Normalization of extracted named-entities in text mining

Biswas, U.

Award date:
2019

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Architecture of Information Systems Research Group

Normalization of Extracted Named-Entities in Text Mining

Master Thesis

Upasana Biswas

Thesis Committee:
Dr. Ekaterini Ioannou
Prof. Johannes C. Scholtes
Dr. Renata Carvalho

Eindhoven, August 2019

Abstract

Normalization of entities is one of the most important tasks in text mining. The entities extracted from raw text can be of any type, ranging from personal names, company names to product descriptions. The entities can also have several textual variations for each individual real-world object. Normalization aims to recognize such variations and link them to that one object they refer to. The crux of normalizing these entities is that the solution should consider several factors like abbreviations, acronyms, spelling mistakes, context insensitivity, length variations and nick names (in case of personal names). The thesis attempts to provide a general solution in matching extracted entities, with the focus on personal names and company names, taking into consideration all the aforementioned factors. The approach starts with implementing the soft TF-IDF algorithm, which is a combination of edit-based and token-based measures. It then moves on to custom rules-based machine learning approaches with Support Vector Machines and XgBoost. The rules evolve from domain specific to being more generalized. The F1 scores vary from 80% to 95%. In addition, deep learning techniques based on a Siamese Convolutional Neural Network (CNN) has been used and experimented with different character embeddings. The evaluation of the models, various issues faced and blocking techniques used for performance optimization have been discussed in details in the later section of the thesis. The overall result is text data normalized into clusters of different entities that contain the real-world entity name for each similar occurrence.

Preface

The thesis is part of my curriculum in Data Science Master's. It is a documentation of my work done as an intern in ZyLAB from February to July 2019. It presents the research aspects of my work in text normalization.
For the guidance and help, I would like to express my gratitude and appreciation to Prof. Johannes Scholtes and Jeroen Smeet for their unconditional help. I would also like to thank Dr. Ekaterini Ioannou for her invaluable advice and feedback at every step.
It was a fun filled and true learning experience doing my internship, reading numerous papers, implementing methods and writing my thesis.

Contents

Chapter 1

Introduction

Human and machine generated data is expected to have a fifty-fold [1] growth rate from 2010 to 2020, globally. The collection and analysis of the data into meaningful insights is important in business processes, for the companies to stay competitive. But most of the produced data is in the form of unstructured text with typographical errors, standardized as well as non-standardized abbreviations and even in different representations. The consequences of the described variations are duplicates in text, that the data extraction techniques are unable to distinguish without the help of special methods or training. Entity name resolution comes in this scenario to address the problem of mapping different textual variations to their actual representative entities.

An *entity* is a data unit that represents a real-world object like person, place, organization and so on. These entities, present in raw text or structured database, are identified, extracted and classified into their corresponding real-world object categories. For each instance in a category, there are duplicates with variations, which are normalized by linking them together and represented as one, unique instance.

Entity resolution (ER) consists of two tasks: entity name matching and entity name clustering. Given a collection of entities and an entity e, *entity matching* (EM) finds the entity in the collection that best matches the entity e or produces a list of ranked entities based on matching score with the entity e. It helps establish the fact that whether the entities refer to the same real-world object, denoted by e here. *Entity clustering* groups strings or entities into their corresponding normalized real-world object names. The results from an entity matching system can be utilized to form clusters among similar entities based on a cut-off distance value, that is tuned manually.

To start with the real world problems in text, a person or a company can be referred to with the full proper name or an alias name or a short form of name in different places of a document. E.g. an email to 'William Scott' might have 'Bill Scott' or 'W. Scott' as the addressed names in the body of the email. Now, the fact that 'W. Scott' is same as 'William Scott' can be treated as a simple string matching problem or a classification problem with the output being a match or non-match. But classifying 'Mr. William Scott' and 'Bill Scott' being the same person, i.e. a match, needs an additional information of first-name to nick-name mapping. However, 'W. Scott' can result in multiple matches with people having their first names starting with 'W' and last name as 'Scott'. In that case, the context around the entity is important to link it to the right person. In this thesis the contextual information has not been considered while evaluating a match and in connection to it, the disambiguation problem of pointing 'W. Scott' as a person and not a company, is also not being addressed here. The approach simply focuses on normalizing entities in database records with mitigating effects of textual errors. So, the task of pairwise matching 'W. Scott' and 'William Scott' is *entity matching* and establishing the fact that 'W. Scott', 'William Scott' and 'Bill Scott' are in the same group, is *entity clustering*.

[1] https://www.siliconrepublic.com/enterprise/50-fold-increase-in-the-size-of-data-globally-between-2010-and-2020-infographic

There are a number of ways to tackle the EM problem. One of the better performing methods is Term Frequency-Inverse Document frequency (TF-IDF) based on 'Bag of words' method, which works without any prior domain knowledge. It is **robust**, in the sense that it can be used for entity matching in any kind of noisy real-world data where word order and position vary. E.g. Phillip M. Kent and Kent Phillip are the same person, for which traditional string matching methods like 'Levenshtein' fails [2], with a similarity score 14%. One such method soft TF-IDF, introduced in (Cohen et al., 2003), is a string matching method which can also be used as an unsupervised clustering method. It has been shown to outperform most of the classical string similarity measures in matching task.

A second method [5], which is more **reliable** but less flexible, involves studying the features of the data and then use an algorithm or tool for learning and evaluation. This approach takes the advantage of combining domain specific knowledge (use of dictionary for nick-name mapping in personal names) along with the features conspicuous in the data. E.g. a predominant feature in Dutch company names entities is the use of the term 'BV'. A learning model such as Support Vector Machine (SVM) or XgBoost, is trained with the extracted features of the data and their known labels. The learned model is later used for classifying unlabelled data. It is a supervised problem and needs manual intervention in labelling the data.

This thesis aims to combine the **robustness** of 'Bag of words' model (TF-IDF) with the **reliability** of feature-based learning algorithms and develop a holistic method for the normalization of entities, extracted by text mining. So, for the extracted lists of named entities such as *person*, *organization* and *location*, the goal is to create three new entities *normalized_person*, *normalized_organization* and *normalized_location* that contain for each original occurrence the normalized (real-world) entity name.

The research work, next, extends to developing a character based Siamese Convolutional Neural Network (SCNN) in Deep Learning (DL), given DL's success in solving normalization problems . The training of the model has been done with two kinds of character level embedding - one hot encoding and lately introduced contextual embedding in 'Flair'. Two kinds of distance metrics, namely L1 norm and cosine distance have been experimented with to observe the differences in performance.

The problem addressed in this thesis is an imbalanced classification problem. The two outcomes are - 'match' and 'non-match', with 'non-match' category representing the overwhelming majority of the data points, for a particular entity, In order for the model to be reliable, a 'few' false positives in the result are still acceptable but not too many false negatives. This is because a falsely detected match can be rejected manually but it is not possible to search the false negatives in the huge number of negatives detected. Thus the model evaluation is done emphasizing more on *Precision* metrics than *Recall* by calculating *Average Precision*. *Average Precision* calculates the sum of the ratios of true positives to predicted positives cumulatively after every prediction and divides it with the total number of ground truth positives rather than total number of predictions, which gives a better representation of the model's performance in case of presence of a few ground truth positives. The *F1 score* has also been calculated to observe the balance between *Precision* and *Recall*.

In addition to the primary goals of this thesis, an attempt has been made to make the approach scalable and reduce the processing time. The unsupervised nature of the bag-of-words technique necessitates N×N (N = number of data points) comparisons, which leads to deteriorating execution time with bigger dataset and longer data points. A blocking approach, based on Locality Sensitive Hashing [3] technique, has been implemented for normalizing personal names. The execution time improved along with some useful prevention of errors.

[2] https://medium.com/@appaloosastore/string-similarity-algorithms-compared-3f7b4d12f0ff
[3] https://towardsdatascience.com/understanding-locality-sensitive-hashing-49f6d1f6134

Variations		Causes
William Kent	William Gary Kent	Middle-names
	W. G. Kent	Acronyms
	Bill Kent/ Willie Kent	Nick-names
	William Knet	Optical Character Recognition (OCR) induced errors or spelling mistakes
	Williâm Ként	Different alphabet

Table 1.1: An example of variations for a person name

1.1 Problem Description

Entity Resolution has been recognized as the base technology for removing duplicates in database and promoting data and information quality in text. The first step used in ER is to identify if two entities refer to the same instance. This thesis aims at determining the entity equivalence, without considering the co-referent information, disambiguation or the context around the entities. Here, the entities extracted are personal names and company names and their variations in text.

For *personal names*, the variations occur due to several factors. Table 1.1 describes the different occurrences of names and their corresponding error types, e.g. the cause 'Nick-names' is responsible for two different variations of the first name 'William' to 'Bill' and 'Willie'. The task is to link or normalize all the variants to one particular name, in this case 'William Kent', chosen based on the requirement.

For *company names*, rather than errors, the variations occur predominantly due to representational differences. In Table 1.2, all the rows correspond to many subsidiaries of one company 'TXU'. While the first row says the company name in general, the second and third rows are acronyms and full forms. While it is considerably easier to detect acronyms in person names, here, matching 'LSP' with 'Lone Star Pipeline' is a challenge due to absence of white-spaces in the acronyms as well as very different other half of the tokens. However, the notable characteristic is that the first 'few' tokens always contain the original company name. Selective weighing of the tokens gives better results but with some induced errors, which are discussed in the later part of the thesis.

Table 1.2: An example of variations for a company name

TXU Corp.
TXU LSP
TXU Lone Star Pipeline Systems
TXU Energy Retail Company
TXU Europe Energy Trading Ltd.

The *application* of ER can be illustrated very simply by Figure 1.1. With growing volume and velocity of data, semantic relationships between entities become complex and somewhat ambiguous due to duplicates of the same entity. ER can solve the problem by canonicalizing references to particular entities. In Figure 1.1, normalization reduced the complexity of a fourth order network to a second order one, by normalizing the two variations 'W. Kent' and 'William Kent' to 'William Kent' and 'Bob Price' and 'Robert Price' to 'Robert Price'.

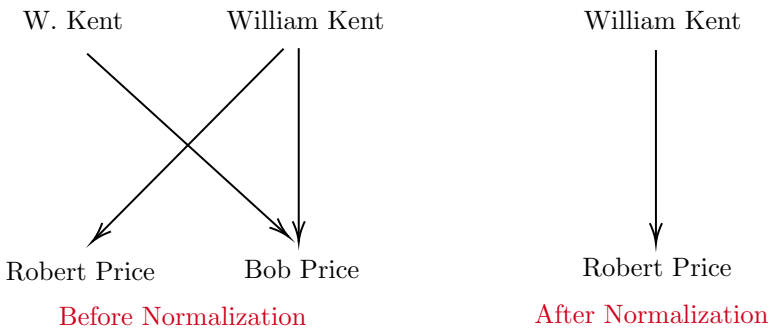


Figure 1.1: Before and after normalization of person names

1.2 Research Questions

The research questions this thesis aims to address, are formulated keeping in mind the problem specificity, intention and feasibility with respect to dataset availability.

- **Is it possible to provide a solution to text normalization which will work independent of the characteristics of the data?** The answer to this question lies in precise construction of features which do not rely too heavily on the data characteristics. While classical machine learning models are able to learn these features and classify, the importance lies in cost of the performance at which this domain independence is achieved.
- **How do the model performances vary with characteristics, pre-processing and class ratio in the dataset?** With differences in the nature of entities and huge class imbalance in real-world application of text normalization, the model performances vary noticeably.
- **Is Deep Learning capable of further improving the performances by machine learning models?** The difference in performance of machine learning models trained with features, and deep learning approaches trained without features, is worth observing in this step.
- **How to collect or prepare labelled dataset?** Preparation of dataset comes with a number of essential prerequisites like correct distribution of noise, imitating the real world text as well as class balance.
- **How to optimize the performance with suitable blocking approaches?** The blocking approach discussed in this thesis is not implemented keeping in mind domain independence but is very specific for personal names data.

Chapter 2

Literature Review

This chapter consists of two sections broadly. Section 2.1 gives a comprehensive overview of the concepts behind the different methods and models used in the experiments in this thesis. It starts with explaining string matching methods to machine learning models like SVM, XgBoost and finally, CNNs in deep learning. Section 2.2 provides a discussion of some of the interesting work done on text normalization with their importance and relevance to this thesis.

2.1 Background

Given the work already done in this domain of text mining, we, now, provide a brief overview of the different models and concepts related to the experiments done as part of the research work.

2.1.1 String-Matching Techniques

Several string-matching techniques have been suggested which tackle different kinds of string matching problems. While edit-based techniques, like Levenshtein, and token-based techniques, like TF-IDF or cosine similarity, have their own pros and cons, the best performing methods are the hybrid ones, one of the most prominent examples of which is soft TF-IDF. Here, Levenshtein, Jaccard and soft TF-IDF techniques have been discussed briefly with their applications, advantages and disadvantages.

Edit-Based Matching Techniques

Most of the edit-based string matching methods work on the sequential position of characters in the strings. They measure the similarity between two strings in terms of a distance d, which represents the minimum number of edit operations like insertions, deletions, substitutions and transpositions needed to transform a string to another. Some of the very well known methods, used in the experiments in this thesis, are discussed below:

- *Levenshtein.* It gives the dissimilarity d between the strings x and y, with number of edit operations required for the transformation. The corresponding similarity score is defined as $s = 1 - d/max(|x|, |y|)$.E.g. *d(Harry, Bary)* = 2 (h $\rightarrow$ b (substitution), r $\rightarrow$ r (deletion)). The computed value of s is 0.6 i.e. 60% similarity computed.

- *Jaccard.* Jaccard similarity treats two strings as sets of letters and measures the similarity as the size of the intersection of the sets divided by the size of the union of the two sets. The score between the strings x and y, is computed by the formula, $(|x \cap y|)/(|x \cup y|)$

Token-Based Matching Techniques

- *Soft TF-IDF.* This technique was introduced by Cohen *et al.*, 2003 [1] and improvised by Moreau *et al.*, 2008 [2]. It is a 'soft' version of TF-IDF method where the similarity score *dist* between the strings S and T is calculated by taking into account only the most similar tokens in $S \cap T$ along with their TF-IDF values. The soft similarity between tokens are calculated by a suitably chosen edit-based string similarity measure. The efficiency of the algorithm is determined by it's ability to replace the requirement for words to be equal with requirement to be similar.

  Let S and T be two strings. Let *CLOSE(θ,S,T)* be the set of words $w \in S$ such that there exists a word $v \in T$, such that *dist(w, v)* $> \theta$. Let *N(w,T)* $= \max(\{dist(w,v) | v \in T\})$. For any $w \in$ *CLOSE(θ,S,T)*, let

  $$S(w, S, T) = weight(w, S).weight(v, T).N(w, T)$$

  where,

  $$weight(w, Z) = \frac{tfidf(w, Z)}{\sqrt{\sum_{w \in Z} tfidf(w, Z)^2}}$$

  Finally,

  $$soft\_TFIDF(S, T) = \sum_{w \in CLOSE(\theta, S, T)} S(w, S, T)$$

  In our experiments, *Levenshtein* has been used as the secondary similarity measure.

  The *advantages* of this technique are:

  - **Robustness** - It is capable of handling varying word order, spelling mistakes and other noise because of token-to-token based calculation. E.g. 'Apple Corporation' and 'Corp. Aple' represent the same entity.
  - **Impact of common words** - Since relative frequencies of words are taken into account, highly common tokens are given less importance computationally. E.g. in the names, 'Will van der Smeet' and 'Richard van der Smeet', low TF-IDF values for the words 'van' and 'der' reduce the overall score.
  - **Flexibility** - Features can be added conveniently to tune the 'soft' part of the calculation. E.g. while calculating distance between tokens, features such as 'prefix', 'substring' can be added to improvise the score.

  The *disadvantages* of this technique are:

  - **Dependency on string lengths** - For longer strings with majority of the tokens matching, the algorithm fails to find the match or mismatch. E.g. 'Indian oil and petroleum corporation ltd.' and 'Bharat petroleum and oil corporation ltd.' are different entities but results in a faulty higher score due to too many exactly matching tokens.
  - **Sensitivity to length variation** - For two strings 'John' and 'John F. Kennedy', only one pair of tokens {'John', 'John'} match and the score drops due to vector normalization. The bigger string has length three, so normalizing the overall score with square root of sum of the TF-IDF values of three words takes a heavy toll.
  - **Computational overhead** - Due to token to token based calculation between every strings and then comparison between one string to every string in the dataset, the execution time is very long.

2.1.2 Support Vector Machines

A Support Vector Machine [16] is a supervised machine learning algorithm, used for both classification and regression purposes. It is formally defined by a separating hyper-plane which categorizes data points. In simple words, given labelled training data, the algorithm finds the optimal hyper-plane that best divides the data into classes.

Support Vectors and Margin

Support vectors are the data points which lying nearest to the hyper-plane. Intuitively, the further from the hyper-plane the data points lie, the more precisely has the dataset been classified. Margin is the distance between the hyper-plane and the support vectors from either sides. A good margin is one where this separation is larger for both the classes. And the importance of the support vectors lies in maximizing the margin from the hyper-plane while classifying the data points.

Kernel

SVMs can be implemented in multiple ways based on the nature of data. The goal of a kernel is to take the input data and transform it into the required mathematical function. Kernels can be of different types: *linear*, *polynomial*, *radial basis function (RBF)* and *sigmoid*. A linearly separable data can be classified by a linear kernel in a two dimension plane. But for non-linearly separable data, it is necessary to move from 2D plane to a multi-dimensional view. This process of mapping the data to higher and higher dimensions until a suitable hyper-plane can be formed to segregate it, is done by different kinds of kernel functions, based on the distribution of the data. As seen in Figure 2.1, the data points are to be classified as car or bicycle based on two parameters - age and income. Clearly, in an attempt to classify linearly, the first graph misclassifies four points. The second one, with polynomial kernel is better but misclassifies three points. The best performance is achieved by the third on using Gaussian kernel.

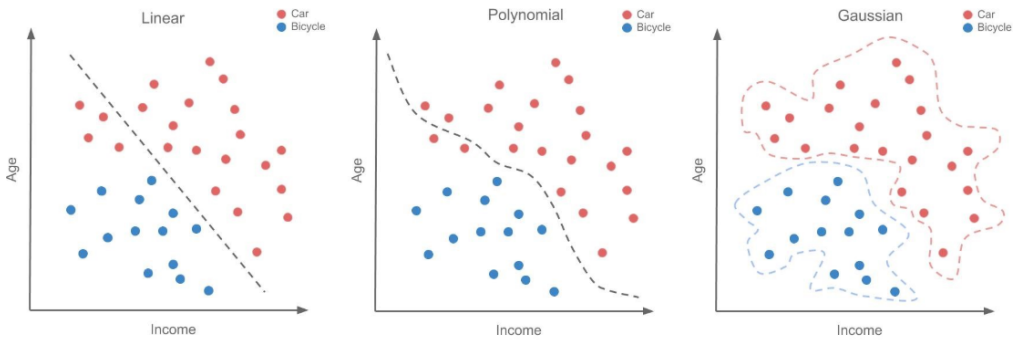


Figure 2.1: Performance of three kinds of SVM kernels on a non-linearly separable data

Regularization

The Regularization parameter, often called the 'C' parameter in python 'sklearn' library determines the level of SVM optimization to avoid misclassification. For larger values of C, the optimization chooses a small margin hyper-plane so as to classify the nearby point properly. A smaller value of C asks the optimizer to look for a large margin hyper-plane with the cost of a few data points being misclassified.

2.1.3 XgBoost

XgBoost or Extreme Gradient Boosting [13] is a decision tree based ensemble machine learning algorithm that uses gradient boosting framework. It is an optimized, flexible and distributed

gradient boosting library. It is used to solve classification, regression and ranking and user-defined prediction problem. The following parts describe the working logic of the algorithm in details.

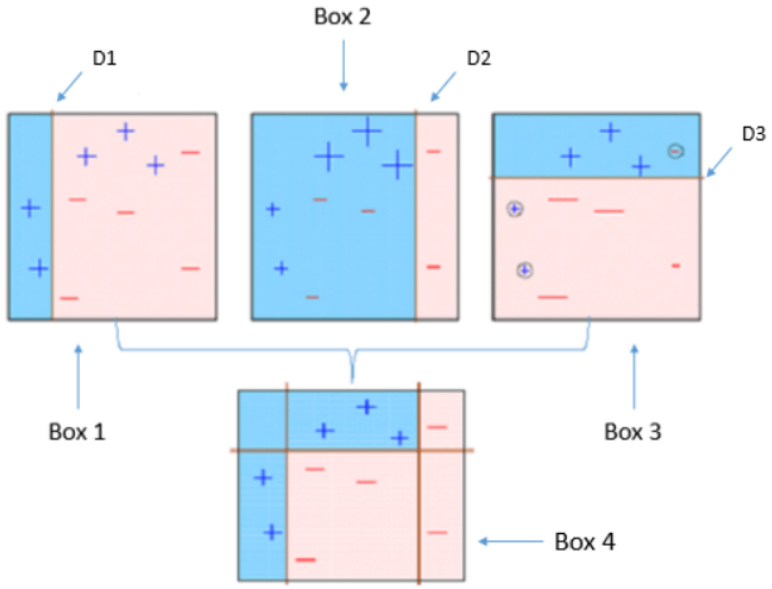


Figure 2.2: Cumulative improvement in classification of + and - by four decision tree based classifiers in four boxes

Decision Trees and Boosting

A decision tree is a tree like structure built by continuously splitting the data based on certain data features or parameters. It has two entities, decision nodes - where the data is split based on the feature, and decision leaves - the outcome of the split.
Boosting on decision trees is the process of starting with weak base decision tree based model and then using the conclusion derived from the weak model to build a new, stronger model by capitalizing on the errors or misclassifications of the previous weak model and trying to reduce them iteratively.
Gradient Boosting is a special case of boosting where the errors from the previous learners are minimized by gradient-descent algorithm.

Gradient Descent

Gradient Descent is an optimization algorithm used to minimize the error of a prediction model by iteratively calculating it and updating the parameters of the model until it reaches the lowest prediction error. The error of a model is given by it's cost function. So by computing the gradient of the function, the slope of the error curve is determined. The parameters of the function are now updated so that the value of the new loss moves along the direction of the slope calculated and finally reaches a minima.

Intuition

An XgBoost is also an ensemble tree method that works on the principle of boosting weak learners using gradient descent architecture. E.g. in figure 2.2, classifier D1 misclassifies three + points. D2 gives more importance on the misclassifications of D1, corrects it but end up misclassifying three - points. D3 works on D2 but fails to classify accurately. Box 4 is the weighted combination of the previous weak classifiers D1, D2 and D3 and manages to classify correctly.

XgBoost is even superior than general Gradient Boosting Machines (GBM) because of algorithmic enhancements and hardware and software optimizations. The factors which contribute to its supremacy are:

- Efficient handling of missing data
- Parallel tree building
- Regularization for overfitting avoidance
- Cache awareness and out-of-core computing

2.1.4 SVM and XgBoost for String-Matching Problem

Internally, both SVM and XgBoost models represent a problem as regression predictive-modeling problem, that only takes numerical values as input. To classify the strings as a match or non-match, requires representing them as numerical features. This representation or encoding is domain specific and there is no general way to build a representation, which will work across every text dataset. In the implementations discussed in this thesis, the dataset has been arranged into pairs of strings with the corresponding label of match or non-match. Every pair of strings is represented as an array of features. So, for a training set consisting of i data points, a single data point refers to a pair of strings with a label. For each of these pairs, j number of features are computed, resulting in an array of size $[i \times j]$. The feature set can consist of a combination of similarity scores, which are essentially float values as well as generated attributes (if a pair contains a common prefix, substring or an acronym) in Boolean forms (1 - Yes, 0 - No). E.g. for this matching string pair 'Phil Kent' and 'Phillip K Gary', multiple edit based similarity measures such as cosine-distance, Levenshtein and Jaro are applied and the feature set is an array of the distances computed between them i.e. [0, 0.43, 0.7]. The SVM and XgBoost learn to build the classification function based on the values in the feature set and use that function to classify data during testing.

2.1.5 Siamese Convolutional Neural Network

The Siamese network [17] is an architecture for non-linear metric learning with similarity information. A Siamese Convolutional Neural Network is a Convolutional neural network (CNN) with Siamese architecture i.e. a network containing two identical sub-network components. It ideally looks like figure 2.3, with two inputs for two sub-networks, which share the same architecture as well as model weights. The main idea behind Siamese networks is that they naturally learn the invariant and selective representations in the data, directly through the use of similarity and dissimilarity information fed to it through training pairs. The input pairs can be numerical data, images or sequential text data. The output of a Siamese network is usually a binary classification,

Figure 2.3: Architecture of a Siamese Convolutional Neural Network

if the inputs are of the same class or not. The data features are learnt in the CNN units, the working of which has been described briefly along with the relevant loss functions used.

CNN for Sequential Data

For sequential data, CNNs work in one dimension (1D). A 1D CNN effectively extracts features from fixed length sequential data, where the feature order in the segment is not so important. It typically performs two operations on the inputs, *convolution* and *pooling*. Convolution is performed on the input data with the use of a filter or kernel, slid over the matrix representation of the data, to produce a feature map. Pooling is used to reduce the dimensionality of the feature maps, created by convolution, in terms of number of parameters and computation in the network. The following part describes the way these operations work on textual data.

Given a sequence of words, $w_{1:n} = w_1,....w_n$, where each word is encoded into a fixed dimension d vector and a window size of k is selected as the number of characters or words allowed in each input segment. A 1D convolution of width-k is moving a sliding window of size k over the sequence i.e. a dot-product between the concatenation of the embedding vectors and a weight vector u, which is then often followed by a non-linear activation function g.
Considering a window k of words w_i,....$\mathrm{w}_i + k$, the concatenated embedding vector is:

$$x_i = [w_i, w_{i+1}....w_{i+k}] \in R^{k \times d} \tag{2.1}$$

the convolution filter is applied resulting in scalar values r_i is:

$$r_i = g(x_i.u) \in R \tag{2.2}$$

Generally, for one convolution layer, more than one filters are applied, $u_1.....u_2$, which can then be represented as a vector multiplied by a matrix U and with an addition of a bias term b:

$$r_i = g(x_i.U + b) \tag{2.3}$$

$$with, r_i \in R^l, x_i \in R^{k \times d}, U \in R^{k.d \times l}, b \in R^l$$

The pooling operation is then used to combine the vectors resulting from different convolution windows into a single l-dimensional vector. It is done by taking the max or the average value observed in resulting vector from the convolutions. Ideally this vector will capture the most relevant features of the sentence. This vector is then fed further down in the network to a full connected layer to perform prediction.

For the Siamese network used in the experiments, the two character encoded inputs are convoluted, max-pooled and flattened at the end of the 'Dense' layer to fixed length vectors. The distance functions used after the 'Dense' layer are briefly described in the next section. The distance functions calculate element-wise distance between the two vectors and are used to calculate the loss. The loss function used is 'Binary Cross Entropy' loss, since it is a binary classification problem. The overall loss calculated is used to update the model for training.

Distance Functions

- *L1 distance.* L1 norm, also known as Manhattan Distance (Eq. 2.4), is the sum of absolute difference of the components of two vectors, where all the components of the vector are weighted equally.

$$d(x, y) = \sum_{i=1}^{k} |x_i - y_i| \tag{2.4}$$

  In the model, the L1 norm is used to calculate the element wise distances between the two encoded vectors produced after convolution, max-pooling and 'Flatten' layers. The values are passed onto the 'Dense' layer with 'Sigmoid' activation to squeeze into a single 0 or 1, which is the model prediction.

- *Cosine distance.* Cosine distance is the normalized dot product of two vectors, as given in the Equation 2.5:

$$d(x, y) = cos(\theta) = \frac{x.y}{\| x \| \| y \|} \tag{2.5}$$

  In the model, the cosine distance is computed between the two encoded vectors. The distance, being a singular value, is passed directly to the output as prediction of the model.

2.1.6 Blocking

A very prominent obstacle in the task of de-duplication or string matching is computational bottleneck. With n number of entities, brute-force approaches use all-to-all comparisons to cluster into similar strings. This requires $O(n^2)$ comparisons which could be huge depending on the value of n, leading to performance degradation with respect to both time and precision. To avoid that, the number of comparisons must be reduced drastically, without compromising the accuracy.

Blocking involves partitioning the records into mutually exclusive blocks based on some key, depending on the nature of the data. Now comparisons are made between records either within each block or between certain blocks, which are selected based on some function of the blocking key. Among the many traditional methods used for constructing blocking partitions, the one used in this thesis is inspired from Locality-Sensitive Hashing (LSH) approach. LSH is quite application specific and can be implemented depending on the domain of the data. LSH manages to ensure that the blocks are manageable small by using all of the information contained in each record. This method still suffers from two major drawbacks. Firstly, it needs domain knowledge to form the hashing key for partitioning. Secondly, the resulting blocks can still be large enough to not reduce the computational complexity considerably. A *blocking* approach for personal names has been discussed in Section 5.5 along with the improvements achieved in the overall performance.

2.2 Related Work

This section discusses some of the relevant work done in ER, focused on text normalization. Section 2.2.1 explains the implementations and comparisons of several string matching techniques, some of which are used in this thesis directly, based on the results observed in these papers. Section 2.2.2 describes the various attempts made to develop machine learning models for string similarity measurements. The features of the strings used in this thesis are partly inspired from these work. Section 2.2.3 is related to deep learning models used for string matching. The SCNN used in this thesis for string matching is based on the results of the approaches discussed here.

2.2.1 String Matching Measures

The literature on string comparison metrics is abundant, consisting of reviews on edit-based, token-based and hybrid string matching techniques. All of these techniques, have complementary strengths and weaknesses depending on the dataset. Some related surveys are:
Cohen *et al.*, 2003 [1] tested, measured and compared several string distance metrics on text data without taking into account the domain knowledge, with the help of their library SecondString. They concluded that the best performing individual approach is a hybrid distance metric on words called Soft TF-IDF, which has been discussed in details in the following parts of the thesis.

Erwan Moreau *et al.*, 2008 [2] combined several measures to formulate a more generic model. They proposed a mix of cosine-like similarity (soft TF-IDF) with Levenshtein-like alignment. The resultant method called *Meta-Levenshtein* (ML), is not a bag-of-words method but it takes into account Inverse Document Frequencies (IDF) of the text. ML performs comparably better and consistent with respect to the traditional string matching methods. It was capable of overcoming some of the pitfalls of soft TF-IDF like normalization, hard threshold tuning and sub-measure selection. Consequently, the performance of the method becomes invariant to the user choices.

Peter Christen, 2006 [3] did an extensive study on issues, characteristics and potential sources of variations and errors in personal names as database records. Apart from comparing the performances of classical string matching techniques, he included phonetic encoding based name matching techniques. While phonetic encoding with pattern matching failed to perform as expected, the overall conclusion he drew is, the dataset decides the performance of a certain technique and there is no single best technique available.

2.2.2 Machine Learning Models

Incorporation of domain-dependent and domain-independent rules in training a model to determine a match/non-match is one of the best ways to get high accuracy results in string matching. Some of the relevant works are presented below in the following paragraphs.
Bilenko and Mooney, 2002 [4] proposed a domain independent method for duplicate detection in database records. The approach involves two steps: *i)* Training similarity metrics to identify duplicate values in each field of a database record and *ii)* Combining multiple similarity metrics calculated for each field to learn a decision function for row level duplicate detection. A Support Vector Machine (SVM) was trained with matrices of field-to-field basis similarity metrics, corresponding to each table record. Significant performance improvement was achieved over fixed-cost string distance metrics on most of the different kinds of datasets used, proving the model's invariance to data specific features.

Emiel Caron *et al.*, 2016 [5] worked on clustering of organizational names database based on similarities in their associated meta data. They created a set of rules targeting specific elements of the organizations' characteristics such as organization type, country, city, postal code. Each rule is given a manual score either individually or in combination with other rules. First, record pairs are created based on the aforementioned characteristics, considering the string similarity. Then the rules are applied on the pairs to compute the score. Pairs with scores above a threshold are clustered by means of single-linkage, hierarchical clustering. Precision and recall performance values are calculated cluster wise per scientific organization. The cluster with the highest value of F1 measure is taken as the best cluster. The best cluster obtained in the dataset of University name variations he used, has an average precision of 0.95 and a recall of 0.80. Although unsupported by facts, the authors also conclude that this score based clustering approach can be used for other data types as long as relevant meta data is available.

Tejada *et al.*, 2002 [6] developed an object identification system called 'Active Atlas' to identify matching objects. It consists of two different stages. The first stage applies general string transformation rules like acronym, abbreviation etc. to compute a set of mapping rules based on the labelled data provided. The output is a set of attribute level similarity scores for a pair of strings along with the necessary features. This serves as the basis for the learning to begin. The next stage determines the importance of the selected features in the prior stage and fine tunes them iteratively. To make up for the highly inaccurate similarity score calculated initially, it employs an active supervised learning that refines both the mapping rules and transformation weights to capture the precise relationship between the objects. The system managed to achieve high accuracy over four domains of very 'limited' size dataset, without much user intervention.

Borggrewe *et al.*, 2016 [14], in an attempt to generate a domain independent string matching solution, uses the string matching scores of several well known string matching methods like Levenshtein, Jaccard etc. and combines them in a voting algorithm in SVM. The quality of the matching is evaluated against a number of labelled datasets. SVM outperformed in all the cases in comparison to the individual string matching methods. In order to show the real-world benefit of the voting algorithm, normalized entities derived from the 'Enron' data set are clustered and visualized and the differences between the original and the normalized data are observed. When compared to other string matching algorithms, the results observed for SVM have significantly lower number of duplicate references to real-world entities and improved quality of clustering.

2.2.3 Deep Learning Approaches

While the above approaches were developed aiming at domain independence, yet formulation of features is somewhat necessary for a consistent and good performance. Deep learning rules this out by capturing local character-level features in the strings. Several models have been experimented with comparable performances, but due to the use of different set-ups, application and datasets, they cannot be directly compared and concluded to be the one best approach. The deep learning models implemented in this thesis take inspiration from all of these and try to solve the problem of text normalization without features. Some of the most promising and pertinent models are discussed here:
Deepak Gottapu *et al.*, 2016 [7] uses a hybrid human-machine approach for entity resolution of products records. The machine part includes a one layered CNN which takes the records as fixed length word embedded vectors and gives an output as probability scores for each labels. Once the entire dataset gets a label (highest probability) with the trained CNN, all the records under each label are taken as identical records. Hereafter, crowdsourcing comes to play on the records with predicted labels. With the help of simple queries, records under a certain label are extracted and only those are sent to the crowds which have slightly varying probabilities assigned over more than one labels, which implies the confusion of the model in giving a hard decision. To solve these uncertainties, a certain threshold for probabilities is chosen, suitable record pairs created and sent to the crowd for reference. The machine labelling part here only reached an accuracy of 53% for correct classification, which, the authors say, can be improved by adding more layers to the CNN model. They reported much better overall results when compared to using Jaccard similarity. The benefit of this approach is that it saves a lot of time when compared to entity resolution done by string matching techniques on big datasets, the fact remaining that with expanding data size, the errors in human assisted sorting part will only exacerbate.

Zhe Gan *et al.*, 2017 [8] used a combination of a character level deep 'conflation' model with cosine similarity to determine a string match. The conflation model consists of two parts *i) a deep feature extractor* and *ii) a ranker*. The deep feature extractor transforms the character encoded input strings into a fixed dimension vector, which are the extracted deep feature vectors. After obtaining the vectors for the query string and each of the target output strings, the pairwise semantic relevance scores are calculated in the form of cosine similarity. During test time, the query-output strings are ranked according to the scores. Finally, a score cut-off is selected to include the relevant entities for the query string. The deep feature extractor was used as both an LSTM and CNN. CNN outperformed LSTM on the proprietary business dataset used but overall, the authors proves that deep learning models worked far better than traditional models.

Kooli *et al.*, 2018 [9] worked on record linkage, where record pairs are represented by n-gram based word embedding in order to tackle the out-of-dictionary problem. Three deep neural network models: *i) Multi Layer Perceptron (MLP) ii) Long short term memory (LSTM)* and *iii) Convolutional Neural Network (CNN)* are trained with labelled record pairs and give a binary classification as output. Evaluated on two different sets of data, LSTM outperforms the other two models on professional database and CNN outperforms in the scientific publications data. Also, compared to the classical string comparison methods like Jaro, Levenshtein and machine learning algorithms like SVM, Naive Bayes, the deep neural networks performed much better. The considerable improvement and consistency in performance show the ability of deep learning to handle non-normalized text.

Paul Neculoiu *et al.*, 2016 [10] presented a Siamese architecture for the task of job title normalization. The model is a stack of character level bidirectional LSTMs with 'contrastive loss' calculated pairwise. The model learns to represent fixed length character encoded strings into a finite dimension embedding space by using information about syntactic and semantic similarities between the strings. The model was trained iteratively with dataset augmentation at every step by addition of each of the features like spelling mistakes, synonym addition, extra word injection and facilitation of manual intervention. The performance of the Siamese model and an n-gram based baseline model are recorded in all the four stages of training and testing. While the baseline

model performed comparably with the LSTMs at the first step, the performance dropped sharply at each stage with addition of noise. The iterative training and testing proved that the Siamese model remained invariant to several features. Another interesting takeaway from this approach is that deep learning models can be naturally scalable with insensitivity to the input distribution of the data.

A very interesting approach proposed by Yoon Kim *et al.*, 2014 [15] in sentence classification is the use of multi channel CNN model with simple layers. The model consists of two set of vectors, each set of vectors is treated as a channel. Filters are applied to both the channels but gradients are back-propagated only through one of the channels, hence fine tuning one set of vectors while keeping the other static. The intuition behind this architecture is that it will be capable of avoiding overfitting by ensuring that the learned vectors do not deviate much from the original values. He compared the performance of this model with three other architectures (random, static and non-static) and observed mixed but competitive results depending on the dataset. He reported considerable performance improvement with the use of pre-trained word vectors, which in our case is not much relevant.

2.2.4 Blocking

Although, performance optimization is not the focus of my thesis, yet, the blocking attempt made to optimize the performance of the models, is derived and improvised based on two most relevant literature discussed here.
Jure Leskovec *et al.*, 2010 [11] presents a very detailed description on traditional blocking approaches like LSH, which is the most relevant technique for record linkage problems.

Rebecca Steorts *et al.*, 2014 [12] compared between traditional, hash based and cluster based blocking approaches. Different aspects of the blocking methods are discussed taking into consideration their simplicity, domain knowledge requirement for implementation, time complexities, robustness and sensitivity to noisy data. All of the approaches are run on different kinds of dataset and the recall rate varies accordingly. Two datasets are use: *RLData500* and *'noisy' dataset.* While the first one is a synthetic database of 500 records with 10 percent duplicates, the 'noisy' dataset has been built by a data generation and corruption tool, with different field attributes like gender, city etc. On the noisy data, two variants of LSH technique performed better than traditional approaches, whereas on 'RLData500', traditional blocking approaches worked the best. The authors concluded that each of these methods rely heavily on the application characteristics.

Chapter 3

Methods for Normalization of Extracted Named-Entities

This chapter provides a summary of all the methods implemented for matching the named-entities. It starts with a section on pre-processing of the strings, which is different for personal names and company names. It then gives a brief overview of how the methods or models are used for matching purpose. Each of the models are trained independently, evaluated on the same sets of testing data and then their performances compared. The expected end result for all the methods is getting a match (1) or no-match (0), for a pair of strings. The chapter ends with the blocking approach experimented with, for performance optimization. The following sections elaborate the entire process in details.

3.1 Pre-processing

The pre-processing of the data depends on it's language and domain and is essentially a determining factor in good performance of the models. In text, names come in all forms and format, with and without noise. Some of the common pre-processing steps are required for both personal names and company names. Additionally, some extra steps are also applied depending on the noise and type of the data. Likewise, for personal names, a dictionary has been constructed, which covers a wide range of names extracted from Wikipedia, for mapping nick names. The implementations are done, keeping in mind, primarily personal names and organization names. They are discussed below:

- **[Common]** Stripping punctuation and extra white spaces, because in non-contextual string-matching, punctuation do not have any significance in determining the similarity.
- **[Common]** Lower casing the strings so that the models do not differentiate between 'Aron' and 'aron'.
- **[Common]** De-duplication of strings, e.g. 'Karen Ng Ng' → 'Karen Ng'
- **[Personal Names]** Removal of any token in a string which contains a number, e.g. George the **3rd**.
- **[Personal Names]** Normalization of all other alphabets to English alphabet. E.g. Schônbrunne → Schonbrunne
- **[Personal Names]** Removal of designations like 'Dr.', 'Phd', 'Jr.' etc because occurrences of these words are occasional and they influence the overall score by changing the length of the strings, without adding any information.

- [**Company Names**] Replacing abbreviations of certain words like 'corp' → 'corporation', 'ltd' → 'limited'. This step is required because the models identify them as different words despite the fact that they are the same ones.

3.2 Soft TF-IDF

Soft TF-IDF is an unsupervised approach, with no training phase. For a list of entities, it compares each item to every other items and calculates a similarity score between 1 to 100. The 'soft' calculation part between the tokens in the strings has been done by Levenshtein distance metric. A pair of strings are taken to be a match if the score is above a cut-off. The cut-off score has been selected carefully by observing the number of false negatives and false positives on increasing or decreasing it. Notably, the cut-off varies widely depending on the nature of the dataset.
Soft TF-IDF is first applied on unlabelled 'Enron' personal names and company names datasets. For company names, the matches and mis-matches generated have been further used to build a small, labelled company names evaluation dataset with limited human efforts. The performance evaluation of the algorithm has been carried out by applying it on various labelled datasets.
For company names, the performance of the algorithm is improvised by giving more weights to the first one to two tokens in the strings due to the observations explained in Section 4.2.

3.3 Machine Learning Models

This section describes how machine learning models such as SVM and XgBoost have been used to train with the labelled strings. The functional aspects of these models and the means adopted to counteract their abilities to train with only numerical data have been discussed in details in Section 2.1.4. The different feature vectors used for string representation has been discussed in the following part in details.

3.3.1 String Representation

Since both SVM and XgBoost can train only on numerical data, the strings are represented numerically, with or without the features. The *distance-metrics-array* does not include any string specific features but consists of multiple similarity scores obtained from different string-matching techniques on each pair. The *features-as-Boolean-values* and *features-as-percentages* represents the strings with their common features, either as Boolean values or in proportions.

1. **Distance metrics array:** The string distance metrics measure string similarities taking into account different factors. So five different metrics - Levenshtein, Jaro, Jaro-Winkler, Jaccard, and cosine-similarity are taken to calculate the distances. E.g. for the string pair 'Phillip K. Allen' and 'Allen Phillip K.', the corresponding distances computed by the five metrics are [0.19, 0.70, 0.70, 1.0, 1.0]. Thus, every string pairs are converted into an array of distances and used as the training data.

2. **Features as Boolean values:** Since a matching pair of strings may contain a number of variations in the form of initials, prefixes and nick-names, they are represented here taking into account all the variations alongside the similarity scores, resulting in seven different parameters. E.g. the string pair 'William Henry Codd' and 'Bill H. Codd' can be represented as:

   Prefix. 0 → There is no prefix in the names

   Initials. 1 → {H, Henry}

   Dictionary. 1 → {Bill, William}

   Subset. 1 → Determines if one string is a subset of another

   Levenshtein. 0 → If similarity score is less than 0.6, else 1

Cosine. 0 → If similarity score is less than 0.75, else 1

Soft TF-IDF. 0 → If similarity score is less than 0.6, else 1

So, the feature set for this pair of strings translates to [0, 1, 1, 1, 0, 0, 0].

3. **Features as percentages:** Rather than representing the variations as a mere presence or absence i.e. with a Boolean value of 1 or 0 respectively, here, the proportions of token pairs contributing to each type variation in the strings, are calculated with respect to the total number of token pairs, and included in the features set. E.g. the same string pair 'William Henry Codd' and 'Bill H. Codd' with three pairs of tokens possible, can be represented as:

   Equality. 1/3 → {Codd, Codd}

   Initials. 1/3 → {H, Henry}

   Dictionary. 1/3 → {Bill, William}

   Prefix. 0 → There is no prefix in the names

   Dropped pairs. 0 → If there are no token pairs left after computing the above three parameters

   So, the feature set for this pair of strings translates to [0.3, 0.3, 0.3, 0, 0].

3.3.2 SVM and XgBoost

The SVM has been trained with all three kinds of string representations described above and evaluated for both balanced and imbalanced dataset. Two kernels were tested with - Linear and Radial, with different values of regularization parameter C. For *distance-metrics-array*, linear kernel performs better and the performance improves considerably with increase in C value from 1 to 5, after which the performance gains flattens out. For *features-as-Boolean-values* and *features-as-percentages* representations, radial kernel maintains a very good and stable performance with all values of C.

The XgBoost model has been implemented in a similar way to the SVM. The only tuning parameter experimented with is the max-depth of the decision trees, which gave the best performance for the value 6. Since the real world application of text normalization involves very large datasets' size, XgBoost, with it's ability to execute the code in parallel, takes comparatively less time to train on the data.

Both the models are trained on labelled person names dataset. However, the models are evaluated on class-balanced and class-imbalanced personal names testing set as well as three other datasets of different domains, to assess their generalization capabilities.

3.4 Siamese Convolutional Neural Network

Since the problem of text normalization relies on both semantic and syntactic similarity, deep learning models are expected to work successfully in this problem. The Siamese model implemented here is believed to be capable of handling data heterogeneity, noise and language insensitivity in the input string pairs by capturing the local correlation between the characters with automatic feature extraction.
To train a deep learning model, huge amount of training dataset is required. As discussed in 4.1, the augmented and labelled person names dataset has been used for training. The training set has been split into 80% training and 20% validation sets. Testing has been done on both balanced and imbalanced person datasets.
To investigate the effects of different kinds of embeddings, the models are trained with different embedding configurations. One is simple one-hot character encoding where a character set of 37 letters are chosen. The second one is pre-trained, contextual character embedding from the library 'Flair' [18]. The input strings are made of fixed length characters. Representing them as encoded

vectors produces arrays of size [m×n], where m is the length of the strings and n is the embedding vector length. They are fed in pairs with corresponding labels of 1 or 0.
A number of model architecture has been experimented with by varying the numbers of convolution layers, max-pooling size and regularizers. Two kinds of distance functions, L1 norm and cosine distance, are used. The performance with L1 distance is consistent and very good with a simple architecture of one to two convolution layers. On the other hand, training with cosine distance as the distance function generates a biased and high variance model with widely fluctuating accuracy scores over the testing sets. However, the variance decreases with adding more layers but still under performing than L1 norm. Figure 3.1 shows the basic architecture of the the model.

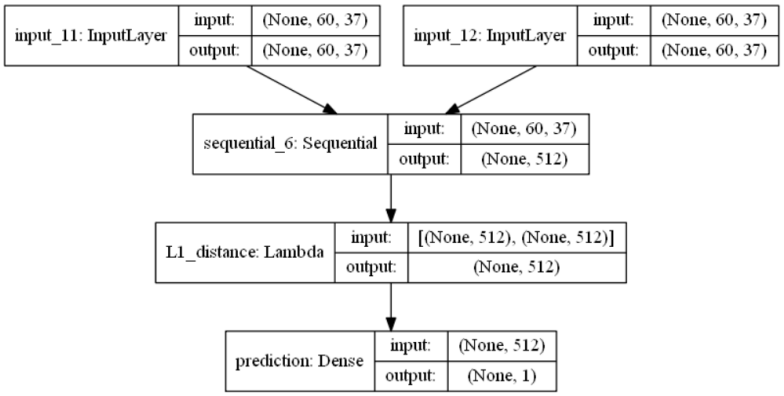


Figure 3.1: Architecture of the SCNN used with L1 norm as the distance function

3.5 Blocking

The blocking approach, implemented specifically for person names, divides the dataset into partitions and facilitates selective comparison between the blocks rather than the whole dataset. Blocking aims to reduce computational complexity so it has only been used when the soft TF-IDF algorithm has been applied in an unsupervised way on unlabelled 'Enron' data.
In Figure 3.2, column 'Data' represents the list of names present in a dataset. Each name is assigned a key, computed by taking together the first letters of the tokens of the name. The names with the same keys are put together in the same blocks. Now, all the possible combinations are generated as a subset of the letters present in each original block key. During comparison, a block will be compared to only those blocks who have keys which are present in that subset combination

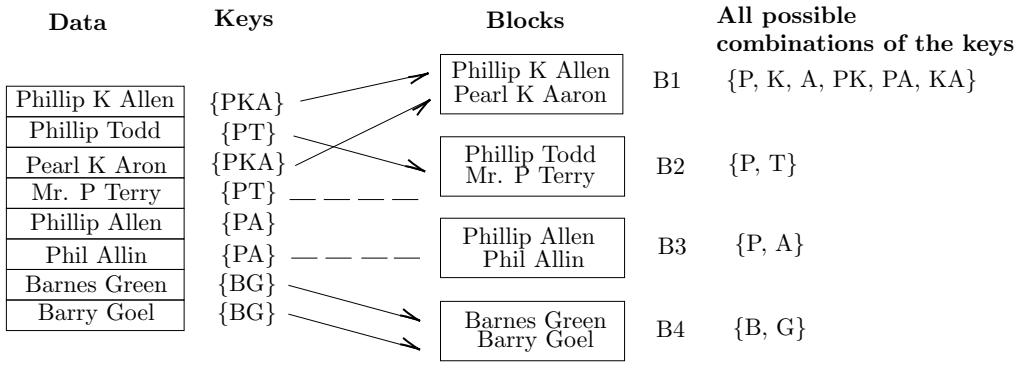


Figure 3.2: Partitioning of person names entities into distinct blocks based on the hashing keys generated from the first letters of the tokens of each name in the list

generated. The main idea is that all the similar names will be taken into account if the first letters of the whole name or combinations of first letters are considered. E.g. four blocks B1, B2, B3 and B4 have been created based on the keys generated from the names. For the name '**P**hillip **K** **A**llen', '**PKA**' is the key. The combinations subset for 'PKA' consists of six sub-keys P, K, A, PA, KA, PK. So, to find a match for this name, it will be compared with all the other names in it's own block i.e. B1 and with those in block B3, which has the key 'PA'. No other block has a key which is also present in the subset generated from B1 key. This saves the name from being compared with every other names in the dataset, thus reducing the performance overhead.

Chapter 4

Experimental Results

4.1 Dataset

Since, the primary objective of the project is to normalize personal names and company names, dataset gathering was focused on finding labelled personal names and organization names, both of which are mostly impossible to find because of privacy issue. Some work has been done on normalization of names but the source or the dataset has not been disclosed to the public. Table 4.1 gives a detailed overview of all the 'labelled' datasets used in our normalization experiments. The 'Dataset' column gives the domain of the data, the 'Details' column describes the purpose of the dataset and the two columns 'String1' and 'String2' show two examples of string pairs present per dataset along with their labels [match → 1, mis-match → 0] in the adjacent column 'Label'.

4.1.1 Construction of New Dataset

Initially, experimentation was started with the unlabelled 'Enron' dataset, but due to it's non-representative nature, a person names dataset has been constructed for both training and evaluation. Besides, for deep learning, a lot of labelled training data is required. For that purpose, SPARQL queries are run on Dbpedia to extract any entity of type 'person' with variations in sub-type, like actor, singer and so on. Their known names versus birth names are collected as potential matches. While in most of the cases, the matches are correct, a larger section represents mis-matches. E.g. 'William Feller' and 'Vilibald Sreako Feller' represent the same person but not identifiable even with human intellect without appropriate context. These examples are labelled as mis-matches. At the end of the process, 10000+ rows of positives are obtained with 300+ rows of negatives. The dataset is then augmented to make the matching pairs more representative of real world noise and human mistakes, add more mis-matching pairs and make it suitable as a training dataset.

Data Augmentation

The data here has two classes, positive and negative. The positive pairs are injected with noise by the following ways:

1. Insertion, deletion and replacement of random characters
2. Transposition of two characters
3. Deletion of space between tokens in names
4. Deletion of tokens in the strings
5. Swapping of tokens' order in names

Dataset	Details	Examples		
		String1	String2	Label
Constructed Personal Names	Training set, Size: 32000 pairs	Terence Scully	Terence Scully	1
	Testing set, Size: 1650 pairs	T. Scully	Terry Scully	1
	Class: Balanced (45% matches) Class: Imbalanced (4% matches)			
Zagat's Restaurant	Testing set, Size: 188 pairs	Arnie Morton's of Chicago 435 S. La Cienega Blvd. Los Angeles 310-246-1501 Steakhouses	Arnie Morton's of Chicago 435 S. La Cienega Blvd. Los Angeles 310/246-1501 American	1
	Class: Balanced (55% matches)	Art's Deli 12224 Ventura Blvd. Studio City 818-762-1221 Delis	Art's Delicatessen 12224 Ventura Blvd. Studio City 818/762-1221 American	1
Universities	Testing set, Size: 72 pairs	Virginia University, Charlottesville	University of Virginia, Charlottesville, Virginia	1
	Class: Balanced (55% matches)	Analytical Mechanics Associates	Analytical Mechanics Associates, Inc., Hampton	1
Enron Company Names	Testing set, Size: 162 pairs	New York Times	New York Stock exchange	0
	Class: Balanced (60% matches)	aep energy services ltd.	aep energy services gas holding company	1

Table 4.1: An overview of the different labelled datasets used for training and evaluation, with their respective sizes, class balance and two examples of string pairs present per dataset with their labels

The mismatches are created by the following ways:

1. From matches, random row numbers are generated from the pairs in loops

2. The pairs are recorded as non-matches under manual supervision

Finally, a training dataset of 30000+ rows of data are generated with 14000+ rows of positives and 16000+ rows of negatives. The test set consists of 700+ matching data points and 800+ non-matching data points.

Class Balance Selection

The person names test dataset, constructed above, has a good balance of positives and negatives. However, this equal balance is atypical of real world application of text normalization. So, an unbalanced dataset is also created for testing purpose. Five names are selected and different variations of those names are used as matches. The whole dataset consists of roughly 5% of matches and rest mis-matches. Evaluating the models against this imbalanced dataset inspects their precision, which now is highly sensitive because of the huge class imbalance in the data. The first row in Table 4.1 gives the details of the constructed person names dataset.

4.1.2 Available Labelled Datasets

Table 4.1 shows the details of the datasets available in different repositories in the last three rows. For company names, there is no convenient way to collect their variations in text or database. So a portion of 'Enron' company names data has been managed to label with the help of unsupervised string matching method (soft TF-IDF) and human effort and used as one of the evaluation sets. Two more datasets, Zagat's restaurant dataset and University names dataset have been collected from dataset collections of 'Cora'[1] repository. 'Zagat's dataset' is a collection of restaurant names with 120 matches, while the 'University' dataset consists of 40 matching pairs. Both of them are made class balanced by adding comparable number of mis-matching string pairs.

[1] http://www.cs.utexas.edu/users/ml/riddle/data.html

String1	String2	Predicted Label	Errors
Jason R.	Jackson R.	Match	Typical error
Carl	Cary	Match	Single token error
Tyson	Dyson	Match	
Barbara	Barbara A. Lloyd	Mis-match	Error due to difference in string length
Jose Luis Diaz Mendoza	J. L. Mendoza	Mis-match	Error due to initials
Richard Brown	Bob Brown	Mis-match	Error due to nick-names

Table 4.2: Observed errors in person names in soft TF-IDF results

String1	String2	Before weights	After weights	Status
duke energy trading and marketing	oneok energy marketing and trading	Match	Mismatch	Error solved by putting weights
pge energy trading canada corporation	pge	Mismatch	Match	
new york times	new york stock exchange	Mismatch	Match	Induced error
new york times	new york city pension funds group	Mismatch	Match	
dynergy inc	dyngy power marketing	Mismatch	Match	Unsolvable

Table 4.3: Observations for company names in soft TF-IDF results

4.2 Results and Observation

4.2.1 Soft TF-IDF

Soft TF-IDF works fairly well in not too big datasets and better for the balanced one. It is first applied in an unsupervised way to unlabelled 'Enron' personal names and company names datasets. The cut-off score ranges between 0.55 to 0.65 and is selected based on the number of false-positives and false-negatives detected. The overall precision calculated varies from 0.80 to 0.90 based on dataset type, distribution of names and variations in names.

Since, it is hard to calculate performance metrics accurately for unlabelled data, it has been further applied on the constructed labelled personal names datasets. For the class-imbalanced dataset, the average precision is 0.78 and for the class-balanced one, it is 0.91. On further analysis of the errors, it could be seen that there are certain cases which this method is not capable of handling, as illustrated below.

Table 4.2 presents the observations made on the person names dataset. The first row represents a very typical error by the models which rely heavily on edit distance based string matching mechanisms. The single token errors are unavoidable as well, due to which, in the later stages, these are excluded from evaluation under the intuition that if a single token name is present in the dataset, the corresponding full name or partial name is bound to be present and will be detected eventually. The errors due to string length difference and presence of acronyms and nick-names are caused by scores, which fall way below the selected cut-off.

Table 4.3 presents the observations made on the company names dataset. The performance of the algorithm on company names is highly sensitive to the presence of extra words in the strings. E.g. in the first row, even though the company names do not match, the surrounding tokens make the score reach cut-off. In the second row, the totally different surrounding tokens suppress the score below the cut-off. But in both the cases, the actual company name is the first token. So, the experiment is repeated by putting more weight on the first token of the company names and scores are recalculated. This hugely improvises the overall performance with a few induced errors, as shown in the 'induced error' column. Some unsolvable errors are also noted, which occur due to spelling mistakes.

Dataset	Features	Precision		Recall		F1 Score		Average Precision	
		SVM	XgBoost	SVM	XgBoost	SVM	XgBoost	SVM	XgBoost
Balanced	Distance metrics array	0.88	0.95	0.94	0.82	0.90	0.88	0.85	0.86
	Features as Boolean values	0.97	0.97	0.96	0.93	0.96	0.95	0.95	0.94
	Features as percentages	**0.96**	**0.96**	**0.98**	**0.98**	**0.97**	**0.97**	**0.95**	**0.95**

Table 4.4: Comparison of the models' performance for three different features on a balanced person names dataset

Dataset	Features	Precision		Recall		F1 Score		Average Precision	
		SVM	XgBoost	SVM	XgBoost	SVM	XgBoost	SVM	XgBoost
Imbalanced	Distance metrics array	0.97	0.80	0.53	0.7	0.68	0.74	0.54	0.57
	Features as Boolean values	1.0	1.0	0.87	0.65	0.93	0.78	0.88	0.67
	Features as percentages	**1.0**	**1.0**	**0.93**	**0.93**	**0.95**	**0.96**	**0.94**	**0.94**

Table 4.5: Comparison of the models' performance for three different features on an imbalanced person names dataset

4.2.2 Results with SVM and Xgboost

Both of the models are trained with the person names training set because no other training data is available or could be conveniently constructed. They are then evaluated, compared and improvised individually by adjusting user controlled parameters like similarity metric scores used in the feature set, and specific handling of errors. The following sections show the results of the models on balanced and imbalanced datasets.

Balanced Personal Names Dataset

Table 4.4 shows the results of the experiments. It provides a comparative view of the performances of both the SVM and XgBoost on the balanced dataset, trained individually on all three kinds of features, as described in the previous section.
For the first set of features, the performance is the poorest, which can be justified by the fact that the feature set works on overall similarity scores rather than handling the variations specifically; namely nick-names in person names. Yet, due to the balanced nature of the dataset, the scores are not that poor. Also, the performance is highly sensitive to the cut-offs selected for the various similarity metrics used.
For the second set of features, the performance increases for both the models, with SVM better than XgBoost. The recall score by XgBoost is not good because of too many false negatives in the result. The last row with the third set of features has highlighted scores because they are the best obtained in all possible parameters. Initially, two distance scores. Levenshtein and cosine, are included in the third feature set. The recall rises appreciably and the F1 score improvises with average precision remaining the same. As an experiment, the distance scores are removed from the feature set and the performance remains the same as ever, implying the power of the features used. Most of the errors observed here either need context to be considered a match or are very rare in real world situation.

Imbalanced Personal Names Dataset

Table 4.5 shows the results for an imbalanced dataset. For five selected name, there are 66 total number of matches in the dataset of 1649 data points. The highlighted values in the third row gives the best parameters scores for both the models. Even though, the precision does not improve considerably from training with first to third features set, the recall increases sharply with number of false negatives dropping from 20 to 4, thus giving a better F1 score.

Label	Variations	Prediction	Observations
Phillip Allen	Phillip K. Allen, Allen Keith Phillip, Phillip Keith Decosta Allen	TP	Middle names handled
	Phil Allen, Philly Allen, Philly Keith Allen	TP	Nick names handled
	P. K. Allen, P. Allen, P. K. D. Allen	TP	Acronyms handled
	PhillipAllen	TP	Injected noise handled
	Pgillip Allen, Phillip llen, Philippe Allen, Phllip Alen	TP	
	Phillip Decosta	FN	Failure to handle because of lack of context
	Keith Allen	FN	

Table 4.6: Observations on person names for the best performing models where TP=True Positives and FN=False Negatives detected

Dataset	Best Performing Feature set	Precision		Recall		F1 score	
		SVM	XgBoost	SVM	XgBoost	SVM	XgBoost
Enron company names	Features as percentages	0.91	0.92	0.71	0.69	0.80	0.79
University data	Distance metrics array	1.0	0.78	0.92	0.97	0.96	0.86
Zagat	Distance metrics array	1.0	0.89	0.98	0.96	0.99	0.92

Table 4.7: Performance of SVM and XgBoost on three new test datasets. The models are trained on personal names and tested for all the feature sets. The best performing feature set for both the models with the respective scores are given.

Observations on Personal Names

Table 4.6 provides an overview of how the models are capable of handling most of the variations for personal names. The examples are taken from the results of the imbalanced dataset but they are representative of the nature of real world variations in personal names and the models' performance on them. The results are derived after training with the third feature set. The models can be seen to be capable of handling token order, middle names, initials and noise. But, as expected, the highlighted rows show how the models fail to capture the context in similarity measurement.

Testing on new datasets

Table 4.7 shows the performance of the models on three new testing sets - 'Enron' company names, University data and Zagat's restaurant names. The models have been trained only on personal names dataset due to unavailability of other datasets for training. So, every other testing dataset of different domain are new to the models. As shown, for each dataset, both the models perform the best for a common feature set, e.g. both SVM and XgBoost give the best performance for company names with *features-as-percentages* as the feature set. The SVM gives better scores than XgBoost for all the datasets consistently with higher recall and precision. The feature set *features-as-Boolean-values* performs the worst for all the datasets, and thus not presented in the Table 4.7.
On analysis of the types of errors, specially in company names, the false-positives and false-negatives detected are similar to those reported in Table 4.3. For a pair of company names, 'Reliance Oil & Gas Corporation' and 'Reliance Power Ltd.', the semantic relation between the words ('oil', 'gas', 'power') and ('corporation', 'company') need to be considered while evaluating their similarities. Selective weighing of the first tokens in the strings is a possible solution as well but too domain-specific to be considered for other entities.

Embeddings	Accuracy	Precision	Recall	F1 score	Average Precision
One-hot	**0.88**	**0.92**	**0.81**	**0.86**	**0.94**
Flair	0.87	0.87	0.83	0.85	0.89

Table 4.8: Comparison of the Siamese CNN model performance for two different distance functions used on the balanced person names dataset

4.2.3 Results with Deep Learning

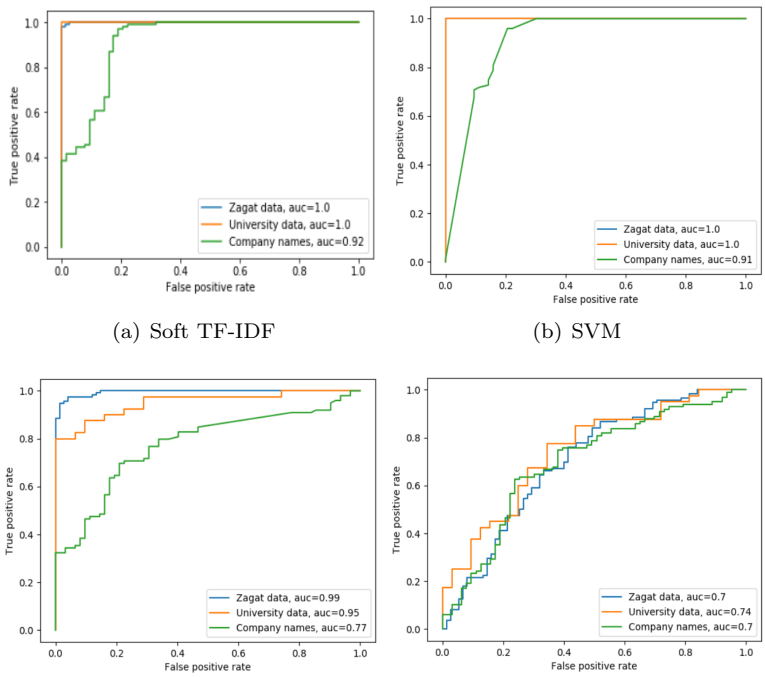
Table 4.8 shows the best results obtained from the experiments on balanced person names test dataset with different configurations. A batch size of 64 and 50 epochs are used for training. The best performing model, as shown by the highlighted values in the first row, is achieved with one convolution layer, L1 distance and one-hot encoding. The precision observed is as high as 0.94, which is quite good for a model trained with no explicit features. Interestingly enough, for L1 distance, increasing the number of convolution layers degrades the performance. But for cosine, model performance improves with more layers. Between one-hot encoding and contextual character-based 'Flair' embedding, one-hot performs more consistently, over multiple runs, with a precision of 0.86 and accuracy never going below 0.85. On studying the false positives and false negatives, no specific error patterns could be observed. However, the pairs with big differences as in presence of middle names or initials are mostly detected as false negatives.
For imbalanced person names dataset, the model performs very poorly. For both the distance functions used and the same architecture as above, the model behaves worse than a random classifier. Since majority of the string pairs are mis-matches, accuracy is good but precision drops below 0.4.

4.2.4 A Comparative Study of the Models on New Datasets

The models, trained on personal names dataset (except Soft TF-IDF), are finally evaluated on the testing datasets of 'Zagat's Restaurant' data, University data and 'Enron' company names data, individually, with the Receiver Operating Characteristics (ROC) curves and their respective Area-Under-Curve (AUC) scores. For classification problems on balanced data, AUC-ROC curve is a performance measurement at various thresholds settings. It is a probability curve where the AUC score represents the degree or measure of separability. So, higher the AUC score, better the model is at predicting matches and mis-matches.
The three testing datasets are balanced and labelled. With respect to the size and class-balance, they do not, quite, represent the real-world application of text normalization scenario. With increasing size and variations, the AUC scores will presumably drop, but in absence of large, available datasets, this study will give us an approximate idea of the generalizing capabilities of the models. It also gives us a direction towards selecting the most appropriate model for further studies. Figure 4.1 presents four plots for the four methods. Each plot consists of three ROC curves, corresponding to three datasets, plotted with False-Positive-Rate (x-axis) against the True-Positive-Rate (y-axis).

- *Soft TF-IDF:* In Figure (a), Zagat's data and the University data have a perfect AUC score of 1, which implies the model has the best possible sense of separability with no or very few false-positives and false-negatives. For company names, the score is acceptable at 0.92.

- *SVM and XgBoost:* Figures (b) and (c) show the curves by SVM and XgBoost respectively. Both the models perform comparably, with SVM having very good AUC score for company names, over XgBoost. This is corroborated by Table 4.7 where XgBoost is shown to have a very low recall score for company names, due to high false-negatives detected.



(a) Soft TF-IDF (b) SVM

(c) XgBoost (d) Siamese CNN

Figure 4.1: The ROC curves for all four methods show the false-positive rate versus true-positive rate on the predictions along with the Area-Under-Curve (AUC) scores

- *Siamese CNN:* Figure (d) shows the worst performance for all the three datasets. The AUC scores are a little higher than 0.5, demonstrating the model's inability to distinguish between positive and negative classes of data outside it's training domain.

To summarize, SVM proves to be the most consistent model which learns to generalize the best on different domains of data.

4.2.5 Results with Blocking

Implementation of blocking before comparison resulted in execution time as well as precision gain in the performance. Due to the avoidance of unwanted comparisons, huge improvement in execution time has been observed for person names. While some very typical errors disappeared, some errors were induced as well. In table 4.9, while the first two rows show how the method increased precision by reducing the scope of comparison, yet, the last one is an example of how the approach will not perform well in case the strings are noisy. Figure 4.2 shows the performance gain achieved with the use of blocking. Without blocking, the execution time increases exponentially with increasing data points. The plot represents testing done on 'Enron' dataset and will vary with more representative data points.

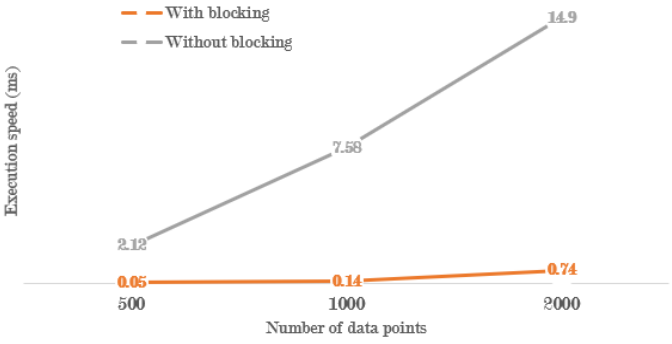


Figure 4.2: Execution time comparison for three different sizes of person names dataset with and without blocking

String1	**String2**	**Comments**
Jim Fallon	Tim Gallen	A match detected without blocking because of similarity in the names, but with blocking, never compared so error avoided
Barry O Neil	Garry O Neil	
Jack Darrell	Jack Farrell	A probable match but never compared due to mismatch in key.

Table 4.9: Observations for blocking on 'Enron' dataset

Chapter 5

Conclusion and Discussion

In this thesis, we present multiple models with different encoding, features and architecture and evaluated on four domains of data. Based on the comparison of their performance, we can conclude that features-based machine learning models perform better than the Siamese network architecture, used here. Among several other factors, an important one is the ease of training ML models. For a consistent performance, while, a mere 400 strong data points are enough to train the SVM and XgBoost, the CNN has to be trained with a 32000 data points training dataset. Initially, when the training of the SCNN was done with 10000 pairs of strings, the training and validation accuracies never reached a stable score over multiple runs. However, due to constraints regarding dataset, speed and memory of the GPU server, the SCNN model could not be experimented with, in it's full potential. Some of the observations regarding performance, dataset and model learning are discussed here, with reference to the research questions and with possibilities to improvise further.

Is it possible to provide a solution to text normalization which will work independent of the characteristics of the data?

The problem of text normalization is addressed in this thesis in a non-contextual way. In general, for two strings to be considered similar, the fundamental properties or features shared by the strings are approximately equivalent. Features such as prefixes, acronyms, abbreviations for tokens are frequent in majority of entities like person names, company names, restaurant names, e.g. (Art's Deli, Art's Delicatessen) as restaurant names with prefix 'Deli', (Larsen & Toubro Infotech, L&T Infotech) as company names with acronyms 'L & T'. So, matching a pair of strings based on these properties but without considering the surrounding context, can be considered a problem, solvable ideally with one model and trained with a selected set of common features. The XgBoost and the SVM implemented in this thesis, perform well even for the unknown 'Zagat' and University dataset. The features in Zagat's dataset such as postal codes, phone numbers, suffixes are not handled specifically in the model training features. Yet, they manage to identify the matches and mismatches with 85% to 90% accuracy. Training them further with more general features can work successfully on different data, independent of the domain.

How do the model performances vary with characteristics, pre-processing and class ratio in the dataset?

As discussed above, the problem of text normalization without context can be solved by considering characteristics conspicuous in all kinds of entities. So, a model trained with features, chosen carefully as not to overfit it, is capable of remaining insensitive to data specific characteristics. This has also been partly illustrated by the satisfactory performance of the ML models on the unknown testing datasets.
Pre-processing is an important factor in deciding the models' performance. E.g. alphabet normalization in personal names improvised the precision greatly. For the restaurant names, this step is

not necessary but including it does not the influence the models' performance negatively either. Another prominent example is the presence of zip codes and phone numbers in the restaurant names pairs. The numbers play a very important part in detecting a match due to the fact that two nearly similar named restaurants can be situated on the same street with, only, different phone numbers or zip codes. In that case, getting rid of numbers in the data, as done for personal names, does not help. So, pre-processing should be done in mediation as not to affect either of the datasets.
The results section shows the varying performance of the model with altering class balance. For a balanced dataset with proportionate matches and mis-matches, it is easier for the model to reach a reasonable score because of the stability offered by the mismatches. That is not the case though, in real world application, where one chosen entity is used to find its other representations, constituting only 5-10% of the whole dataset. The classification task needs to be more precise with false positives as low as possible in order to have a good precision or F1 score. The ML models implemented, show quite good consistency with both balanced and imbalanced person names dataset, thus demonstrating their ability to deal with different representations of an entity efficiently.
To sum up, the performance of the ML models, on the unknown datasets shows that training with the first or third set of features, makes the models more robust or generalized. But training with the second set of features produces an overfitting model with high variance, which is only capable of classifying a super set of the training data. It fails to classify with precision, possibly due to the fact that denoting the presence of token equality or acronyms or prefixes in the long string pairs, with a Boolean 0 or 1 does not give the model an understanding of the common features in the strings completely. So the model is unable to come to a conclusion.

Is Deep Learning capable of further improving the performances by machine learning models?

The answer to this question cannot be hard concluded based on the limited experiments done here. Deep learning models are highly sensitive to dataset strength, string embeddings as well as the architectural aspects and tuning parameters. Here, we experimented with two kinds of character embedding, on-hot and pre-trained 'Flair' encoding and two types of distance functions, L1 and cosine. No explicit features were used for training. So, a conclusive decision needs extensive experimentation with all of the above factors. Interestingly though, even with a limited dataset and a very simple layer architecture, the SCNN manages to behave as a fair classifier for the balanced dataset, which shows the ability of the model to learn the representations inherently. However, it fails to learn invariance to the inputs with their different representations. Thus, for the imbalanced dataset, where the labelled name has to be matched with only a few, noisy and different representations, the model does not identify the positives.

How to collect or prepare labelled dataset?

For person names, a dataset has been efficiently built, augmented and labelled, as discussed previously. For company names, the task is a lot harder because of unavailability of public, labelled dataset, yet a small labelled dataset is used for evaluation.

How to optimize the performance with suitable blocking approaches?

A possible blocking approach for optimizing normalization of person names has been implemented and effectively shown to be improving the execution speed. For the task of matching one selected entity among the whole dataset, blocking approach is possibly not required because only N comparisons are needed, N being the size of the dataset. But for normalizing a whole dataset, where each data point is compared with every other for a possible match, blocking techniques are helpful. However, no one blocking approach can be said to be successful for all domains of data. E.g. company names mostly start with the specific company name and surrounded by general words like petroleum, corporation etc. In that case, the first token or n-gram characters from first 2-3 tokens can be used as the partitioning key.

Chapter 6

Future Work

The problem of text normalization offers countless prospects to work and improvise on. The experiments and their results discussed here, give some definite conclusion and substantial ambiguity to continue further exploration. The future work can be divided into two sections, non-contextual entity normalization and contextual entity normalization. For the non-contextual part, which is the focus of this thesis, the overall observation points out three main areas which call for investigation; they are *i)* dataset, *ii)* feature-extraction and *iii)* model. The contextual part can be researched with to assess if the prediction can benefit from it.

Context-less text normalization

Dataset: The goal of this thesis was to develop a model which would work independent of the characteristics of the data. As easy as it sounds to be, to develop a domain independent working model, there should be sufficient training data which needs to be a right combination of labelled data from all possible domains. The training data can be a mixture of person names, company names, addresses, restaurant names, depending on requirements. A model trained with personal names data cannot be expected to give good accuracy for other domains, because it behaves as a model biased towards training domain features.

Features-extraction: The training of the model depends the most on the features used. The more general it is, the better. For person names, the features used in the best performing model, are prefix, initials, equality, dictionary, string-subset and dropped tokens. The string subset proved to be very useful for personal names but might influence organisation names negatively. Some other features, which could possibly strengthen the model, by keeping the model-learning unaffected for person names but helping in learning the variations in company names are:

- *Abbreviations:* If one token is equal to the subset of the other (*boulevard* $\rightarrow$ *blvd*)
- *Acronym:* If all characters of one token in first string are initial letters of all tokens in the second string (*Lone Star Pipeline Systems* $\rightarrow$ *LSP Systems*)
- *Removal of stop words:* For the two strings 'University of Virginia' and 'Virginia University', removing stop words like 'of', in this case, eliminates unnecessary comparison and helps the model to be more precise
- *Semantic Relation:* This is based on the concept of word2vec embedding. Instead of encoding the words, the distributional similarity of the words in the vector space can be taken into account for a pair of tokens from two strings. It can be represented with a Boolean value or in proportions. Semantic representation will enable the model to distribute weights effectively. E.g. In the strings 'J.P. Morgan group of financial institution' and 'J.P. Morgan bank', the tokens (*financial institution* $\rightarrow$ *bank*) share semantic similarity, otherwise not identifiable by models.

Model Selection: Lastly, choosing the right model is equally important. While features-based models can be very accurate and precise, yet the features need to be hand-crafted and that is only possible with extensive analysis of the data. On the other hand, the performance of the SCNN with simple character encoding and one-to-two layers, calls for more experiments and investigation. Since features based models proved to be very efficient, one promising approach in deep learning could be using multiple channels[15] for each inputs and feeding in the features set and the encoded string along the two channels of an input. Another direction could be replacing CNNs with LSTMs or bi-directional LSTMs since they are known to handle sequential problems the best. Training own data specific embeddings and using them to encode the strings can also be tried and tested with. There are multitudes of areas of improvement and experimentation in deep learning for text normalization.

Context-based text normalization

For contextual text normalization, the context of the entity before and after are considered before linking it to other entities. The advantage of adding context is that two different instances of the same entity, with same names, will not be linked together as a match, even if the calculated similarity score is high. With context, the model learns to understand the surrounding conditions in which a particular instance can or cannot occur. So, it can be expected to give more accurate results. E.g. Currently, none of the models implemented are capable of matching these two instances 'Renee Marie Parilak' and 'Renee Teate Parilak'. Considering the context around might solve the issue.

Bibliography

[1] Cohen, William W., Pradeep Ravikumar, and Stephen E. Fienberg. 2003. A comparison of string distance metrics for name-matching tasks. In Kambhampati, Subbarao and Craig A. Knoblock, editors, Proceedings of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03), August 9-10, 2003, Acapulco, Mexico, pages 73-78.

[2] Erwan Moreau, Franois Yvon, and Olivier Capp. 2008. Robust Similarity Measures for Named Entities Matching. In COLING. 593-600.

[3] Christen, Peter. 2006. A comparison of personal name matching: Techniques and practical issues. Technical Report TR-CS-06-02, Department of Computer Science, The Australian National University, Canberra 0200 ACT, Australia, September.

[4] Bilenko, M., Mooney, R., Cohen, W., Ravikumar, P., and Fienberg, S. (2003). Adaptive name matching in information integration. IEEE Intelligent Systems, 18(5), 16-23. doi: 10.1109 mis.2003/1234765. 6, 11

[5] Emiel Caron and Hennie Daniels, 2016. Identification of Organization Name Variants in Large Databases using Rule-based Scoring and Clustering - With a case study on the Web of Science Database, Conference: 18th International Conference on Enterprise Information Systems, DOI: 10.5220/0005836701820187 6, 11

[6] Sheila Tejada, Craig A. Knoblock and Steven Minton. Learning domain-independent string transformation weights for high accuracy object identification. In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD 2002, page 350, Edmonton, Alberta, Canada, 2002, ACM Press.

[7] Gottapu, R., Dagli, C., and Ali, B. (2016). Entity Resolution Using Convolutional Neural Network. Procedia Computer Science, 95, 153-158. doi: 10.1016/j.procs.2016.09.306 12

[8] Gan, Z., Singh, P. D., Joshi, A., He, X., Chen, J., Gao, J., and Deng, L. (2017). Character-level Deep Conflation for Business Data Analytics. 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). doi:10.1109/icassp.2017.7952551. 12

[9] Nihel Kooli, Robin Allesiardo, and Erwan Pigneul (2018). Deep Learning Based Approach for Entity Resolution in Databases. PagesJaunes - Solocal Group Rennes, France. 2, 12

[10] P. Neculoiu, M. Versteegh, and M. Rotaru, 2016. Learning Text Similarity with Siamese Recurrent Networks. In Workshop on Representation Learning for NLP. 12

[11] Jure Leskovec (Stanford Univ.), Anand Rajaraman (Milliway Labs) and Jeffrey D. Ullman (Stanford Univ.). Mining of Massive Datasets. 13

[12] R. C. Steorts, S. L. Ventura, M. Sadinle, and S. E. Fienberg. A comparison of blocking methods for record linkage. In Privacy in Statistical Databases, pages 253-268, 2014. 13

[13] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. (n.d.). Retrieved July 30, 2019, from https://arxiv.org/abs/1603.02754. 13

[14] Borggrewe, K., & Scholtes, J. (2016). Domain-Independent Method for Entity Resolution by determining Textual Similarities with a Support Vector Machine. Paper presented at The Benelux Conference on Artificial Intelligence, Amsterdam, Netherlands. 13

[15] Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). doi:10.3115/v1/d14-1181 14

[16] Vikramaditya Jakkula. Tutorial on Support Vector Machine (SVM). (n.d.). Retrieved August 1, 2019, from https://pdfs.semanticscholar.org/7cc8/3e98367721bfb908a8f703ef5379042c4bd9.pdf 14

[17] Signature Verification using a "Siamese" Time Delay .. (n.d.). Retrieved August 7, 2019, from https://papers.nips.cc/paper/769-signature-verification-using-a-siamese-time-delay-neural-network. 7

[18] Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual String Embeddings for Sequence Labeling. In: COLING 2018, 27th International Conference on Computational Linguistics (2018), pp. 1638-1649. 12