a different activation function. Where traditional neural networks often employed sigmoid or tanh function, recently the use of the *rectified linear unit* (ReLU) [43] has become more popular [34]. The ReLU function, $f(x) = max(0, x)$, returns the input value, except when this input value goes below some threshold, in which case the threshold is returned. While a relatively simple function, it is capable of providing the same kind of non-linearity as the other activation functions. Due to its simplicity, the ReLU function is cheap to compute. As an additional advantage, it also creates a lot of zero values, which simplify further calculation even further.

Another new activation function is the *maxout* function [21]. The output of the maxout function is defined as the maximum values of all weighted input values. Maxout networks can be used to represent any kind of function. The advantages of maxout are similar to those of the ReLU function.

Deep belief networks

Deep belief networks are a variant of deep neural networks that are based on stacks of either auto-encoders or restricted Boltzmann machines. The name is deep belief network is a reference to

belief networks, also known as Bayesian networks. Like Bayesian networks, deep belief networks model dependencies among the input variables.

The important difference between deep belief networks and normal deep neural networks is the way the training is done. Deep belief networks are pre-trained in an unsupervised fashion on unlabeled data to create a model of the dependencies among the input variables. This is done by training each layer on top of the underlying layer, with each layer improving the overall representation of the data. The higher level representation can then be used as the input to a output layer, like those in neural networks.

After the pretraining, deep belief networks are further trained using the same supervised algorithms used for deep neural networks. Both approaches can also be combined in a semi-supervised approach, by first training on unlabeled data and then finalizing using labeled data. This allows for networks that are based on both labeled and unlabeled data. Aside from the avoiding the problems normally related with training a large network using backpropagation, the ability of semi-supervised learning is another advantage of deep belief networks over regular deep neural networks.

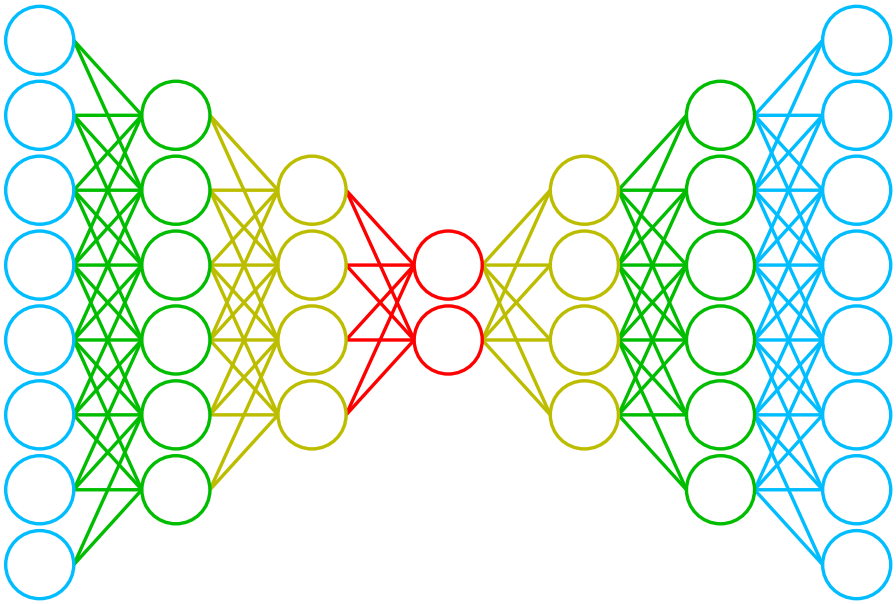


Figure 2.2: A deep auto-encoder is a deep neural network consisting of a number of layers. Characteristic of auto-encoders is the size of the layers: the layer sizes are symmetrical and the closer to the middle, the smaller the layers get.

2.2.2 Auto-encoder

An auto-encoder (see Figure 2.2) is a neural network that aims to transform the input to a lower dimension and back again with as little loss as possible. The goal of such a network is to force the network to learn which information is redundant and can be left out, creating a more compact representation. This compact representation is the basis of techniques such as semantic hashing.

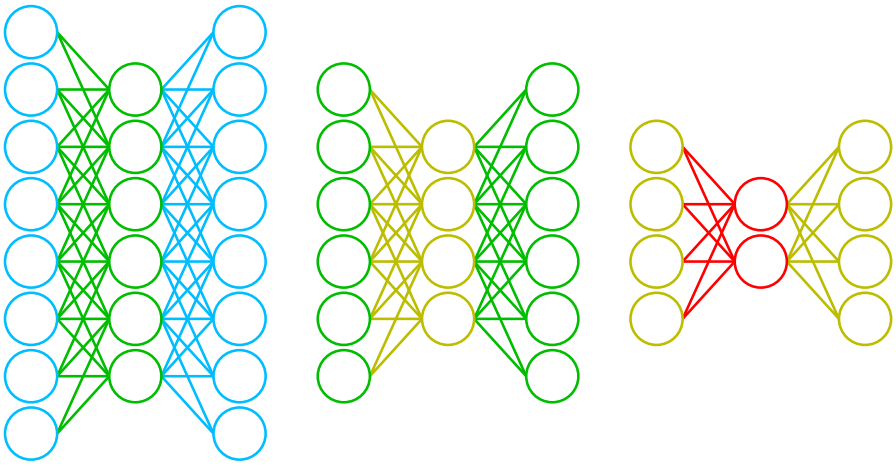


Figure 2.3: A deep auto-encoder can be efficiently trained layer-by-layer by viewing every layer as a small auto-encoder. For the auto-encoder in Figure 2.2, this process combines down to training the above three networks, each time using the previous layers as preprocessing for the next.

In addition, auto-encoders do not require labeled data and are therefore an unsupervised machine learning algorithm. The same data is used as both input and output.

Auto-encoders can occur in both shallow variants, with one hidden layer, and deep variants, with more than one hidden layer. Training of a deep auto-encoder only through the use of backpropagation results in similar problems as with regular deep neural networks. For this reason deep auto-encoders are typically deep belief networks and use the corresponding pretraining technique.

One of the ways to pretrain a deep auto-encoder, and deep belief networks in general, is to view each layer as a shallow auto-encoder. This shallow auto-encoder can then be training using normal techniques. An example of this process is shown in Figure 2.3.

Semantic hashing

Semantic hashing [48] is a technique used to create an index of all documents in such a way that similar documents can easily be retrieved. In semantic hashing the compact code of an auto-encoder network is used for this purpose. Ideally all similar documents have the same compact code. During the training of the auto-encoder network a large amount of noise is added. After training, the compact code is created from the activations in the bottleneck layer of the auto-encoder network. This compact code is converted to binary by applying a threshold. Notice that in [48] the threshold used is close to zero, due to most of the values being closer to zero than one.

2.2.3 Applications

Due to its popularity, deep learning has been applied to many different research areas. Several of them are listed here with the intention to give an idea of the use cases of deep learning. In many cases the application of deep learning to these problems is still an active area of research.

Speech recognition

Speech recognition is the area of research in which the aim is to understand spoken language. A subtask of spoken language understanding is transforming speech to text in an automated fashion. A typical application of speech recognition is in voice controlled user interfaces, for which speech recognition forms an essential part. For a long time, the best performing model for this problem was the hidden Markov model [12]. Several comparisons with traditional neural networks have been made, but the results were not as good as the hidden Markov model [42]. The results of the hidden Markov model were however improved upon with the application of deep learning. Early results using a deep neural network show already substantial improvements [9]. Later models use deep recurrent neural networks in combination long short-term memory units to improve performance even further [22].

Speech synthesis

Speech synthesis is the area of generating speech from text, giving roughly the inverse operation of speech recognition. As with speech recognition, speech synthesis has traditionally been done using hidden Markov models, as well as a number of other techniques. Several deep learning based models have been shown to improve the generated speech. For example, by using a deep belief network as a generative model for speech [28, 36]. In [56] a discriminative deep neural network is used instead, trained to predict the probably distribution normally produced by the hidden Markov model. In [16] a deep neural network is first used to create a compressed version of the generated speech. The compressed speech is then used in second stage to generate final output.

Image recognition

Image recognition is a wider area of research, consisting recognizing objects in images. Early work on deep learning already looked into replacing the previously commonly used principal components analysis with a deep auto-encoder [25]. Deep learning has also been used to improve upon state of the art in different image benchmarks, for example [47] using large convolutional neural networks [31]. Another possibility is the use of a large amount of sparse auto-encoders to allow semi-supervised training [32].

Image recognition has also been used in multi-modal systems. A multi-model system is one where data comes multiple input channels. An example would be converting an image to tags [53]. This is done by creating a deep restricted Boltzmann machine that combines input from both images and text, after two sets several layers solely focused on only one but not the other. More recent works are capable of generating full sentence captions for a given image [55].

Information retrieval

Information retrieval is the area research focused on finding relevant documents from a large corpus given a query. Early work already showed the possibility to use of unsupervised deep belief networks to create compact hash codes for the different documents [24, 48]. These codes can then be used to retrieve similar documents easily. The results of this completely unsupervised system are comparable to that of existing keyword based systems. In [26] a similar process is

done using a deep neural network, but with the use labeled data. A deep stacking network is used in [11], where it is trained in such a way to optimize the ranking of suggested documents.

Natural language processing

Natural language processing is concerned with the processing of natural language. Typical tasks within natural language processing include parsing, part-of-speech tagging, but also higher level problems like machine translation. In [6] a convolutional deep belief network is used to performing many tasks that are traditionally done by different algorithms, such as tagging and chunking. In [5] parsing is done using a deep recurrent convolutional architecture. In [7] a minimal set of predefined features is used, in an attempt to create a model that is as language independent as possible. The internal representation is learned from unlabeled data.

In [51] a recursive neural network is used to create a nested tree structure which can be used for parsing. In [52] a similar nested structure is used for sentiment analysis.

One of the major recent improvements in natural language processing is the use of *word embedding* [6, 7]. Word embedding is a technique in which every word has a corresponding vector it can be mapped to. This vector mapping is constructed in such a way that it aims to represent the context of a given word. The result is that words that occur in similar contexts have very similar vector representations. This is useful because the context of a word contains a lot of information about a given word and its meaning that cannot be extracted from the word itself. For example, the words 'cat' and 'dog' do not have a single letter in common, but both represent animals that are four-legged, pets, etc. Word embedding models can be trained using a number of different algorithms, including neural networks.

Word embedding has been applied to many languages. For use in machine translation word embedding has also been extended to two languages at the same time [57]. This requires additional restrictions to ensure that the word embedding mappings in both languages are compatible. In [18] two language word embedding is also used, augmented with a deep neural network.

Word embedding is essentially a language modeling technique. Language modeling aims to create a statistical probability distribution over words in a text. Historically language modeling has been based on neural networks [2], while more recent work also investigates the possibilities of recurrent neural networks [39].

3
Related Work

The different algorithms used in the evaluation described in Chapter 4 are not new, but originated in related works of research. This chapter aims to give a description of those algorithms. In contrast to Chapter 2, which explains the basic concepts, the algorithms described here that are either used in the evaluation, or closely related to those used in the evaluation.

3.1 Duplicate detection

Duplicate detection is not a new area of research. A lot of previous works, both academic and commercial, investigated different techniques to improve the quality and performance of a duplicate detection system.

The core step in every duplicate detection system is to compare records against each other to determine if they are duplicate records or not. This matching of records is usually in a pair-wise fashion. By comparing the records pair-wise, the pair and the differences can be evaluated in depth. If two records are determined to be duplicate records, they are typically given a score and stored for further reference.

Because of the pair-wise comparison, this matching approach grows quadratic with the number of records. As a result, naively comparing every record with every other record is very time costly on larger datasets. Most duplicate systems therefore use an indexing step. This indexing step aims to reduce the number of evaluations, allowing duplicate detection to remain practical on larger datasets.

Notice that in some situations either the indexing or matching step is left out. If indexing manages to produce high quality results, or the quality requirements are not too high, only using an indexing step will be sufficient. Conceptually this is identical to having a matching phase that classifies all pairs as duplicate. Likewise, on some small datasets indexing might not be needed. The matching phase will then compare every record to every other record.

3.1.1 Matching

In the matching phase records are compared in a pair-wise fashion. This allows for an advanced comparison between two records. This is essentially a classification problem: a pair has to be classified as either duplicate or non-duplicate based on the contents of the records. The approach is typically to calculate a set of features between the fields of the two records. This approach is used not only common in machine learning based systems, but also in systems in which domain experts provide the logic for determining the outcome.

Whereas models configured by domain experts usually stick to more easily understandable models such as decision trees and linear regression, models trained by machine learning can be more complex. Examples of such more complex algorithms are neural networks and support vector machines. In practice the difference in accuracy between different classifiers is relatively small given the same features (5-6% error rate on one dataset, 13-14% on another) [30]. As the different algorithms are functionally interchangeable, some frameworks also allow for the classifier to be changed through configuration [29].

Matching features

The matching phase employs a number of features to determine if two records refer to the same entity. This is more practical than trying to work on the string values directly. As matching is done in a pair-wise comparison, the features are comparisons between the different fields of the two records. Every of the features described below will typically be applied to each field.

Basic features include equals checks or empty checks. String edit distances are also commonly used as features, as they are capable of handling small errors. With string edit distances the output is a measurement of similarity for the two strings. There are many different string edit distances that have been proposed over the years, the list below includes only a small selection. In practice matching algorithms do not need features for many edit distances, as there is a large overlap between the types of string differences covered by the different edit distances.

- The Jaccard index is used mostly in statistics for comparing similarity and diversity of two sets, calculated by dividing the size of the intersection of two sets by the size of the union of these sets. For string comparison this means that both words will be seen as two sets of letters and the order of these letters isn't taken in consideration. This metric is easy to calculate and mostly strong in identifying two completely different strings, meaning two words that don't have many of the same letters in them [23].
- The Jaro distance is a measure of similarity between strings, mostly used for duplicate detection, which scores between 0 (no similarity) and 1 (exact match), intended primarily for short strings. It is based on the number and order of the characters two words share. In record linkage, good results have been obtained using this method or variants of it, in both effectiveness and efficiency [4].
- Levenshtein is the most well-known string distance metric, also often referred to specifically as the string edit distance. It is used mostly in natural language processing and allows for three operations to make one string into another: insertion, deletion and substitution which all have a cost of 1 [35].

There are also edit distances that use statistical information generated on the dataset. An example of statistical information might be if a word is frequent or rare in the given dataset. Using these

features involves doing an additional pass of the data to gather all the required statistics.

Other candidates for features are phonetic algorithms as they allow comparing of two strings that are not exactly equal. As with the string edit distances, there is a lot of overlap between the different phonetic algorithms.

These features can be applied on entire strings, but can also be applied on the different tokens within a string. The *level 2* score is calculated by splitting the string in tokens and recursively applying the distance feature to the different tokens. The scores on the different tokens are combined using the formula shown in Formula 3.1.

$$score(A, B) = \frac{1}{k} \sum_{i=0}^{A_n} max_{j=0}^{B_n} \Big(score(A_i, B_j) \Big) \tag{3.1}$$

Some distances, such as the Monge-Elkan edit distance, already calculate the score on a per token basis internally [41].

3.1.2 Indexing

Performing some sort of indexing is an often necessary optimization step. Comparing every database entry against every other entry is not feasible on databases that can contain hundreds of millions of records. By performing an indexing step the number of pairs that have to be evaluated is reduced. Most practical implementations of duplicate detection systems therefore perform such an indexing prior to comparing records to each other [3].

There are different techniques to reduce the number of comparisons that need to be done in matching. All of these do so by placing some kind of restriction on the duplicates that can be found by the system. The kind of restriction depends on the applied algorithm and its parameters.

As with matching, the indexing has to find a balance between precision and recall. Indexing techniques are mostly focused on recall, but too many candidate pairs have an impact on the speed of the system. While this is a problem in production systems, this might not always be a problem in academic settings. In some scenarios it is even feasible to not include any form of indexing.

Blocking

Blocking [15] is widely used as an indexing technique as it provides a good balance between precision and recall. In blocking, the number of pairs to be evaluated by matching is reduced by dividing the records in a number of blocks. Only records within the same block are compared by matching. Records are typically part of multiple groups as a means to introduce redundancy. Another way to implement blocking is to calculate several hashes for each record. Two records are only compared if there is at least one common hash. This can be efficiently implemented using a hash map, by checking which two records generate a hash collision.

Not surprisingly, the differences between a good configuration and a poor configuration with the same indexing algorithm is far larger than the difference between different algorithms [3]. As a result choosing the right parameter values is essential, regardless of the indexing technique. When the parameters are not chosen by a domain expert, but automatically trained, the quality

depends on the used training algorithm. For blocking systems the typical choice is by far the genetic algorithm, as this tends to result in good configuration values [14, 27].

Also not surprising, indexing techniques, such as blocking, are more restricted in terms of matching quality than matching algorithms. In [27] a F1-score for blocking is reported of 0.42 for one dataset and 0.68 for another. In [3] the F1-score results for blocking are between 0.5 and 0.6 for two of the four datasets, and below 0.1 for the other two. In [14] recall scores are reported between 0.84 and 0.93 for the three datasets.

There are several different algorithms described in literature to automatically determine the near-optimal blocking hashes. There are both supervised and unsupervised algorithms. The supervised algorithms require an annotated sample to generate the blocking hash. The unsupervised variants, on the other hand, do not require annotated data but place several assumptions on the dataset [44].

A common method to generate the blocking hash functions is with the use of a genetic algorithm. The genetic algorithm method takes an incremental and greedy approach when it comes to finding multiple hashes [10, 38]. The algorithm will first try to find the single best hash according to the specified metric. The covered duplicate pairs are then removed from the training set, and the algorithm will try to find the single best hash on this new set. This process continues until all duplicate pairs are covered or additional hashes no longer improve the overall metric. This incremental process is called the Sequential Covering Algorithm [40] and is used in several indexing algorithms.

3.2 Natural language processing with deep learning

While the application of deep learning to duplicate detection on textual data seems to be novel, there is a lot of existing work that covers the processing of textual data. Within the field of natural language processing, deep learning recently has been applied to many aspects with great success.

3.2.1 Semantic hashing

Retrieval of similar entries is not limited to duplicate detection: within the field of information retrieval there are also several techniques that aim to find similar documents given a query. This process is conceptually very similar to the indexing techniques used in duplicate detection.

In [48] a technique is described to generate hashes for documents in such a way to take into account more than just weighted word counts. The process uses an auto-encoder neural network. The values in the middle layer, the chokepoint layer, are eventually used as a hash code. The network is trained to go from the word count vector, through the chokepoint layer, back again to a word count vector. To improve results, a large amount of noise is added just before the middle layer. The values in the middle layer are compared to a threshold to produce a binary code. The threshold was adjusted to account for the fact that low values occur more often than high values.

This is improved upon in [26], where the auto-encoder network is modified to include an error depending on the labeled data. This update error is based upon the difference in hash between the query and document. A relevant label forces the hash to become more equal, an irrelevant label forces the hash to become different. The author reports that using the labeled data in

addition to the unlabeled data increases the NDCG@10 (Normalized Discounted Cumulative Gain) from 0.459 to 0.486 for use in information retrieval systems.

Another technique, also presented in [26], is word hashing. In word hashing sets of n-grams are used to represent words. Doing so increases the NDCG@10 even further, to a reported 0.494. As the number of n-grams is smaller than the number of words, this approach requires less input features than a bag-of-words approach. Word embedding is less complex than word embedding and does not represent the context of word. An additional advantage is that morphological variations are mapped to similar n-grams, making word hashing capable of doing stemming operations.

3.2.2 Word embedding

Another area that is related to duplicate detection of textual data is natural language processing. One of the important recent developments in natural language processing is word embedding.

Traditionally words are encoded by an arbitrary index: the word with index four is then represented by a list of features in which the fourth feature is one and everything else zero. Longer sequences can be represented as a bag-of-words or by concatenating the word feature in order. Both ways of encoding result in a number of features equal or larger than the dictionary size, which is usually very large in most scenarios. The underlying reason is that such encoding does not carry a lot of information. These encodings also have difficulty dealing with words unseen in the training data.

Word embedding instead chooses to represent word as vectors in a low dimensional, continuous space, preferably with some meaningful dimensions. There are different algorithms in use to generate this meaningful encodings. Usually these algorithms are trained on large sets of unlabeled data, such as the entire Wikipedia corpus. Common algorithms are neural networks but also different kinds of principle component analysis, such as for example in [33].

In some cases it is still useful to extend the word embedding with other kinds of information, such as if a word is uppercase [6, 7].

4 Methodology

Each of the research questions implies the comparison of different variants. For example, the main research question, research question 1, is about the effect of additional unlabeled data. To answer this question two variants have to be compared, one of which uses semi-supervised learning to make use of unlabeled data. In same way, the approach to answer the other research questions listed in Chapter 1 is to compare the different algorithm variants against each other on different datasets. This chapter describes the evaluation methodology, how the evaluation is conducted, with the intention of making the results reproducible.

4.1 Evaluation setup

The evaluation consists of an indexing evaluation and a matching evaluation. The setup of both evaluations is largely the same for simplicity. Aspects in which both evaluations differ will be mentioned.

During the training of the different algorithms both labeled and unlabeled data are available. The evaluation is restricted to the part of the dataset that have been manually classified as either duplicate or non-duplicate. As the unlabeled data does not contain labels, it cannot be used during the evaluation phase.

4.1.1 K-fold

The different algorithms are evaluated on the datasets using an K-fold evaluation. In a K-fold evaluation the dataset is partitioned in K distinct folds. In each round the algorithm is trained on all but one of these parts and then evaluated on the part that has been left out. This gives a better approximation of how well the algorithms perform on unseen data. The evaluation on each left out fold results in K confusion matrices. These confusion matrices are summed to create

one overall confusion matrix, which is used for calculation of the error metrics. This process introduces less bias than calculating an error metric for each fold separately and then taking the average [17].

The experiment uses a K-fold evaluation with just 2 folds. Using a higher number of folds will increase the number of training data potentially available to the algorithms. For example, when using a more typical 10 folds, the algorithm would have access to 90% of all labeled data during each training run, compared to 50% with 2 folds. However, as the evaluation is focused on small training sets, the amount of available training data is already artificially limited. Having more folds also reduces the number of labeled pairs still together in the testing fold: if one of the two records in the pair is not in the fold, the pair is incomplete and disappears.

To minimize the influence of the exact distribution of the folds and training data selection, the process of randomly creating the folds and performing the K-fold evaluation is repeated 5 times. This kind of evaluation is known as the 5x2 cross-validation [1]. Repeating the 2-fold more than five times no longer gives new information as the repetitions become too dependent [13]. All the indexing algorithms are evaluated on exactly the same folds, as well as all matching algorithms. No special measures are taken to ensure the folds are kept completely identical between the indexing and matching evaluation.

The classifiers will also be evaluated on the training set to determine if the algorithms are overfitting or underfitting. When determining the best algorithms the results on the training set will not be considered. In the training set evaluation the amount of test data varies between variants, as the different variants will be provided with different amounts of training data.

Distribution of pairs and records over folds

The creation of the folds involves distributing both pairs, for supervised training, and records, for unsupervised training, over the folds. Both categories cannot be simply distributed independently: pairs have to be discarded when both records related to the pairs are not in the same fold. It follows that the probability of this occurring increases as the number of folds increase.

To soften this issue, the pairs are distributed over the folds first. The records are then distributed over the folds according to the pairs they are related to. The remaining records, those that are not linked to any marked pair, will be distributed randomly to be used as unlabeled data.

As some records are linked to in multiple pairs, some records might risk ending up in more than one fold. In this case, the record is assigned to only one of the folds. Assigning a record to multiple folds would result in the possibility of leaking of information from the evaluation data to the training data.

4.1.2 Training set sizes

For each algorithm there will be variants which discard part of the labeled training data, resulting in variants with a different number of training samples. These variants can then be compared against each other, to determine the effect of the training sample size. The part of the training data to discard is selected randomly. Labeled data that has been discarded will stay available to the algorithm as unlabeled data, in addition to the already existing unlabeled data.

To determine which training set sizes provide a useful comparison, an initial evaluation is done on the tuning set with many different training set sizes. Based on the results, it has been decided

to compare the algorithms with 50, 100 and 200 labeled training set samples for each of the two classes, duplicate and non-duplicate.

Some of the algorithms listed below also make use of unlabeled data. For these algorithms unlabeled data is available. Unlike the labeled training data, there are no artificial limits on the number of samples. For practical reasons however, unlabeled data was restricted to 10000 samples (or less) in algorithm implementation.

4.1.3 Error metrics

During the evaluation the metrics precision, recall, F1-score, Matthews correlation coefficient (MCC) and correctness rate will be measured and reported. These error metrics will be used for comparing the different algorithms.

Notice that while for the indexing evaluation recall is considered more important in practice than precision, no such preference applied during the evaluation. The assumption is that algorithms capable of optimizing for balanced precision and recall are also capable of optimizing for recall, as optimizing purely on recall is easier in theory[1].

4.1.4 Significance checking

In addition to measuring error metrics, data is also collected on the significance of the differences between the different algorithms. The significance check is performed using a binomial test [49]. In order to calculate the significance of the difference in performance between two variants all samples that both classified identically are first discarded. The remaining samples are samples A got wrong and B got right, and samples A got right and B got wrong. These differing samples are interpreted as a series of Bernoulli-trials. The p-value for the two algorithms can be calculated using the binomial distribution. The resulting values will be corrected using the Holm-Bonferroni method to adjust the fact that multiple significance checks are performed for all the different variants.

4.2 Matching variations

The first part of the evaluation covers pair-wise matching. To determine the effect of the different matching algorithms the variants are compared against each other. The hyperparameters of each of the different variants are determined using the tuning set (see Section 4.4).

4.2.1 Production matching

The production matching is the matching algorithm used by a commercial company in their latest duplicate detection engine. As this is part of proprietary software, the specific implementations details are not known. In addition, while the other variants are tuned towards a high MCC, this is not the case for the production matching algorithm.

[1]The theoretically best indexing technique in terms of recall is no indexing at all, simply comparing every record against every other record. In practice some importance has to be given to precision.

4.2.2 Linear classifier

A linear classifier is used as a baseline for the matching evaluation. The reason to use a linear classifier is that while a linear classifier has only limited processing capabilities, the input features are already very high-level. In addition, a linear classifier is comparable to the production implementation.

This linear classifier does not work on the string values directly, but instead use a set of features. For each pair to be matched we calculated a set of string distance metrics between the different columns of the two records. These metrics are calculated separately for each of the columns. The result is a feature space of around a hundred numerical features. This feature space also consists of features which indicate anomalies like empty columns. Working with these features is more practical than the original feature space of around ten string features.

For practical reasons the linear classifier is implemented as a neural network containing only a single neuron, which is conceptually identical to a linear classifier. The hyperparameters have been set at 1000 iterations and a learning rate of 0.010, as determined on the tuning set.

4.2.3 Deep learning-based matching

The matching based on deep learning consists of multiple variants. The evaluation is not only to compare traditional matching techniques against deep learning, but also to evaluate multiple deep learning variants and see which technique works best for matching.

Variant	**Features**	**Encoding**	**Semi-supervised**
Belief Network baseline	-	-	semi-supervised
w. Bag-of-words	bag-of-words	both, unmodified	semi-supervised
w. Word Hashing	word hashing	both, unmodified	semi-supervised
w. Word Embedding	word embedding	both, unmodified	semi-supervised
w. Bag-of-words (diff)	bag-of-words	abs. difference	semi-supervised
w. Word Hashing (diff)	word hashing	abs. difference	semi-supervised
w. Word Embedding (diff)	word embedding	abs. difference	semi-supervised
Belief Network baseline (superv. only)	-	-	supervised only
w. Bag-of-words (superv. only)	bag-of-words	both, unmodified	supervised only
w. Word Hashing (superv. only)	word hashing	both, unmodified	supervised only
w. Word Embedding (superv. only)	word embedding	both, unmodified	supervised only
w. Bag-of-words (diff, superv. only)	bag-of-words	abs. difference	supervised only
w. Word Hashing (diff, superv. only)	word hashing	abs. difference	supervised only
w. Word Embedding (diff, superv. only)	word embedding	abs. difference	supervised only

Table 4.1: Overview of the deep learning based matching algorithm variants.
The table indicates for each variant which additional features (in addition to the base features) are included, how the additional features are encoded and how the variant is trained.

There are variants for each type of word encoding techniques: bag-of-words, word hashing and word embedding. The word encoding techniques create a representation for each of the words in

every record. Regardless of word encoding technique, the encoding is applied on only one word at the time. In fields that contain multiple words, the field will be split and the word encoding technique will be applied independently to each of the words in this field. This process is repeated for every field in the record.

As the number of words can vary between different records, a fixed maximum length is set. Values with contain more words than this maximum number are cropped. The maximum length is set in such a way that this is a rare event. Shorter values are padded with a special padding word. Again, this process is used regardless of word encoding technique.

The resulting representation will be provided to the deep belief network. As matching involves two records, the features of the deep belief network includes the representations of both of these two records. In addition to the representations of these two words, the edit distance features that are already present in the belief network baseline are also included.

The hyperparameters for all variants have been optimized using the tuning set. For each of the variants the highest performing configuration of hyperparameters has been selected for use the evaluation. The overview of selected hyperparameters can be found in Table 4.2.

Variant	**Iterations**	**Learning rate**	**Layer size**	**Feature size**	**Ngram size**
Belief Network baseline	100	.100	500	-	-
w. Bag-of-words	100	.010	1000	100	-
w. Word Hashing	1000	.100	100	10	5
w. Word Embedding	1000	.100	100	50	-
w. Bag-of-words (diff)	100	.100	500	100	-
w. Word Hashing (diff)	100	.100	100	100	3
w. Word Embedding (diff)	100	.010	500	100	-

Table 4.2: Overview of the hyperparameter configuration for each of the matching variants The supervised only variants use the same hyperparameter configuration as the corresponding semi-supervised variant. Layer size indicates the size of the hidden layer. Feature size indicates the number of additional inputs in the network per token in the record.

Difference calculation

Instead of providing the word representations of both records separately, it is also possible to provide the difference between both. This makes the new features conceptually a lot closer to the edit distance metrics, which are common in matching, as discussed in Chapter 3. For example, the baseline matching algorithms also use several edit distance features.

This difference calculation variant replaces the features of both records by a new set of features, namely the absolute difference between the two records. This difference is calculated separately for each feature. As both records use the same word encoding techniques this features should be roughly comparable.

The word in the fields between two duplicate records might be present in different orders. This is something the difference calculation cannot represent naturally, as the difference is calculated between each word in one record with the corresponding word in another record. So the first

word of one record is compared against the first word in the other record, the second word to the second word, etc. To handle this case, the difference calculation provides features not just between one word and the word at same position in the other record, but also between a word and all other words in that field.

Bag-of-words

The baseline word encoding technique used during the evaluation is bag-of-words. While simple to implement, bag-of-words has long been used for the representation of words.

In bag-of-words a collection of words is represented by a vector where the value at each index indicates the frequency of the corresponding word. The precise mapping of indexes to words is based on the vocabulary.

As the two other word encoding techniques are applied on word level, this approach was also taken for the bag-of-words implementation. This means that every word in every field has its own bag-of-words vector, and each of these bags-of-words cover only one word. In other words, the values in the vector will be zero everywhere, except for the value at the index of the word that is being represented.

Word hashing

Word hashing [26] is a technique in which the N-grams of a word are used to represent a word. The advantage of this over a normal bag-of-words is that textually similar words are grouped together. For example, the presentation of 'arguments' and 'argument' will be very similar. The idea behind word hashing is roughly comparable to that of stemming.

In the word hashing variants all words are represented as a collection of short substrings of a few characters. These character N-grams are then represented using a bag-of-words encoding, but with every value indicating the presence of a N-gram instead of a word. Like in bag-of-words, the result will be a vector where the value at every index will indicate the frequency of the corresponding N-gram. For simplicity frequencies are restricted to either non-zero or zero.

For the N-grams encoding N-grams with length 3 are used, which are also known as trigrams. Notice that the N-grams are on character level, so each trigram will be a string of 3 characters. The start and end of a word are explicitly modeled, by padding words with a padding character before converting to N-grams. So for example 'word' would result in the N-gram '_wo', 'wor', 'ord' and 'rd_'.

Word embedding

Word embedding is a technique which tries to represent the meaning of a word rather than its textual representation. In order to do so, word embedding is not, in contrast to the other variants, based on the bag-of-words model. In word embedding the word vectors are created by trying to predict the context in which by word is used by the word itself, using for example a neural network.

The word embedding model used in the evaluation is trained using the provided unlabeled data. While pre-trained word embedding models are available for English natural text, the datasets

used in this evaluation do not contain English or natural language. Separate word embedding models are trained for each column.

4.3 Indexing variations

The second part of the evaluation covers the indexing phase in the duplicate detection process. In order to determine the effect of different deep learning techniques as well as the effect of different training sizes, several indexing variants are compared, similarly to the matching variants. Besides the deep learning variant, the commonly used genetic algorithm trained blocking indexing is also included to serve as a baseline. The hyperparameters of the different indexing variants are determined using a tuning evaluation on the tuning set.

All indexing algorithms described below are based on an indexing technique called blocking. In blocking records are divided in blocks using hash functions. For each record multiple hashes are calculated. Only records with hash collisions are considered for more in-depth matching. The main difference between the different algorithms is in how to determine the different hashing functions.

While normally multiple hash functions are calculated for each record, all algorithms below have been configured to generate only one hash. By forcing all algorithms to generate only one hash the comparison between algorithms becomes focused on the quality of the generated hash and not the number of hashes. The assumption is made that an algorithm capable of generating one high quality hash function is also capable of generating multiple hashes. Techniques like sequential covering [40], that do exactly that, are already commonplace in indexing (see also Section 3.1.2).

4.3.1 Production indexing

The production indexing is the indexing algorithm used by a commercial company in their latest duplicate detection engine. Like the production matching variant, this is proprietary software and the specific implementations details are not known.

As the production indexing algorithm generates multiple hashes and there is no way to configure this, the single hash function with the highest MCC on the training set will be selected from the generated hash functions.

4.3.2 Blocking with a genetic algorithm

Blocking in combination with a genetic algorithm is a common choice for indexing in duplicate detection: here genetic algorithms are used to find the optimal hash function. As it is a common choice, this variant is used as a baseline in the indexing evaluation. The genetic algorithm uses an evaluation function to evaluate candidates, which is a simple evaluation of the candidate hash on the training set. As the fitness metric the MCC score over the training data is used.

Genetic algorithms have the tendency to create results that are overly specific to the given training set. One way to reduce this form of overfitting is use different training samples every generation [20]. This is realized by limiting the fitness function to a subset of complete training set. This subset changes with every generation, which in turn results in slightly different fitness functions every generation.

The hyperparameters for the genetic algorithm have been optimized on tuning set. This resulted in a 100000 population size, 64 elite count, .10 mutation rate, 30 generations and 75% of the training data is used in each generation.

4.3.3 Deep learning-based indexing

To determine the effect of deep learning techniques on the indexing phase, deep learning variants are compared to a few baseline implementations. While the deep learning techniques have shown good results in other tasks, their performance in duplicate detection is yet to be determined. For this reason a number of deep learning variants are evaluated, each with a different combination of deep learning techniques.

Variant	**Features**	**Semantic Hashing**
Blocking baseline	-	-
w. Bag-of-words	bag-of-words	no
w. Word Hashing	word hashing	no
w. Word Embedding	word embedding	no
w. Bag-of-words (semantic h.)	bag-of-words	semi-supervised
w. Word Hashing (semantic h.)	word hashing	semi-supervised
w. Word Embedding (semantic h.)	word embedding	semi-supervised
w. Bag-of-words (semantic h., superv. only)	bag-of-words	supervised only
w. Word Hashing (semantic h., superv. only)	word hashing	supervised only
w. Word Embedding (semantic h., superv. only)	word embedding	supervised only

Table 4.3: Overview of the deep learning based indexing algorithm variants.
The table indicates for each variant which additional features (in addition to the base features) are included, how the additional features are encoded.

The implementations for the different word encoding techniques are identical to the implementations used in the matching phase.

As with the matching variants, the indexing variants have been optimized using the tuning set. In Table 4.4 an overview is given of the hyperparameters for the different variants. In each of the variants the hyperparameters of the genetic algorithm are identical to those in the blocking baseline.

Semantic hashing

In semantic hashing, within an information retrieval context, a document is converted a code by running it through an auto-encoder network. The activation values in the compressed middle layer are then used as the code for the document. In the original application semantic hashing is trained completely unsupervised.

There are a number of differences between document retrieval and duplicate detection, making semantic hashing not directly applicable for duplicate detection. For example, full documents are longer and contain more noise such as stop words. Contact data on the other hand is far more

Variant	Iterations	Learning rate	Layer size	Feature size	Ngram size
w. Bag-of-words	-	-	-	100	-
w. Word Hashing	-	-	-	100	3
w. Word Embedding	-	-	-	100	-
w. Bag-of-words (semantic h.)	10	.010	400	100	-
w. Word Hashing (semantic h.)	50	.010	200	100	3
w. Word Embedding (semantic h.)	10	.0001	200	100	-

Table 4.4: Overview of the hyperparameter configuration for each of the indexing variants. The supervised only variants use the same hyperparameter configuration as the corresponding semi-supervised variant. Layer size indicates the size of the bottleneck layer in the auto-encoder network. Feature size indicates the number of additional inputs in the network per token in the record.

concise. In addition, the requirements for automated duplicate detection are generally stricter than for search.

Another difference in usage is that sematic hashing is used in combination with the genetic algorithm. The genetic algorithm and its hyperparameters are identical to the baseline variant. The genetic algorithm is also provided with the original values that are being used in the baseline genetic algorithm variant.

The semantic hashing code is originally a vector of real values. In the original application of document retrieval these values are converted to binary values using a threshold. The threshold is selected so that both bit values are equally likely. The same conversion is applied here, resulting in a number of bit values.

These bit values are used as input for the genetic algorithm. Each of these bit values are provided separately, allowing the genetic algorithm to pick only some of the bit values to use for blocking. This gives the genetic algorithm control more control over the strictness than providing the semantic hashing code as one large value. In addition to the fields containing specific bits from the semantic hashing, the genetic algorithm is also provided with the fields already used in the blocking baseline.

The semantic hashing itself is trained on the input words, represented either using the bag-of-words encoding, word hashing or word embedding. Depending on the variant, training is done either semi-supervised or supervised only. In the semi-supervised variant, the auto-encoder is first trained using unlabeled data. In the supervised step penalties are introduced in the auto-encoder that forces duplicate pairs to have more similar semantic hashes, and non-duplicate pairs to have different semantic hashes.

In the variants without semantic hashing the input words, in the corresponding representation, are directly converted to bit values. These bit values are provided to the genetic algorithms, again providing each bit value as a separate field.

4.4 Datasets

The different algorithms will be evaluated on two datasets in order to compare the differences in performance. In addition, a third dataset, the tuning set, will be used for tuning. All three datasets are samples taken from real-life datasets. As one of the evaluation datasets originates from the same source as the tuning set, there a possibility its results are overly optimistic results. By including a second dataset, finding can be confirmed on a dataset that is not related to the tuning set.

	Dataset C	**Dataset R**	**Tuning set**
# records	100000	65384	32676
# labeled pairs	3292	5884	2144
# labeled duplicate pairs	679	1985	845
# labeled non-duplicate pairs	2613	3899	1299

Table 4.5: Overview of the datasets used.
Notice that the tuning set is only used for tuning, not during the actual evaluation.

The datasets have undergone an annotation phase in which a domain expert manually marked pairs as either duplicate or non-duplicate. Although the process of annotating data has been extensive, due to the large amount of possible pairs, only a part of all possible pairs has been annotated. For both datasets the number of records and number of annotated pairs are listed in Table 4.5.

In preparation for the evaluation, all datasets have been preprocessed to make it more suitable for duplicate detection. The preprocessing consists of conversation to lowercase, removal of diacritics and trimming of whitespaces. The preprocessing has been applied indiscriminately to all fields of all records.

The tuning set and dataset R are created using the same original source. To split the original dataset in the two sets the original dataset was split into three parts: two parts were used for dataset R and the third is used for the tuning set. The logic to separate the datasets is the same logic as used during the creation of the K-folds, as described earlier. This method of splitting increases the amount of labeled pairs that is still valid after the splitting, while still assuring that both datasets have no common records (and therefore no common key pairs).

The purpose of the tuning set is solely to be used for tuning of the classifiers.The tuning set is therefore not included in the actual evaluation. The other two datasets are used for the evaluation, but not for tuning. As mentioned, the tuning set and dataset R are created using the same original dataset. Dataset C is created from a different dataset and is intended to give insight in the performance on dataset that is not related to the tuning set.

5
Evaluation results

This chapter contains the results of the evaluation described in Chapter 4. These results will be interpreted and discussed in Chapter 6.

The algorithms listed in the different tables are referred to by a name indicating the algorithm used and a number indicating the training size used. Notice that a training size of n describes a classifier trained on n reference pairs marked as duplicate and n reference pairs marked as non-duplicate, resulting in a total reference size of $2n$. In addition to marked reference pairs, some algorithms also use unlabeled data.

The main part of the evaluation consists of a K-fold evaluation for both indexing and matching. The results for the indexing algorithms on dataset C are listed in Table 5.4 and in Table 5.3 for dataset R. The results for the matching algorithms are listed in Table 5.2 for dataset C and in Table 5.1 for dataset R.

The complete overview of which of these results are significant can be found in Appendix A. The different algorithms were also evaluated in a training set evaluation: these results can be found in Appendix B.

Algorithm	n	**Prec.**	**Rec.**	**F1**	**MCC**	**Corr.**
w. Word Embedding (diff, superv. only)	200	.921	.957	.939	.901	.953
w. Bag-of-words	200	.922	.956	.938	.901	.953
w. Bag-of-words (diff, superv. only)	200	.917	.961	.939	.901	.953
w. Bag-of-words (superv. only)	200	.919	.956	.938	.900	.952
w. Word Hashing (diff, superv. only)	200	.915	.959	.937	.898	.952
Production baseline	200	.931	.938	.935	.896	.951
w. Word Embedding (diff)	200	.915	.957	.936	.896	.951
Belief Network baseline	200	.914	.958	.936	.896	.951
Belief Network baseline (superv. only)	200	.913	.957	.935	.895	.950
w. Word Hashing (superv. only)	200	.916	.953	.934	.894	.950
w. Bag-of-words (diff)	200	.911	.959	.934	.894	.950
w. Word Hashing (diff)	200	.912	.956	.934	.894	.949
w. Word Embedding	200	.922	.945	.933	.893	.949
w. Word Hashing	200	.914	.954	.933	.893	.949
w. Word Embedding (superv. only)	200	.921	.943	.932	.891	.949
w. Bag-of-words (diff, superv. only)	100	.910	.947	.928	.884	.945
Belief Network baseline (superv. only)	100	.909	.943	.926	.880	.943
Belief Network baseline	100	.909	.942	.925	.879	.943
w. Word Embedding (diff, superv. only)	100	.908	.942	.924	.878	.942
w. Bag-of-words (diff)	100	.907	.942	.924	.878	.942
w. Word Hashing (diff, superv. only)	100	.904	.945	.924	.878	.942
w. Word Hashing (diff)	100	.905	.938	.921	.873	.940
w. Bag-of-words (superv. only)	100	.905	.938	.921	.873	.940
w. Bag-of-words	100	.908	.934	.921	.873	.940
w. Word Embedding (superv. only)	100	.904	.936	.920	.871	.939
w. Word Embedding	100	.906	.933	.920	.871	.939
w. Word Embedding (diff)	100	.905	.934	.919	.871	.939
Linear classifier baseline	200	.897	.942	.919	.869	.938
w. Word Hashing (superv. only)	100	.901	.934	.917	.866	.937
w. Word Hashing	100	.892	.936	.914	.861	.934
Linear classifier baseline	100	.877	.922	.899	.837	.923
w. Word Hashing (diff, superv. only)	50	.877	.912	.894	.830	.919
w. Bag-of-words (diff, superv. only)	50	.875	.915	.894	.830	.919
w. Word Embedding (diff, superv. only)	50	.878	.909	.893	.828	.919
Belief Network baseline	50	.876	.907	.891	.825	.917
w. Word Hashing (diff)	50	.885	.894	.889	.823	.917
w. Word Embedding (diff)	50	.883	.894	.888	.821	.916
w. Bag-of-words (superv. only)	50	.874	.905	.889	.822	.916
w. Bag-of-words (diff)	50	.875	.902	.889	.821	.915
Belief Network baseline (superv. only)	50	.868	.911	.889	.820	.915
Production baseline	100	.889	.880	.885	.816	.914
w. Word Embedding	50	.878	.892	.885	.816	.913
w. Word Embedding (superv. only)	50	.876	.894	.885	.815	.913
w. Word Hashing (superv. only)	50	.871	.900	.886	.816	.913
w. Bag-of-words	50	.879	.888	.884	.814	.913
w. Word Hashing	50	.873	.891	.882	.810	.911
Linear classifier baseline	50	.837	.897	.866	.782	.896
Production baseline	50	.864	.851	.857	.774	.894

Table 5.1: K-fold evaluation results of the matching algorithms on dataset R. The algorithms are trained using with differing amounts of training data (n). For each algorithm the overall precision, recall, F1-score, Matthews correlation coefficient (MCC) and correctness rate over all runs are reported.

Algorithm	n	**Prec.**	**Rec.**	**F1**	**MCC**	**Corr.**
w. Word Embedding (superv. only)	200	.853	.955	.901	.872	.952
w. Bag-of-words (superv. only)	200	.854	.950	.899	.870	.951
w. Word Embedding	200	.848	.953	.897	.867	.950
w. Word Hashing (superv. only)	200	.853	.938	.893	.861	.949
w. Bag-of-words	200	.849	.944	.894	.862	.949
w. Word Hashing	200	.850	.942	.893	.862	.949
w. Bag-of-words (diff)	200	.838	.945	.888	.855	.945
w. Word Hashing (diff, superv. only)	200	.835	.946	.887	.854	.945
w. Word Hashing (diff)	200	.832	.946	.885	.852	.944
w. Bag-of-words (diff, superv. only)	200	.832	.944	.884	.850	.943
w. Word Embedding (diff, superv. only)	200	.823	.949	.882	.847	.942
Production baseline	200	.843	.916	.878	.841	.941
w. Word Embedding (diff)	200	.819	.952	.880	.845	.941
Belief Network baseline (superv. only)	200	.818	.935	.872	.834	.937
Belief Network baseline	200	.813	.935	.870	.831	.936
Production baseline	100	.815	.923	.866	.825	.934
w. Word Embedding	100	.817	.916	.864	.822	.934
w. Bag-of-words (superv. only)	100	.815	.918	.863	.822	.933
Linear classifier baseline	200	.793	.958	.868	.830	.933
w. Bag-of-words	100	.812	.913	.860	.817	.932
w. Word Hashing (superv. only)	100	.803	.916	.856	.812	.929
w. Word Embedding (superv. only)	100	.799	.921	.856	.813	.929
w. Bag-of-words (diff, superv. only)	100	.797	.916	.852	.808	.927
w. Word Hashing	100	.797	.916	.852	.808	.927
w. Bag-of-words (diff)	100	.792	.921	.852	.808	.927
w. Word Hashing (diff)	100	.790	.924	.851	.807	.926
w. Word Embedding (diff, superv. only)	100	.786	.917	.846	.800	.924
w. Word Hashing (diff, superv. only)	100	.784	.919	.846	.800	.923
w. Bag-of-words (superv. only)	50	.778	.913	.840	.792	.920
w. Word Embedding	50	.780	.896	.834	.783	.918
w. Word Embedding (diff)	100	.770	.911	.834	.784	.917
Belief Network baseline	100	.774	.900	.832	.781	.917
Linear classifier baseline	100	.754	.938	.836	.788	.916
Production baseline	50	.766	.905	.830	.778	.915
w. Word Hashing (superv. only)	50	.760	.912	.829	.778	.914
w. Bag-of-words (diff)	50	.755	.918	.829	.778	.913
w. Word Embedding (superv. only)	50	.757	.912	.827	.776	.913
Belief Network baseline (superv. only)	100	.761	.900	.825	.771	.912
Belief Network baseline	50	.756	.897	.821	.766	.910
w. Word Hashing (diff)	50	.745	.924	.825	.773	.910
w. Bag-of-words	50	.752	.901	.820	.765	.909
w. Bag-of-words (diff, superv. only)	50	.745	.917	.822	.769	.909
w. Word Embedding (diff)	50	.738	.913	.816	.761	.906
w. Word Embedding (diff, superv. only)	50	.736	.910	.814	.758	.904
w. Word Hashing (diff, superv. only)	50	.730	.917	.813	.758	.903
w. Word Hashing	50	.733	.905	.810	.753	.903
Belief Network baseline (superv. only)	50	.738	.889	.806	.747	.902
Linear classifier baseline	50	.707	.929	.803	.746	.896

Table 5.2: K-fold evaluation results of the matching algorithms on dataset C.
The algorithms are trained using with differing amounts of training data (n). For each algorithm the overall precision, recall, F1-score, Matthews correlation coefficient (MCC) and correctness rate over all runs are reported.

Algorithm	n	**Prec.**	**Rec.**	**F1**	**MCC**	**Corr.**
w. Word Hashing	200	.776	.668	.718	.573	.804
w. Bag-of-words	200	.764	.681	.720	.571	.802
w. Word Embedding	200	.776	.659	.713	.567	.802
w. Bag-of-words (semantic h.)	200	.764	.663	.710	.559	.797
Blocking baseline	100	.744	.669	.704	.545	.790
w. Word Hashing (semantic h., superv. only)	200	.778	.613	.686	.540	.790
Production baseline	200	.727	.699	.713	.547	.789
Blocking baseline	200	.736	.677	.705	.542	.789
w. Word Hashing (semantic h.)	200	.754	.640	.692	.536	.787
w. Bag-of-words (semantic h., superv. only)	200	.735	.672	.702	.538	.787
w. Word Embedding	100	.709	.693	.700	.525	.779
Blocking baseline	50	.713	.682	.697	.523	.779
w. Word Embedding (semantic h.)	200	.729	.642	.683	.514	.777
w. Bag-of-words	100	.718	.665	.690	.517	.777
w. Word Hashing	100	.703	.675	.689	.509	.772
w. Word Embedding (semantic h.)	50	.724	.601	.657	.485	.765
w. Word Embedding	50	.667	.720	.693	.499	.761
w. Word Embedding (semantic h.)	100	.722	.567	.635	.464	.757
w. Bag-of-words	50	.661	.684	.672	.471	.751
w. Word Hashing (semantic h.)	100	.670	.621	.644	.446	.744
w. Word Embedding (semantic h., superv. only)	200	.702	.522	.599	.421	.739
w. Bag-of-words (semantic h., superv. only)	100	.648	.648	.648	.438	.737
Production baseline	50	.621	.739	.675	.458	.734
Production baseline	100	.622	.711	.664	.444	.731
w. Word Hashing	50	.613	.735	.669	.446	.728
w. Word Hashing (semantic h., superv. only)	100	.649	.562	.602	.393	.723
w. Bag-of-words (semantic h.)	100	.619	.660	.639	.412	.721
w. Bag-of-words (semantic h., superv. only)	50	.622	.636	.629	.404	.720
w. Word Embedding (semantic h., superv. only)	100	.643	.516	.572	.364	.712
w. Word Hashing (semantic h.)	50	.600	.551	.574	.338	.695
w. Bag-of-words (semantic h.)	50	.572	.669	.617	.361	.689
w. Word Hashing (semantic h., superv. only)	50	.583	.565	.574	.326	.686
w. Word Embedding (semantic h., superv. only)	50	.602	.441	.509	.290	.682

Table 5.3: K-fold evaluation results of the indexing algorithms on dataset R.
The algorithms are trained using with differing amounts of training data (n). For each algorithm the overall precision, recall, F1-score, Matthews correlation coefficient (MCC) and correctness rate over all runs are reported.

Algorithm	n	**Prec.**	**Rec.**	**F1**	**MCC**	**Corr.**
w. Word Hashing	200	.508	.857	.638	.527	.777
w. Word Hashing (semantic h.)	100	.508	.682	.582	.442	.776
w. Bag-of-words	200	.504	.872	.639	.530	.774
w. Word Hashing (semantic h., superv. only)	200	.502	.745	.600	.466	.772
w. Bag-of-words	100	.499	.869	.634	.523	.770
w. Word Hashing	100	.498	.843	.626	.509	.769
w. Word Hashing (semantic h.)	200	.494	.788	.607	.478	.766
w. Bag-of-words (semantic h., superv. only)	50	.489	.618	.546	.395	.765
w. Bag-of-words	50	.492	.854	.625	.509	.765
w. Word Embedding	50	.489	.863	.624	.509	.762
Production baseline	50	.486	.730	.584	.443	.761
w. Word Hashing (semantic h., superv. only)	100	.480	.655	.554	.403	.758
w. Bag-of-words (semantic h., superv. only)	200	.482	.816	.606	.479	.757
w. Word Embedding (semantic h., superv. only)	50	.474	.584	.524	.366	.756
w. Word Embedding	100	.481	.862	.618	.500	.755
w. Bag-of-words (semantic h., superv. only)	100	.478	.764	.588	.450	.755
w. Word Embedding	200	.480	.871	.619	.504	.754
w. Word Hashing	50	.478	.786	.594	.460	.754
w. Bag-of-words (semantic h.)	50	.464	.628	.534	.375	.749
w. Word Hashing (semantic h.)	50	.462	.609	.525	.364	.748
w. Bag-of-words (semantic h.)	200	.468	.839	.601	.474	.744
w. Word Embedding (semantic h.)	50	.457	.649	.537	.377	.743
Production baseline	200	.467	.851	.603	.479	.743
w. Bag-of-words (semantic h.)	100	.464	.802	.588	.453	.743
w. Word Embedding (semantic h., superv. only)	100	.463	.836	.596	.467	.740
Production baseline	100	.461	.843	.596	.469	.738
w. Word Embedding (semantic h.)	100	.459	.811	.587	.451	.738
w. Word Embedding (semantic h., superv. only)	200	.457	.925	.612	.504	.731
w. Word Embedding (semantic h.)	200	.457	.925	.612	.504	.731
Blocking baseline	200	.457	.925	.612	.504	.731
Blocking baseline	100	.457	.925	.612	.504	.731
w. Word Hashing (semantic h., superv. only)	50	.422	.538	.473	.295	.725
Blocking baseline	50	.446	.887	.594	.472	.722

Table 5.4: K-fold evaluation results of the indexing algorithms on dataset C.
The algorithms are trained using with differing amounts of training data (n). For each algorithm the overall precision, recall, F1-score, Matthews correlation coefficient (MCC) and correctness rate over all runs are reported.

6

Discussion of results

This chapter focuses on the interpretation and discussion of the results of the evaluation, as shown in Chapter 5. The results in the signficance tables (see Appendix A) and in the training set evaluation (see Appendix B) are also used to aid the interpretation of the results. For clarity this chapter in split into two parts, one discusses the results of the matching algorithms, and one discusses the results of the indexing algorithms.

6.1 Matching evaluation results

When looking at the matching results, a few gaps in performance stand out. Generally, and in line with expectations, variants with a larger training set tend to perform better. This effect is strongest in the matching algorithms, where even best algorithm gets outperformed by almost all of the variants which use more training data. In the indexing evaluation there is more overlap between the different training set sizes, but more adding more training data to same algorithm never seems to worsen the results.

Another very noticeable effect happens in the training set results. For all algorithms aside from the *Linear classifier baseline* and *Production baseline*, the training set results show 'perfect' scores. Although these results are not that much higher than those in the K-fold evaluation, they do seem to indicate that the algorithms are suffering of overfitting. Techniques such as regularisation are likely to be a good starting point for further improvement.

A lot of the differences between different algorithms are not directly relevant to the research questions listed in Chapter 1. The discussion below focuses on the differences between algorithms relevant to the research questions.

When looking at the differences between the *Linear classifier baseline* and *Belief Network baseline (superv. only)*, the belief network seems to score higher on both in most scenarios, except with 100 training samples on dataset C. When looking at the significance tables however, these

differences on dataset C turn out to be not significant. The *Belief Network baseline (superv. only)* signifcantly outperforms the *Linear classifier baseline* on dataset R for all dataset sizes. However, as mentioned, there are no significant differences between these algorithms on dataset C. This difference between the two datasets could be the result of differences between the datasets, for example dataset C being overall less complex. It could also be a result of the fact that the tuning set originated from the same source as dataset R, making the belief network, which has more hyperparameters than the linear model, more optimized for the dataset R.

When comparing semi-supervised training with supervised only training, a lot of the supervised only algorithms appear to score at the top of evaluation results for both datasets, especially the variants where there are also additional features.

Between the *Belief Network baseline* and *Belief Network baseline (superv. only)* there is only one significant result: on dataset C with 50 training samples the semi-supervised variant significantly outperforms the supervised only variant. For the variants with additional features there are also a few significant results, also all on dataset C. For 100 training samples *w. Word Embedding (diff, superv. only)* significantly outperforms *w. Word Embedding (diff)*. For 50 training samples *w. Bag-of-words (superv. only)* significantly outperforms *w. Bag-of-words* and *w. Word Hashing (superv. only)* significantly outperforms *w. Word Hashing*.

The first thing noticeable about these results is the fact that the significant differences between semi-supervised and supervised only seem to occur on dataset C. In addition, without extra features semi-supervised tends outperform supervised only, but with extra features it is the other way around and supervised only performs better. These results seem to suggest that the additional features do not work well in combination with semi-supervised training. A possible explanation would be that the unsupervised training has difficulty creating a more compact representation of the extra features in addition to the standard features, as the layer configuration stays identical. That most differences occur on smaller training sets is in line with expectation: the semi-supervised pretraining has a larger influence when the training sizes are smaller.

The final research question is about the effect extra features have on matching quality. Looking at the results table, a lot of the variants with additional features seem to score higher than the baseline. But when comparing to *Belief Network baseline* these differences are for a large part not significant. On dataset R all differences are not significant. On dataset C, most variants with additional features significantly outperform the belief network on training set sizes 100 and 200. The exceptions are *w. Word Hashing (diff, superv. only)* on training set size 100, and *w. Word Embedding (diff)* and *w. Word Embedding (diff, superv. only)* on both 100 and 200. For 50 training samples only *w. Bag-of-words (superv. only)* significantly outperforms the belief network.

Comparing the additional features to a baseline variant without any additional features is one thing, but to properly answer the related research question it is also important to see how new features such as word hashing and word embedding compare to the more traditional bag-of-words. After all, looking at the results table shows that the simple bag-of-words encoding is scoring very well.

The significance table shows that on dataset R both *w. Word Embedding (diff, superv. only)* and *w. Word Hashing (diff, superv. only)* significantly outperform *w. Bag-of-words* with 50 training samples, but so does *w. Bag-of-words (diff, superv. only)*. A similar situation occurs on dataset C for all training sample sizes, where *w. Word Embedding* significantly outperforms *w. Bag-of-words (diff)*, but so does *w. Bag-of-words (superv. only)*. In general it seems that the combination of other parameters, such as *diff* and semi-supervised or supervised only, have a larger effect than the actual choice of feature representation. In this sense, a simple bag-of-words

implementation, given the right configuration of the other parameters, performs just as well as more computationally complex features.

Finally there are difference variants based on how these features are encoded for the two records. For *diff* variants only the absolute difference between both records is provided, in the other variants the features are simply provided unmodified for both records. When looking at the results table, for dataset R both seem to score equally well, whereas for dataset C providing the values unmodified seems to be scoring higher than *diff*.

The significance table shows a couple of significant results on dataset R: *w. Bag-of-words (diff, superv. only)* significantly outperforms *w. Bag-of-words* on both 50 and 100 training samples. On 50 training samples *w. Word Embedding (diff, superv. only)* also significantly outperforms *w. Word Embedding* and *w. Word Hashing (diff, superv. only)* significantly outperforms *w. Word Hashing*. For dataset C, *w. Bag-of-words (superv. only)* significantly outperforms *w. Bag-of-words (diff)* for all training set sizes, as well *w. Word Embedding* significantly outperforms *w. Word Embedding (diff)*, also for all training set sizes.

It is interesting to note that on dataset R *diff* performs better, while for dataset C providing the feature unmodified performs better. This difference might be due to subtle differences between the two datasets.

6.2 Indexing evaluation results

Besides evaluating different matching algorithms, we also evaluated several indexing algorithms. A first look at the indexing results show that the scores are not only a lot lower than the scores of the matching algorithms, they are also more spread out and there is a larger overlap between the different training sample sizes.

The lower results compared to the matching algorithms are not surprising: the indexing algorithms are conceptually limited compared to the matching algorithms, and this shows in the matching quality. The indexing algorithms calculate only a single hash for each record, if two records have the same hash they are considered a match. The advantage of this approach is that it scales a lot better to large datasets.

When looking at the result tables, there are a number of patterns visible. For example, as expected, the top variants are those with 200 training samples. The variants on top are the ones that use additional features, but mostly in the unmodified variant. The bottom of the list also seems to be dominated by feature variants, but with semantic hashing, in particular supervised only and lower training sizes. On dataset C, the *Blocking baseline* is also very noticeable at the bottom.

This might be the result of the fact that the blocking baseline has been tuned on a dataset which is related to dataset R but not C. When looking at the training set results, the blocking baseline suffers from overfitting on dataset C, but not on dataset R, where the cross evaluation error and training error are close. A possible explanation would be that both datasets differ in complexity with dataset C being more complex, and the algorithm has been tuned towards this complexity. The variants with extra features also less affected by this, as the additional features add complexity.

As with the matching algorithms, overfitting also seems to be a problem in most other indexing algorithms. While the results of the training set evaluation are not as extreme as for the matching

algorithms, which would not be possible due to constraints placed on the indexing algorithms, the results for the different algorithms are overall higher on the training set.

When comparing the different variants with additional features to the blocking baseline without additional features, there are several differences. The blocking baseline with 200 training samples is scoring higher than most other variants on dataset R, with a few exceptions. These are *w. Bag-of-words*, *w. Bag-of-words (semantic h.)*, *w. Word Hashing*, *w. Word Embedding*, which both significantly outperform the baseline. As already mentioned, the *Blocking baseline* scores a lot lower compared to other variants on dataset C. As a result almost all of the variants outperform the *Blocking baseline* on this dataset. When comparing against the highest scoring *Production baseline* instead, only the *w. Word Hashing* variant significantly outperforms the baseline. One of the reasons that word hashing performs better than the bag-of-words or the word embedding encoding might be due to the fact that it is not limited by typing mistakes and spelling variations.

When looking specifically at the addition of the *w. Word Hashing* and *w. Word Embedding* features compared to the traditional *w. Bag-of-words* encoding, there do not seem to be any large differences in performance. Neither of the two algorithms manages to significantly outperform the list of word variant with the same amount of training data. In fact, the list of word variants significantly outperforms the word embedding variant. It is possible that techniques such as word embedding are not necessarily suited for use in the indexing. It might also that, for example, word embedding is less suited due to the type of datasets in this domain, in which fields are typically only short strings and not proper full natural text.

In addition to just the extra features, the evaluation also included variants in which the extra values are passed through a semantic hashing step. On both datasets, the semantic hashing variants tend to score lower than the corresponding variants without semantic hashing. In almost all cases, the variant without semantic hashing significantly outperforms the variant with semantic hashing. On dataset R, exceptions are the bag-of-words with 200 training samples and word embedding with 50 training samples. On dataset C, the exception is word hashing for every training sample size.

The lower scores for the variants with semantic hashing can be explained as these variants compress the feature space, while the variants without semantic hashing can use the entire uncompressed feature space. It is likely that the feature space is has little redundancy, forcing the semantic hashing algorithm to throw away potentially useful information.

One of the research questions is about the effect of semi-supervised training compared to supervised only. Two variants of semantic hashing are included in the indexing evaluation for this purpose: one with semi-supervised training and one using only supervised training. As already mentioned, on first sight the supervised only variants seem to on the lower scoring end.

The significance table shows a similar pattern. On dataset R, *w. Word Embedding (semantic h.)* significantly outperforms *w. Word Embedding (semantic h., superv. only)* for every training set size. *w. Word Hashing (semantic h.)* significantly outperforms *w. Word Hashing (semantic h., superv. only)* with 100 training samples. *w. Bag-of-words (semantic h.)* significantly outperforms *w. Bag-of-words (semantic h., superv. only)* with 200 training samples. In this case, the reverse also happens twice: for 50 and 100 training samples *w. Bag-of-words (semantic h., superv. only)* significantly outperforms *w. Bag-of-words (semantic h.)*. On dataset C, the only significant result is with 100 training samples, where *w. Word Embedding (semantic h.)* significantly outperforms *w. Word Embedding (semantic h., superv. only)*.

7

Conclusions

Chapter 1 listed several questions that together form the backbone of this work. To be able to answer these research questions, an evaluation was done with many different algorithm variants, both matching and indexing (see Chapter 4). In this chapter we will look back at each of these questions and try to answer them based on the results as interpretered in the discussion (Chapter 6).

Research question 1 *How does the use of unlabeled data in addition to labeled data influence duplicate detection quality?*

The evaluation included multiple variants that used unlabeled data and multiple variants that didn't use unlabeled data, for both matching and indexing algorithms.

For the matching algorithms the addition of unlabeled data seems to have little effect on the quality of the matching. Of the two datasets on which the algorithms are evaluated, one, dataset R, has no significant differences between the two. On the other dataset, dataset C, semi-supervised training has the expected effect, but only on a small training set of 50 samples (improving the F1-score from 0.806 to 0.821). Overall, while adding unlabeled data does not improve the quality in most cases, it does not seem to harm it either.

When combining the semi-supervised training with the additional features, however, the addition of unlabeled data does not seem to have any impact, as none of the differences are significant.

For the indexing algorithms semi-supervised training improves the quality over supervised only training as expected, with the differences being most clear on dataset R (e.g. improving the F1-score from 0.702 to 0.710 for *w. Bag-of-words (semantic h.)* with 200 samples on dataset R). In contrast to matching, in indexing the additional features do not seem to be a problem. However, both approaches are vastly different and it is likely that this difference originates somewhere in those differences.

Research question 2 *How does the use of layers such as auto-encoders to create a belief network influence duplicate detection quality?*

As part of the evaluation several variants were included to compare baseline algorithms with algorithms incorporating auto-encoders. This was to distinguish between the effects of the changes in network structure and changes in the use of unlabeled data. Adding auto-encoder layers is a technique that allows for semi-supervised training.

For the matching algorithms the additional auto-encoder layer improves quality on dataset R (e.g. improving the F1-score from 0.919 to 0.935 with 200 samples), and does not seem to have any effect on the other dataset, dataset C. Overall, while improving results on only one dataset, the additional auto-encoder layer does not seem to hurt quality either.

For the indexing algorithms quality seemed to be lower for variants with additional encoder layers (e.g. reducing the F1-score from 0.720 to 0.702 for *w. Bag-of-words* with 200 samples on dataset R). This is the opposite result of the matching algorithm. Again, these differences are likely the result of different approaches between indexing and matching. In particular, the lower results are likely due to the fact that the additional layers are used in a semantic hashing setup. Compared to situation without semantic hashing, semantic hashing reduces the feature space, leading to lesser performance.

Research question 3 *How does the use of other techniques commonly used in natural language processing with deep learning, such as word embedding and semantic hashing, influence duplicate detection quality?*

To determine the effect of additional features, the evaluation also included several variants with different sets of features.

For the matching algorithms there is no difference on one of the datasets, dataset R. On the other dataset, dataset C, the additional features improve the matching quality (e.g. improving the F1-score from 0.872 to 0.899 for *w. Bag-of-words (superv. only)* with 200 samples), but only for the large training set sizes (100 and 200). Overall this would be a positive effect, with the impact being limited to one dataset. The effect is also limited to larger dataset sizes, which is not surprising given that having a lot more features than training samples is not ideal for training.

While additional features improve the results, the research question is specifically about the features typically used in natural language processing with deep learning, such as word embedding and word hashing. For the matching, however, neither of the word embedding and word hashing features improve the performance compared to the traditional bag of word features.

For the indexing algorithms the additional features improve results (e.g. improving the F1-score from 0.705 to 0.720 for *w. Bag-of-words* on dataset R with 200 samples), but this depends largely the right configuration. Purely adding the features is an improvement over the baseline, but adding features in combination with semantic hashing lowers results.

As with the matching algorithms, in indexing there also does not seem to be any reason to use the word embedding and word hashing features compared to the traditional bag of words features, as there were no differences in quality.

7.1 Future work

This research has focused on improving the performance of duplicate detection algorithms with small training sets by focusing at the use of unlabeled data, as well as the use of features such as word hashing and word embedding.

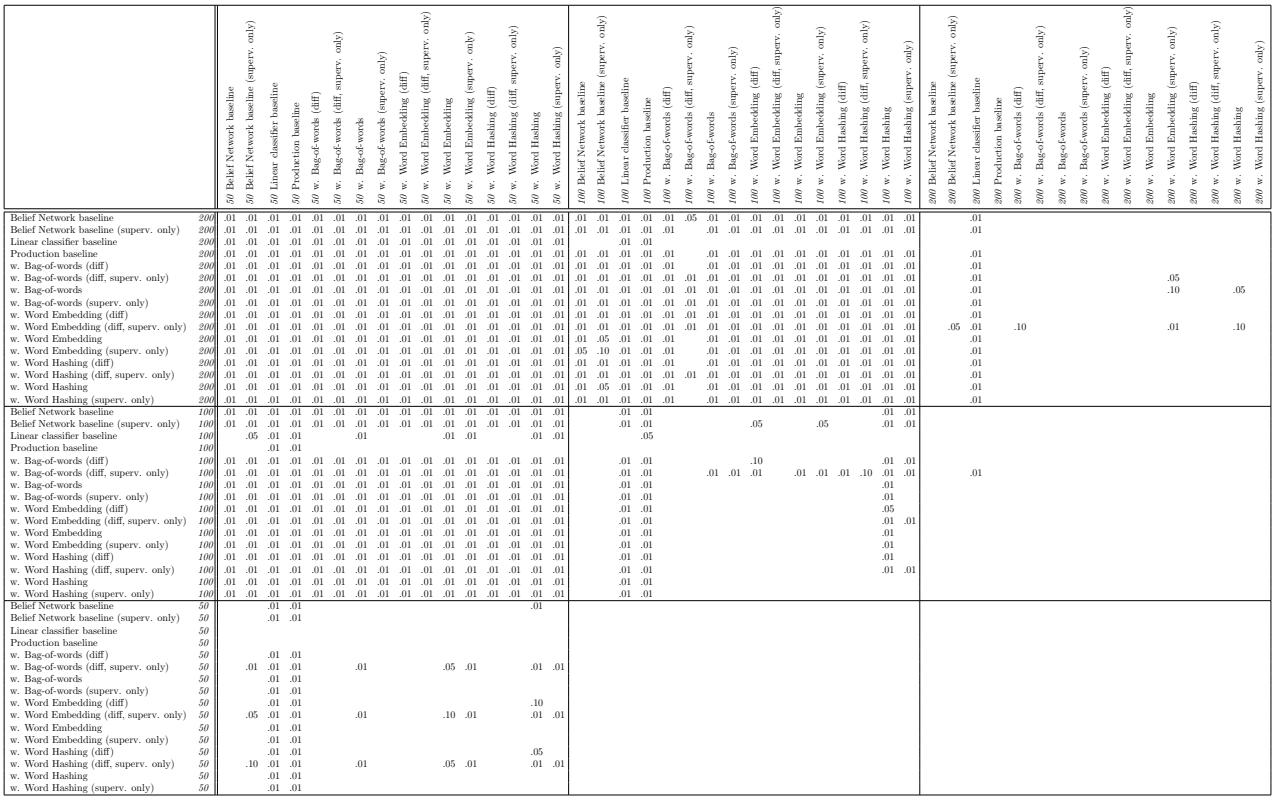
- In the evaluation it was assumed that the training set samples were selected randomly (out of the entire labeled training set). However, in the production system the training set is trained incrementally, in which data to be labeled is selected based on the previous labeled data. This process is called active learning [50]. Active learning techniques are likely another aspect that can be used to improve results in the production system.
- An important aspects that was not touched is regularisation. Regularisation is a method to prevent overfitting. As a lot of the variants in the evaluation had more parameters than examples, it is not surprising that a lot were suffering from overfitting (see Appendix B). Regularization is very likely to help these scenarios. In addition, the magnitude of regularization could be adjusted depending on the number of training samples: in scenarios with little training data more regularization is applied, in scenarios with a lot of training regularization is reduced.
- In the evaluation word embedding is used as an additional feature set in both matching and indexing. Currently this word embedding model is being trained on data from the dataset. Another possibility would be to use a pretrained word embedding model, trained using for example Wikipedia. Such a model would be more in-depth than the current model. This does require that the data has similarities to natural text, and in the same language as the word embedding model.
- In this evaluation the input features, such as the edit distances, were considered fixed and the additional neural network layers are placed on top of those features. An alternative possibility would be to create a deeper neural network by replacing the edit distances with neural network layers. These lower layers can be tasked with calculating edit distances, but unlike fixed edit distance algorithms, can adapt to the data. Errors from the higher layers can be propagated back into the lower layers tasked with calculating edit distances. Conceptually this would be similar to learnable edit distances [46], but integrated as part of a larger duplicate detection system.
- Another technique that could be useful in relation to learning edit distance features from scratch are recursive neural networks. Using recursive neural networks it would be possible to work directly on the input strings and have the network learn how to compare them.

A

Significance tables

This chapter contains the significance tables for the results listed in Chapter 5.

The significance overview for the indexing algorithms on dataset C are listed in Table A.4 and in Table A.3 for dataset R. The significance overview for the matching algorithms are listed in Table A.2 for dataset C and in Table A.1 for dataset R.



		50 Belief Network baseline	50 Belief Network baseline (superv. only)	50 Linear classifier baseline	50 Production baseline	50 w. Bag-of-words (diff)	50 w. Bag-of-words (diff, superv. only)	50 w. Bag-of-words	50 w. Bag-of-words (superv. only)	50 w. Word Embedding (diff)	50 w. Word Embedding (diff, superv. only)	50 w. Word Embedding	50 w. Word Embedding (superv. only)	50 w. Word Hashing (diff)	50 w. Word Hashing (diff, superv. only)	50 w. Word Hashing	50 w. Word Hashing (superv. only)	100 Belief Network baseline	100 Belief Network baseline (superv. only)	100 Linear classifier baseline	100 Production baseline	100 w. Bag-of-words (diff)	100 w. Bag-of-words (diff, superv. only)	100 w. Bag-of-words	100 w. Bag-of-words (superv. only)	100 w. Word Embedding (diff)	100 w. Word Embedding (diff, superv. only)	100 w. Word Embedding	100 w. Word Embedding (superv. only)	100 w. Word Hashing (diff)	100 w. Word Hashing (diff, superv. only)	100 w. Word Hashing	100 w. Word Hashing (superv. only)	200 Belief Network baseline	200 Belief Network baseline (superv. only)	200 Linear classifier baseline	200 Production baseline	200 w. Bag-of-words (diff)	200 w. Bag-of-words (diff, superv. only)	200 w. Bag-of-words	200 w. Bag-of-words (superv. only)	200 w. Word Embedding (diff)	200 w. Word Embedding (diff, superv. only)	200 w. Word Embedding	200 w. Word Embedding (superv. only)	200 w. Word Hashing (diff)	200 w. Word Hashing (diff, superv. only)	200 w. Word Hashing	200 w. Word Hashing (superv. only)
Belief Network baseline	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.05	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01													
Belief Network baseline (superv. only)	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01		.01	.01	.01	.01	.01	.01	.01	.01	.01			.01													
Linear classifier baseline	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01	.01																											
Production baseline	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01													
w. Bag-of-words (diff)	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01													
w. Bag-of-words (diff, superv. only)	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01					.05							
w. Bag-of-words	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01					.10		.05					
w. Bag-of-words (superv. only)	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01													
w. Word Embedding (diff)	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01													
w. Word Embedding (diff, superv. only)	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.05	.01	.01	.10				.01		.10					
w. Word Embedding	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.05	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01													
w. Word Embedding (superv. only)	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.05	.10	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01													
w. Word Hashing (diff)	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01													
w. Word Hashing (diff, superv. only)	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01													
w. Word Hashing	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.05	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01													
w. Word Hashing (superv. only)	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01													
Belief Network baseline	*100*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01	.01																									
Belief Network baseline (superv. only)	*100*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01	.01			.05			.05		.01															
Linear classifier baseline	*100*		.05	.01	.01			.01				.01	.01				.01					.05																							
Production baseline	*100*			.01	.01																																									
w. Bag-of-words (diff)	*100*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01	.01			.10																			
w. Bag-of-words (diff, superv. only)	*100*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01	.01			.01			.01	.01	.01	.01	.10	.01	.01											
w. Bag-of-words	*100*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01	.01											.01											
w. Bag-of-words (superv. only)	*100*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01	.01											.01											
w. Word Embedding (diff)	*100*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01	.01											.05											
w. Word Embedding (diff, superv. only)	*100*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01	.01											.01	.01										
w. Word Embedding	*100*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01	.01											.01											
w. Word Embedding (superv. only)	*100*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01	.01											.01											
w. Word Hashing (diff)	*100*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01	.01											.01											
w. Word Hashing (diff, superv. only)	*100*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01	.01											.01	.01										
w. Word Hashing	*100*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01	.01																							
w. Word Hashing (superv. only)	*100*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01			.01	.01																							
Belief Network baseline	*50*			.01	.01												.01																													
Belief Network baseline (superv. only)	*50*			.01	.01																																									
Linear classifier baseline	*50*																																														
Production baseline	*50*																																														
w. Bag-of-words (diff)	*50*			.01	.01																																									
w. Bag-of-words (diff, superv. only)	*50*		.01	.01	.01			.01				.05	.01			.01	.01																													
w. Bag-of-words	*50*			.01	.01																																									
w. Bag-of-words (superv. only)	*50*			.01	.01																																									
w. Word Embedding (diff)	*50*			.01	.01							.10				.01																													
w. Word Embedding (diff, superv. only)	*50*		.05	.01	.01			.01				.10	.01			.01	.01																													
w. Word Embedding	*50*			.01	.01																																									
w. Word Embedding (superv. only)	*50*			.01	.01																																									
w. Word Hashing (diff)	*50*			.01	.01											.05																													
w. Word Hashing (diff, superv. only)	*50*		.10	.01	.01			.01				.05	.01			.01	.01																													
w. Word Hashing	*50*			.01	.01																																									
w. Word Hashing (superv. only)	*50*			.01	.01																																									

Table A.1: Significance levels of the matching algorithms on dataset R.
Every cell contains the calculated *p-value*: a low value indicates that the algorithm listed on the left performs significantly better than the algorithm listed on the top. Values are rounded upwards to 0.01, 0.05 or 0.10. Values higher than 0.10 are omitted. All values have been corrected using the Holm-Bonferroni method.

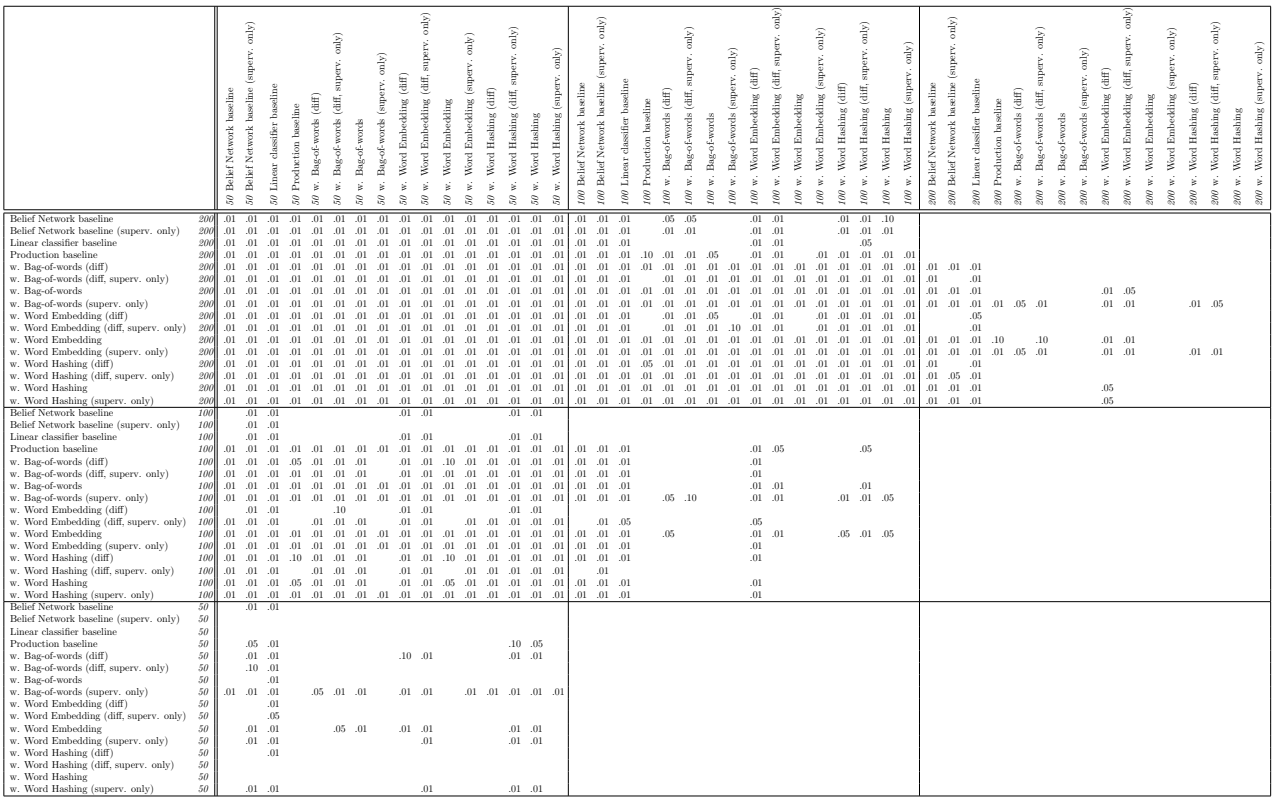


Table A.2: Significance levels of the matching algorithms on dataset C.
Every cell contains the calculated *p-value*: a low value indicates that the algorithm listed on the left performs significantly better than the algorithm listed on the top. Values are rounded upwards to 0.01, 0.05 or 0.10. Values higher than 0.10 are omitted. All values have been corrected using the Holm-Bonferroni method.

		50 Blocking baseline	50 Production baseline	50 w. Bag-of-words	50 w. Bag-of-words (semantic h.)	50 w. Bag-of-words (semantic h., superv. only)	50 w. Word Embedding	50 w. Word Embedding (semantic h.)	50 w. Word Embedding (semantic h., superv. only)	50 w. Word Hashing	50 w. Word Hashing (semantic h.)	50 w. Word Hashing (semantic h., superv. only)	100 Blocking baseline	100 Production baseline	100 w. Bag-of-words	100 w. Bag-of-words (semantic h.)	100 w. Bag-of-words (semantic h., superv. only)	100 w. Word Embedding	100 w. Word Embedding (semantic h.)	100 w. Word Embedding (semantic h., superv. only)	100 w. Word Hashing	100 w. Word Hashing (semantic h.)	100 w. Word Hashing (semantic h., superv. only)	200 Blocking baseline	200 Production baseline	200 w. Bag-of-words	200 w. Bag-of-words (semantic h.)	200 w. Bag-of-words (semantic h., superv. only)	200 w. Word Embedding	200 w. Word Embedding (semantic h.)	200 w. Word Embedding (semantic h., superv. only)	200 w. Word Hashing	200 w. Word Hashing (semantic h.)	200 w. Word Hashing (semantic h., superv. only)
Blocking baseline	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01		.01	.01	.01	.01	.05	.01	.01	.01	.01	.01						.01	.01				
Production baseline	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01		.01	.01	.01	.01	.01	.01	.01	.01	.01	.01						.01	.01				
w. Bag-of-words	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01		.01		.01	.01	.01	.01		.01
w. Bag-of-words (semantic h.)	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.05	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01		.01		.01	.01	.01			
w. Bag-of-words (semantic h., superv. only)	*200*	.10	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01		.01	.05	.01	.01		.01	.01	.01	.01	.01						.01	.01				
w. Word Embedding	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01		.01		.01	.01	.01	.01		.01
w. Word Embedding (semantic h.)	*200*		.01	.01	.01	.01	.01	.01	.01	.01	.01	.01		.01		.01	.01		.01	.01		.01	.01						.01					
w. Word Embedding (semantic h., superv. only)	*200*				.01	.01			.01		.01	.01				.01			.01			.01											
w. Word Hashing	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01		.01		.01	.01	.01	.01		.01
w. Word Hashing (semantic h.)	*200*	.05	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01		.01	.05	.01	.01		.01	.01	.01	.01	.01						.01	.01				
w. Word Hashing (semantic h., superv. only)	*200*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01		.01	.01	.01	.01	.01	.01	.01	.01	.01	.01						.01	.01				
Blocking baseline	*100*	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01		.01	.01	.01	.01	.01	.01	.01	.01	.01	.01						.01	.01				
Production baseline	*100*				.01				.01		.01	.01									.01													
w. Bag-of-words	*100*		.01	.01	.01	.01	.01	.01	.01	.01	.01	.01		.01		.01	.01		.01	.01		.01	.01							.01				
w. Bag-of-words (semantic h.)	*100*				.01				.01		.01	.01																						
w. Bag-of-words (semantic h., superv. only)	*100*				.01	.01			.01		.01	.01				.01					.01													
w. Word Embedding	*100*		.01	.01	.01	.01	.01	.01	.01	.01	.01	.01		.01		.01	.01		.01	.01		.01	.01							.01				
w. Word Embedding (semantic h.)	*100*		.01		.01	.01			.01	.01	.01	.01		.01		.01	.01			.01		.01	.01							.01				
w. Word Embedding (semantic h., superv. only)	*100*				.01				.01		.01	.01																						
w. Word Hashing	*100*		.01	.01	.01	.01	.05		.01	.01	.01	.01		.01		.01	.01		.01	.01		.01	.01							.01				
w. Word Hashing (semantic h.)	*100*				.01	.01			.01	.01	.01	.01		.05		.01				.01			.01											
w. Word Hashing (semantic h., superv. only)	*100*				.01				.01		.01	.01																						
Blocking baseline	*50*		.01	.01	.01	.01	.01	.01	.01	.01	.01	.01		.01		.01	.01		.01	.01		.01	.01							.01				
Production baseline	*50*				.01	.05			.01		.01	.01				.05				.01														
w. Bag-of-words	*50*		.01		.01	.01			.01	.01	.01	.01		.01		.01	.01			.01			.01							.01				
w. Bag-of-words (semantic h.)	*50*																																	
w. Bag-of-words (semantic h., superv. only)	*50*				.01				.01		.01	.01																						
w. Word Embedding	*50*		.01	.01	.01	.01			.01	.01	.01	.01		.01		.01	.01			.01		.01	.01							.01				
w. Word Embedding (semantic h.)	*50*		.01	.01	.01	.01			.01	.01	.01	.01		.01		.01	.01			.01		.01	.01							.01				
w. Word Embedding (semantic h., superv. only)	*50*																																	
w. Word Hashing	*50*				.01				.01		.01	.01								.01														
w. Word Hashing (semantic h.)	*50*								.05																									
w. Word Hashing (semantic h., superv. only)	*50*																																	

Table A.3: Significance levels of the indexing algorithms on dataset R.
Every cell contains the calculated *p-value*: a low value indicates that the algorithm listed on the left performs significantly better than the algorithm listed on the top. Values are rounded upwards to 0.01, 0.05 or 0.10. Values higher than 0.10 are omitted. All values have been corrected using the Holm-Bonferroni method.

		50 Blocking baseline	*50* Production baseline	*50* w. Bag-of-words	*50* w. Bag-of-words (semantic h.)	*50* w. Bag-of-words (semantic h., superv. only)	*50* w. Word Embedding	*50* w. Word Embedding (semantic h.)	*50* w. Word Embedding (semantic h., superv. only)	*50* w. Word Hashing	*50* w. Word Hashing (semantic h.)	*50* w. Word Hashing (semantic h., superv. only)	*100* Blocking baseline	*100* Production baseline	*100* w. Bag-of-words	*100* w. Bag-of-words (semantic h.)	*100* w. Bag-of-words (semantic h., superv. only)	*100* w. Word Embedding	*100* w. Word Embedding (semantic h.)	*100* w. Word Embedding (semantic h., superv. only)	*100* w. Word Hashing	*100* w. Word Hashing (semantic h.)	*100* w. Word Hashing (semantic h., superv. only)	*200* Blocking baseline	*200* Production baseline	*200* w. Bag-of-words	*200* w. Bag-of-words (semantic h.)	*200* w. Bag-of-words (semantic h., superv. only)	*200* w. Word Embedding	*200* w. Word Embedding (semantic h.)	*200* w. Word Embedding (semantic h., superv. only)	*200* w. Word Hashing	*200* w. Word Hashing (semantic h.)	*200* w. Word Hashing (semantic h., superv. only)
Blocking baseline	*200*	.01																														
Production baseline	*200*	.01											.01											.01					.01	.01		
w. Bag-of-words	*200*	.01			.01		.01	.01	.01	.01	.01	.01	.01	.01		.01	.01	.01	.01	.01			.10	.01	.01		.01	.01	.01	.01	.01		
w. Bag-of-words (semantic h.)	*200*	.01										.05	.01											.01					.01	.01		
w. Bag-of-words (semantic h., superv. only)	*200*	.01										.01	.01	.01		.01			.01	.01				.01	.01		.05			.01	.01	
w. Word Embedding	*200*	.01										.01	.01	.01		.05			.01	.01				.01	.01					.01	.01	
w. Word Embedding (semantic h.)	*200*	.01																														
w. Word Embedding (semantic h., superv. only)	*200*	.01																														
w. Word Hashing	*200*	.01	.01	.01	.01		.01	.01	.01	.01	.01	.01	.01	.01		.01	.01	.01	.01	.01			.01	.01	.01		.01	.01	.01	.01	.01		
w. Word Hashing (semantic h.)	*200*	.01			.01			.01			.01	.01	.01	.01		.01			.01	.01				.01	.01		.01			.01	.01	
w. Word Hashing (semantic h., superv. only)	*200*	.01			.01			.01	.10	.01	.01	.01	.01	.01		.01	.01	.01	.01	.01				.01	.01		.01	.01	.01	.01	.01		
Blocking baseline	*100*	.01																														
Production baseline	*100*	.01											.10											.10					.10	.10		
w. Bag-of-words	*100*	.01			.01			.01		.01	.01	.01	.01	.01		.01	.01	.01	.01	.01				.01	.01		.01	.05	.01	.01	.01		
w. Bag-of-words (semantic h.)	*100*	.01											.01											.01					.01	.01		
w. Bag-of-words (semantic h., superv. only)	*100*	.01										.01	.01	.01					.01	.01				.01					.01	.01		
w. Word Embedding	*100*	.01										.01	.01	.01		.01			.01	.01				.01	.01		.05			.01	.01	
w. Word Embedding (semantic h.)	*100*	.01																														
w. Word Embedding (semantic h., superv. only)	*100*	.01											.01											.01					.01	.01		
w. Word Hashing	*100*	.01			.01			.01		.01	.01	.01	.01	.01		.01	.01	.01	.01	.01				.01	.01		.01	.10	.01	.01	.01		
w. Word Hashing (semantic h.)	*100*	.01			.01			.01	.01	.01	.01	.01	.01	.01		.01	.01	.01	.01	.01			.05	.01	.01		.01	.01	.01	.01	.01		
w. Word Hashing (semantic h., superv. only)	*100*	.01										.01	.01	.01		.10			.01	.01				.01	.10					.01	.01	
Blocking baseline	*50*																															
Production baseline	*50*	.01					.01					.01	.01	.01		.01			.01	.01				.01	.01		.01			.01	.01	
w. Bag-of-words	*50*	.01			.05		.01				.10	.01	.01	.01		.01			.01	.01				.01	.01		.01		.10	.01	.01	
w. Bag-of-words (semantic h.)	*50*	.01										.01	.01											.01						.01	.01	
w. Bag-of-words (semantic h., superv. only)	*50*	.01			.05		.01				.10	.01	.01	.01		.01			.01	.01				.01	.01		.01			.01	.01	
w. Word Embedding	*50*	.01					.01					.01	.01	.01		.01			.01	.01				.01	.01		.01			.01	.01	
w. Word Embedding (semantic h.)	*50*	.01										.10																				
w. Word Embedding (semantic h., superv. only)	*50*	.01										.01	.01	.01					.01	.05				.01						.01	.01	
w. Word Hashing	*50*	.01										.01	.01	.01					.01	.01				.01	.10					.01	.01	
w. Word Hashing (semantic h.)	*50*	.01										.01																				
w. Word Hashing (semantic h., superv. only)	*50*																															

Table A.4: Significance levels of the indexing algorithms on dataset C.
Every cell contains the calculated *p-value*: a low value indicates that the algorithm listed on the left performs significantly better than the algorithm listed on the top. Values are rounded upwards to 0.01, 0.05 or 0.10. Values higher than 0.10 are omitted. All values have been corrected using the Holm-Bonferroni method.

B

Training set evaluation results

This chapter contains the results of training set evaluation. The results of the K-fold evaluation are listed in Chapter 5.

The results for the indexing algorithms on dataset C are listed in Table B.4 and in Table B.3 for dataset R. The results for the matching algorithms are listed in Table B.2 for dataset C and in Table B.1 for dataset R.

Notice that while normally a training set evaluation is a evaluation on the same data that was used for training, this is not fully the case here. Some of the algorithm variants use only a certain part of the available training set for training. So while the data used for training is present in the evaluation data, not all evaluation data has been used for training.

Algorithm	n	**Prec.**	**Rec.**	**F1**	**MCC**	**Corr.**
w. Bag-of-words	50	1.000	1.000	1.000	1.000	1.000
Belief Network baseline (superv. only)	50	1.000	1.000	1.000	1.000	1.000
w. Word Embedding	50	1.000	1.000	1.000	1.000	1.000
w. Word Embedding (superv. only)	100	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words (superv. only)	100	1.000	1.000	1.000	1.000	1.000
w. Word Hashing (diff, superv. only)	200	1.000	1.000	1.000	1.000	1.000
Belief Network baseline (superv. only)	100	1.000	1.000	1.000	1.000	1.000
Belief Network baseline	50	1.000	1.000	1.000	1.000	1.000
w. Word Embedding (diff, superv. only)	100	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words (diff, superv. only)	200	1.000	1.000	1.000	1.000	1.000
w. Word Hashing (superv. only)	50	1.000	1.000	1.000	1.000	1.000
w. Word Embedding (diff, superv. only)	200	1.000	1.000	1.000	1.000	1.000
w. Word Embedding (superv. only)	50	1.000	1.000	1.000	1.000	1.000
w. Word Hashing (diff, superv. only)	100	1.000	1.000	1.000	1.000	1.000
w. Word Hashing (superv. only)	200	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words (diff)	50	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words (superv. only)	50	1.000	1.000	1.000	1.000	1.000
w. Word Hashing (diff)	200	1.000	1.000	1.000	1.000	1.000
w. Word Embedding (diff)	50	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words (superv. only)	200	1.000	1.000	1.000	1.000	1.000
w. Word Embedding (diff, superv. only)	50	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words (diff, superv. only)	100	1.000	1.000	1.000	1.000	1.000
w. Word Hashing (diff)	50	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words (diff)	200	1.000	1.000	1.000	1.000	1.000
w. Word Embedding (superv. only)	200	1.000	1.000	1.000	1.000	1.000
w. Word Hashing	100	1.000	1.000	1.000	1.000	1.000
w. Word Embedding	100	1.000	1.000	1.000	1.000	1.000
w. Word Hashing (diff)	100	1.000	1.000	1.000	1.000	1.000
w. Word Hashing	200	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words (diff, superv. only)	50	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words	100	1.000	1.000	1.000	1.000	1.000
w. Word Hashing	50	1.000	1.000	1.000	1.000	1.000
w. Word Embedding (diff)	100	1.000	1.000	1.000	1.000	1.000
Belief Network baseline	100	1.000	1.000	1.000	1.000	1.000
w. Word Embedding	200	1.000	1.000	1.000	1.000	1.000
w. Word Hashing (diff, superv. only)	50	1.000	1.000	1.000	1.000	1.000
w. Word Hashing (superv. only)	100	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words (diff)	100	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words	200	1.000	1.000	1.000	1.000	1.000
Belief Network baseline	200	1.000	1.000	1.000	1.000	1.000
Belief Network baseline (superv. only)	200	1.000	1.000	1.000	1.000	1.000
w. Word Embedding (diff)	200	.999	1.000	.999	.998	.999
Linear classifier baseline	50	.996	.998	.997	.994	.997
Linear classifier baseline	100	.986	.983	.984	.969	.985
Production baseline	200	.980	.978	.979	.959	.979
Linear classifier baseline	200	.966	.972	.969	.938	.969
Production baseline	50	.981	.950	.965	.932	.966
Production baseline	100	.974	.943	.958	.918	.959

Table B.1: Training set evaluation results of the matching algorithms on dataset R. The algorithms are trained using with differing amounts of training data (n). For each algorithm the overall precision, recall, F1-score, Matthews correlation coefficient (MCC) and correctness rate over all runs are reported.

Algorithm	n	**Prec.**	**Rec.**	**F1**	**MCC**	**Corr.**
w. Bag-of-words	50	1.000	1.000	1.000	1.000	1.000
Belief Network baseline (superv. only)	50	1.000	1.000	1.000	1.000	1.000
w. Word Embedding	50	1.000	1.000	1.000	1.000	1.000
w. Word Embedding (superv. only)	100	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words (superv. only)	100	1.000	1.000	1.000	1.000	1.000
w. Word Hashing (diff, superv. only)	200	1.000	1.000	1.000	1.000	1.000
Belief Network baseline (superv. only)	100	1.000	1.000	1.000	1.000	1.000
Belief Network baseline	50	1.000	1.000	1.000	1.000	1.000
w. Word Embedding (diff, superv. only)	100	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words (diff, superv. only)	200	1.000	1.000	1.000	1.000	1.000
w. Word Hashing (superv. only)	50	1.000	1.000	1.000	1.000	1.000
w. Word Embedding (diff, superv. only)	200	1.000	1.000	1.000	1.000	1.000
w. Word Embedding (superv. only)	50	1.000	1.000	1.000	1.000	1.000
w. Word Hashing (diff, superv. only)	100	1.000	1.000	1.000	1.000	1.000
Belief Network baseline	200	1.000	1.000	1.000	1.000	1.000
w. Word Hashing (superv. only)	200	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words (diff)	50	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words (superv. only)	50	1.000	1.000	1.000	1.000	1.000
w. Word Hashing (diff)	200	1.000	1.000	1.000	1.000	1.000
w. Word Embedding (diff)	50	1.000	1.000	1.000	1.000	1.000
Belief Network baseline (superv. only)	200	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words (superv. only)	200	1.000	1.000	1.000	1.000	1.000
w. Word Embedding (diff, superv. only)	50	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words (diff, superv. only)	100	1.000	1.000	1.000	1.000	1.000
w. Word Hashing (diff)	50	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words (diff)	200	1.000	1.000	1.000	1.000	1.000
w. Word Embedding (superv. only)	200	1.000	1.000	1.000	1.000	1.000
w. Word Hashing	100	1.000	1.000	1.000	1.000	1.000
w. Word Embedding	100	1.000	1.000	1.000	1.000	1.000
w. Word Hashing (diff)	100	1.000	1.000	1.000	1.000	1.000
w. Word Hashing	200	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words (diff, superv. only)	50	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words	100	1.000	1.000	1.000	1.000	1.000
w. Word Hashing	50	1.000	1.000	1.000	1.000	1.000
w. Word Embedding (diff)	100	1.000	1.000	1.000	1.000	1.000
Belief Network baseline	100	1.000	1.000	1.000	1.000	1.000
w. Word Embedding	200	1.000	1.000	1.000	1.000	1.000
w. Word Hashing (diff, superv. only)	50	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words (diff)	100	1.000	1.000	1.000	1.000	1.000
w. Bag-of-words	200	1.000	1.000	1.000	1.000	1.000
w. Word Embedding (diff)	200	1.000	1.000	1.000	1.000	1.000
w. Word Hashing (superv. only)	100	.999	1.000	1.000	.999	1.000
Linear classifier baseline	50	.984	.996	.990	.980	.990
Linear classifier baseline	100	.973	.992	.982	.964	.982
Linear classifier baseline	200	.958	.987	.972	.944	.972
Production baseline	100	.969	.971	.970	.940	.970
Production baseline	50	.961	.974	.967	.934	.967
Production baseline	200	.963	.969	.966	.932	.966

Table B.2: Training set evaluation results of the matching algorithms on dataset C. The algorithms are trained using with differing amounts of training data (n). For each algorithm the overall precision, recall, F1-score, Matthews correlation coefficient (MCC) and correctness rate over all runs are reported.

Algorithm	n	**Prec.**	**Rec.**	**F1**	**MCC**	**Corr.**
w. Word Hashing	50	.909	.844	.876	.762	.880
w. Bag-of-words (semantic h., superv. only)	50	.867	.864	.866	.732	.866
w. Word Hashing (semantic h., superv. only)	50	.938	.780	.852	.738	.864
w. Bag-of-words	50	.884	.806	.843	.703	.850
w. Word Hashing (semantic h.)	50	.915	.756	.828	.697	.843
w. Bag-of-words (semantic h.)	50	.830	.858	.844	.682	.841
w. Word Hashing	100	.914	.725	.809	.672	.829
w. Word Hashing (semantic h., superv. only)	100	.901	.726	.804	.659	.823
w. Word Embedding (semantic h., superv. only)	50	.876	.750	.808	.651	.822
w. Bag-of-words (semantic h., superv. only)	100	.844	.773	.807	.632	.815
w. Word Embedding	50	.834	.782	.807	.627	.813
w. Bag-of-words (semantic h.)	100	.826	.779	.802	.616	.807
w. Word Hashing	200	.904	.689	.781	.633	.807
w. Bag-of-words	100	.875	.702	.779	.614	.801
w. Word Hashing (semantic h.)	100	.856	.712	.777	.601	.796
w. Bag-of-words (semantic h., superv. only)	200	.869	.694	.772	.601	.794
w. Word Embedding (semantic h.)	50	.862	.700	.773	.599	.794
Blocking baseline	50	.841	.722	.777	.592	.793
w. Bag-of-words (semantic h.)	200	.868	.690	.769	.598	.793
w. Bag-of-words	200	.870	.687	.768	.597	.792
w. Word Hashing (semantic h., superv. only)	200	.906	.647	.755	.606	.790
w. Word Embedding	100	.823	.728	.772	.575	.785
w. Word Hashing (semantic h.)	200	.857	.671	.753	.573	.780
w. Word Embedding	200	.862	.663	.750	.572	.778
w. Word Embedding (semantic h., superv. only)	100	.845	.675	.750	.562	.775
w. Word Embedding (semantic h.)	100	.849	.666	.747	.561	.774
w. Word Embedding (semantic h., superv. only)	200	.850	.663	.745	.560	.773
w. Word Embedding (semantic h.)	200	.825	.688	.750	.549	.771
Blocking baseline	100	.827	.681	.747	.548	.769
Blocking baseline	200	.826	.680	.746	.546	.768
Production baseline	200	.812	.697	.750	.540	.767
Production baseline	50	.748	.758	.753	.502	.751
Production baseline	100	.723	.701	.712	.432	.716

Table B.3: Training set evaluation results of the indexing algorithms on dataset R. The algorithms are trained using with differing amounts of training data (n). For each algorithm the overall precision, recall, F1-score, Matthews correlation coefficient (MCC) and correctness rate over all runs are reported.

Algorithm	n	**Prec.**	**Rec.**	**F1**	**MCC**	**Corr.**
w. Word Hashing	50	.938	.906	.922	.846	.923
w. Word Hashing (semantic h.)	50	.947	.866	.905	.821	.909
w. Word Hashing (semantic h., superv. only)	50	.959	.838	.894	.808	.901
w. Bag-of-words	50	.874	.926	.899	.793	.896
w. Word Hashing	100	.864	.908	.885	.766	.882
w. Bag-of-words (semantic h., superv. only)	50	.923	.814	.865	.751	.873
w. Bag-of-words	100	.839	.909	.873	.738	.868
w. Bag-of-words (semantic h.)	50	.907	.818	.860	.738	.867
w. Word Hashing	200	.827	.906	.865	.720	.859
w. Word Embedding	50	.819	.908	.861	.712	.854
w. Word Hashing (semantic h.)	100	.884	.809	.845	.706	.852
w. Bag-of-words	200	.822	.896	.857	.705	.851
w. Word Hashing (semantic h., superv. only)	100	.872	.817	.844	.698	.849
w. Bag-of-words (semantic h.)	100	.803	.893	.846	.678	.837
w. Bag-of-words (semantic h., superv. only)	100	.827	.847	.837	.670	.835
w. Word Embedding	100	.786	.909	.843	.669	.831
w. Word Embedding (semantic h.)	50	.842	.808	.824	.657	.828
w. Word Hashing (semantic h.)	200	.810	.847	.828	.649	.824
w. Word Embedding	200	.775	.905	.835	.652	.821
w. Word Hashing (semantic h., superv. only)	200	.834	.799	.816	.640	.820
w. Word Embedding (semantic h., superv. only)	50	.881	.728	.797	.640	.815
w. Bag-of-words (semantic h.)	200	.778	.878	.825	.634	.814
w. Bag-of-words (semantic h., superv. only)	200	.786	.862	.822	.629	.813
w. Word Embedding (semantic h.)	100	.763	.877	.816	.612	.802
Blocking baseline	50	.746	.914	.821	.618	.801
Blocking baseline	100	.732	.926	.818	.609	.793
w. Word Embedding (semantic h., superv. only)	100	.754	.869	.807	.592	.792
w. Word Embedding (semantic h., superv. only)	200	.730	.923	.816	.604	.791
w. Word Embedding (semantic h.)	200	.730	.923	.816	.604	.791
Blocking baseline	200	.730	.923	.816	.604	.791
Production baseline	100	.742	.860	.797	.568	.780
Production baseline	200	.741	.850	.792	.559	.776
Production baseline	50	.754	.754	.754	.508	.754

Table B.4: Training set evaluation results of the indexing algorithms on dataset C. The algorithms are trained using with differing amounts of training data (n). For each algorithm the overall precision, recall, F1-score, Matthews correlation coefficient (MCC) and correctness rate over all runs are reported.

Bibliography

[1] E. Alpaydin. *Introduction to machine learning*. MIT press, 2014.

[2] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, 2003.

[3] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *Knowledge and Data Engineering, IEEE Transactions on*, 24(9):1537–1555, 2012.

[4] W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string metrics for matching names and records. In *Knowledge Discovery and Data Mining workshop on data cleaning and object consolidation*, volume 3, pages 73–78, 2003.

[5] R. Collobert. Deep learning for efficient discriminative parsing. In *International Conference on Artificial Intelligence and Statistics*, number EPFL-CONF-192374, 2011.

[6] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[7] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

[8] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967.

[9] G. Dahl, D. Yu, L. Deng, and A. Acero. Context-dependent dbn-hmms in large vocabulary continuous speech recognition. In *Proc. ICASSP*, 2011.

[10] N. Dalvi, V. Rastogi, A. Dasgupta, A. Das Sarma, and T. Sarlós. Optimal hashing schemes for entity matching. In *Proceedings of the 22nd international conference on World Wide Web*, pages 295–306. International World Wide Web Conferences Steering Committee, 2013.

[11] L. Deng, X. He, and J. Gao. Deep stacking networks for information retrieval. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 3153–3157. IEEE, 2013.

[12] L. Deng, M. Lennig, F. Seitz, and P. Mermelstein. Large vocabulary word recognition using context-dependent allophonic hidden markov models. *Computer Speech & Language*, 4(4):345–357, 1990.

[13] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923, 1998.

[14] L. O. Evangelista, E. Cortez, A. S. da Silva, and W. Meira Jr. Adaptive and flexible blocking for record linkage tasks. *Journal of Information and Data Management*, 1(2):167, 2010.

[15] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[16] R. Fernandez, A. Rendel, B. Ramabhadran, and R. Hoory. F0 contour prediction with a deep belief network-gaussian process hybrid model. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6885–6889. IEEE, 2013.

[17] G. Forman and M. Scholz. Apples-to-apples in cross-validation studies: pitfalls in classifier performance measurement. *ACM SIGKDD Explorations Newsletter*, 12(1):49–57, 2010.

[18] J. Gao, X. He, W.-t. Yih, and L. Deng. Learning semantic representations for the phrase translation model. *arXiv preprint arXiv:1312.0482*, 2013.

[19] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[20] I. Gonçalves and S. Silva. Experiments on controlling overfitting in genetic programming. In *Proceedings of the 15th Portuguese Conference on Artificial Intelligence: Progress in Artificial Intelligence, EPIA*, volume 84, 2011.

[21] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[22] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6645–6649. IEEE, 2013.

[23] L. Hamers, Y. Hemeryck, G. Herweyers, M. Janssen, H. Keters, R. Rousseau, and A. Vanhoutte. Similarity measures in scientometric research: the jaccard index versus salton's cosine formula. *Information Processing & Management*, 25(3):315–318, 1989.

[24] G. Hinton and R. Salakhutdinov. Discovering binary codes for documents by learning deep generative models. *Topics in Cognitive Science*, 3(1):74–91, 2011.

[25] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[26] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 2333–2338. ACM, 2013.

[27] S. Janssen. Comparison of blocking and field encoding techniques for duplicate detection in contact data. 2013.

[28] S. Kang, X. Qian, and H.-Y. Meng. Multi-distribution deep belief network for speech synthesis. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8012–8016. IEEE, 2013.

[29] L. Kolb, A. Thor, and E. Rahm. Dedoop: efficient deduplication with hadoop. *Proceedings of the VLDB Endowment*, 5(12):1878–1881, 2012.

[30] M. Kolkman and S. Janssen. Automated duplicate detection in contact data. 2013.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[32] Q. V. Le. Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8595–8598. IEEE, 2013.

[33] R. Lebret and R. Collobert. Word emdeddings through hellinger pca. *arXiv preprint arXiv:1312.5542*, 2013.

[34] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[35] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[36] Z.-H. Ling, L. Deng, and D. Yu. Modeling spectral envelopes using restricted boltzmann machines and deep belief networks for statistical parametric speech synthesis. *Audio, Speech, and Language Processing, IEEE Transactions on*, 21(10):2129–2139, 2013.

[37] B. W. Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975.

[38] M. Michelson and C. A. Knoblock. Learning blocking schemes for record linkage. In *Proceedings Of The National Conference On Artificial Intelligence*, volume 21, page 440. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[39] T. Mikolov. Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April*, 2012.

[40] T. M. Mitchell. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45, 1997.

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

[42] N. Morgan. Deep and wide: Multiple layers in automatic speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):7–13, 2012.

[43] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[44] A. Nikolov, M. d'Aquin, and E. Motta. Unsupervised learning of link discovery configuration. In *The Semantic Web: Research and Applications*, pages 119–133. Springer, 2012.

[45] D. M. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

[46] E. S. Ristad and P. N. Yianilos. Learning string-edit distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(5):522–532, 1998.

[47] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[48] R. Salakhutdinov and G. Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[49] S. L. Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data mining and knowledge discovery*, 1(3):317–328, 1997.

[50] B. Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52(55-66):11, 2010.

[51] R. Socher, C. D. Manning, and A. Y. Ng. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, pages 1–9, 2010.

[52] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 151–161. Association for Computational Linguistics, 2011.

[53] N. Srivastava and R. R. Salakhutdinov. Multimodal learning with deep boltzmann machines. In *Advances in neural information processing systems*, pages 2222–2230, 2012.

[54] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 111–147, 1974.

[55] K. Xu, J. Ba, R. Kiros, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015.

[56] H. Ze, A. Senior, and M. Schuster. Statistical parametric speech synthesis using deep neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7962–7966. IEEE, 2013.

[57] W. Y. Zou, R. Socher, D. M. Cer, and C. D. Manning. Bilingual word embeddings for phrase-based machine translation. In *EMNLP*, pages 1393–1398, 2013.