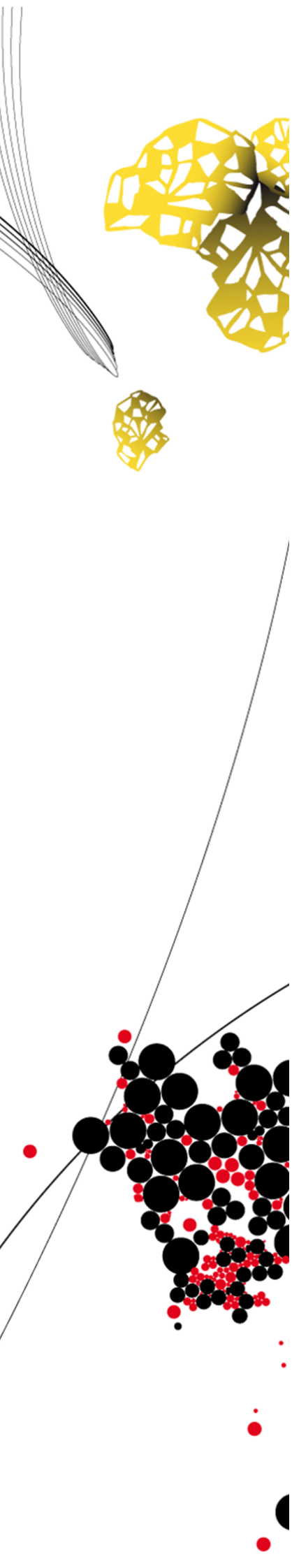


UNIVERSITY OF TWENTE.

**Faculty of Electrical Engineering,
Mathematics & Computer Science**

Automatic Product Name Recognition from Short Product Descriptions

Elnaz Pazhouhi

M.Sc. Thesis
March 2018

Supervisors:
Dr. Mariët Theune, HMI Group, University of Twente
Dr. ir. Dolf Trieschnigg, Mydatafactory
Dr. ir. Djoerd Hiemstra, Database Group, University of Twente

Human Media Interaction Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Acknowledgments

After passing all the ups and downs, now I am taking my last steps to finish this thesis. For me it was an exciting journey in the field of information extraction, full of new challenges and interesting problems. Now everything looks neat and clear but it was not like this at the beginning. It took sometime to define the problem and research questions clearly in a context that can be beneficial not only for academic purposes but also for practical industrial applications. Next I spent some more time to investigate different approaches and techniques, select a subset of the most effective ones and put them together and form a solution space. The implementation of the solutions was also an interesting part of the work where I developed a machine learning framework that helped me to automate the main steps of my investigations.

Mariët and Dolf, I am grateful to both of you for all your support and all your constructive feedback and comments throughout this work. You helped me to stay focused on the main research questions, also to define and present the concepts and results in a clear, understandable, and concise way. I would like also to thank Djoerd Hiemstra for his comments and his willingness to read and approve this thesis.

.

Abstract

This thesis studies the problem of product name recognition from short product descriptions. This is an important problem especially with the increasing use of ERP (Enterprise Resource Planning) software at the core of modern business management systems, where the information of business transactions is stored in unstructured data stores. A solution to the problem of product name recognition is especially useful for the intermediate businesses as they are interested in finding potential matches between the items in product catalogs (produced by manufacturers or another intermediate business) and items in the product requests (given by the end user or another intermediate business).

In this context the problem of product name recognition is specifically challenging because product descriptions are typically short, ungrammatical, incomplete, abbreviated and multilingual. In this thesis we investigate the application of supervised machine-learning techniques and gazetteer-based techniques to our problem. To approach the problem, we define it as a classification problem where the tokens of product descriptions are classified into I, O and B classes according to the standard IOB tagging scheme. Next we investigate and compare the performance of a set of hybrid solutions that combine machine learning and gazetteer-based approaches.

We study a solution space that uses four learning models: linear and non-linear SVC, Random Forest, and AdaBoost. For each solution, we use the same set of features. We divide the features into four categories: token-level features, document-level features, gazetteer-based features and frequency-based features. Moreover, we use automatic feature selection to reduce the dimensionality of data; that consequently improves the training efficiency and avoids over-fitting.

To be able to evaluate the solutions, we develop a machine learning framework that takes as its inputs a list of predefined solutions (i.e. our solution space) and a preprocessed labeled dataset (i.e. a feature vector X, and a corresponding class label vector Y). It automatically selects the optimal number of most relevant features, optimizes the hyper-parameters of the learning models, trains the learning models, and evaluates the solution set. We believe that our automated machine learning framework can effectively be used as an AutoML framework that automates most of the decisions that have to be made in the design process of a machine learning

solution for a particular domain (e.g. for product name recognition).

Moreover, we conduct a set of experiments and based on the results, we answer the research questions of this thesis. In particular, we determine (1) which learning models are more effective for our task, (2) which feature groups contain the most relevant features, (3) what is the contribution of different feature groups to the overall performance of the induced model, (4) how gazetteer-based features are incorporated into the machine learning solutions, (5) how effective gazetteer-based features are, (6) what the role of hyper-parameter optimization is and (7) which models are more sensitive to the hyper-parameter optimization.

According to our results, the solutions with maximum and minimum performance are non-linear SVC with an F_1 measure of 65% and AdaBoost with an F_1 measure of 59% respectively. This reveals that the choice of the learning algorithm does not have a large impact on the final performance of the induced model, at least according to the studied dataset. Additionally, our results show that the most effective feature group is the document-level features with 14.8% contribution to the overall performance (i.e. F_1 measure). In the second position, there is the group of token-level features, with 6.8% contribution. The other two groups, the gazetteer-based features and frequency-based features have small contributions of 1% and 0.5% respectively. However more investigations relate the poor performance of gazetteer-based features to the low coverage of the used gazetteer (i.e. ETIM).

Our experiments also show that all learning models over-fit the training data when a large number of features is used; thus the use of feature selection techniques is essential to the robustness of the proposed solutions. Among the studied learning models, the performance of non-linear SVC and AdaBoost models strongly depends on the used hyper-parameters. Therefore for those models the computational cost of the hyper-parameter tuning is justifiable.

Contents

Chapter 1

Introduction

Named Entity Recognition (NER) is a relatively new domain in the field of *information extraction*. Named entity recognition has been developed as one of the sub-tasks of information extraction where the named entities in the text are classified in pre-defined categories. Person, location, organization and time are some examples of general named entities, while music, game, and book are some examples of domain-specific named entities [1]. NER also is used for the recognition of product named entities (e.g. product name, brand, size). This is so-called *Product Named Entity Recognition* (PNER) [2, 3] that is one of the domain-specific subcategories of NER.

This thesis investigates different approaches in product named entity recognition. Our work is especially motivated by the company Mydatafactory [4]. The company is interested in tagging product names in the short product descriptions collected from ERIKS [5], a Dutch company that is active as a technical wholesaler and manufacturer. It is the supplier of large companies such as Shell. Their dataset is multilingual and they are interested in techniques that are able to automatically recognize product names in the product descriptions.

1.1 Motivation

We are living in an information age, the era that information technology influences almost all aspects of our life. The organization of modern business activities is one of those aspects. Nowadays Enterprise Resource Planning (ERP) systems are an inseparable part of modern business management systems. They are used to collect, store, manage and interpret data from many different business activities running in an organization. ERP systems track business resources, raw materials, production processes, orders and purchases. This is where the data from different departments (e.g. manufacturing, purchasing, sales, accounting, etc.) are collected in a centralized manner to be able to monitor and track the core activities of businesses.

One part of the data stored in ERP systems, is business exchange transactions. These are the transactions between the main producer, intermediate businesses and end users. Products[1] are the subject of these transactions. They are provided by a supplier and are sold to a customer. These transactions can happen between a business and the end user of the product, known as B2C (Business to Customer) transactions, or between two businesses, known as B2B (Business to Business) transactions. In both cases one side of the transaction describes its needs in the form of a product description. This is typically a short description that specifies the important features of the requested product. The other side of the transaction also has a set of product descriptions stored in the form of product catalogs describing the products that the supplier sells. The product descriptions are written in natural language. A match between the customer and supplier product descriptions is a potential business transaction. As a result, tools that are able to dig into the data stored in ERP systems and relate product descriptions are useful for enterprises. This is specially very interesting for the wholesalers and intermediate businesses. The main goal in these businesses is to find the best matches between the customers' requests and the product catalogs that they receive from the customers and the suppliers respectively.

The problem of matching product descriptions is not trivial. There is no inter-business standard for representing product descriptions. They are sometimes outdated, incomplete, company-specific, and abbreviated because of technical constraints. As a result two seemingly different product descriptions which do not share any syntactic similarity may refer to the same product. So the relation between terms in two different product descriptions is in many cases a semantical relation. For example two terms in two different product descriptions may be synonyms that refer to the same actual product. However, one term is commonly used in one business domain while the other one is common in another business domain.

One approach to tackle this problem is to use the dataset of the previously matched product descriptions to extract the semantical relations between the named entities in them. The fact that two product descriptions are matched implies that there is a relation between their named entities. One of the most important named entities in the domain of product descriptions is the *product name*. This means that one critical step in the above-mentioned approach is to develop a technique to automatically recognize product names in the product descriptions. This leads us to the field of named entity recognition and its domain-specific subcategory, product named entity recognition.

[1] In the context of this thesis we use the term products to refer to both the physical products and also the services.

1.2 Problem Statement

This thesis addresses the problem of automatic recognition of product names from unstructured short product descriptions stored in ERP databases. We elaborate this problem using the following example. The example shows one of the product descriptions of our dataset.

"CARROSSERIERONDEL M6 X 30 DIN440R"

We are interested in recognizing the product names "CARROSSERIERONDEL" and "DIN440R" in the product description. This is especially a challenging task, because product descriptions are multilingual, short and ungrammatical (i.e. they do not follow standard grammatical rules or writing conventions such as capitalization) and they contain minimal linguistic context.

1.3 Research Objective

The aim of this thesis is to investigate the state-of-the-art NER techniques for language-independent product name recognition. Specifically we look into machine learning and gazetteer-based approaches. We design a solution space that contains machine-learning based NER solutions with different configurations. The solutions use ensemble learning models such as AdaBoost, Random Forest, linear and non-linear support vector classifier (SVC). Then we develop a machine learning framework that enables us to evaluate the proposed solutions and study different aspects of each solution. We are interested in the relative performance of different learning models, analyzing the usefulness of different feature groups used to train the predictive model and the role of hyper-parameter tuning. Moreover, we study how product name gazetteers can be integrated into learning models and how effective they are when they are used for product name recognition.

1.4 Research Questions

The main research question of this thesis is:

How can existing named entity recognition techniques be used for product name recognition?

To be able to answer this research question, we divide it into smaller sub-questions:

- RQ1: What are the main discriminating features representing a predictive model for product names in our dataset?

- RQ2: Among the learning models chosen for this study (i.e. linear SVC, non-linear SVC, RF and AdaBoost), which one induces a better predictive model?
- RQ3: How can gazetteers be incorporated into the predictive model? And to what extent can this improve the performance of product name recognition?
- RQ4: Tuning the hyper-parameters of the models is an optimization problem that may strongly affect the induced model but it imposes high computational cost. The question is whether in the context of our PNER problem, the computational cost imposed by the hyper-parameters tuning justifies the performance gain that can be obtained from it? Does the answer to this question depend on the used learning model?

1.5 Contributions

The contributions of this thesis include:

- Designing a set of hybrid NER solutions that combines gazetteer-based and machine learning based approaches in NER.
- Developing an automatic machine learning framework that enables us to automatically optimize the hyper-parameters of the solutions and then study various aspects of the designed solution space.
- Using the developed machine learning framework to answer the proposed research questions (see Section 1.4)

1.6 Outline

This thesis is organized in five chapters. After the introduction, Chapter 2 gives an overview of existing NER techniques and important machine learning concepts used in this research. In Chapter 3 we focus on our methodology, we present our feature set, solution space, machine learning framework, the pre- and postprocessing steps and our evaluation method. In Chapter 4, we present the results of our experiment and based on the result we answer the proposed research questions. Finally we conclude the thesis in Chapter 5 and discuss some promising future directions.

Chapter 2

Background

Named Entity Recognition (NER) is one of the sub-tasks of information extraction. The objective of NER is to annotate the phrases of a given text with some predefined categories [1]. The categories in NER are divided into *general* and *domain specific* categories. Person, organization, time, and location are examples of the famous general named entities [6, 7]. In addition to general named entities, each domain of expertise has its own domain specific categories such as genes, protein names, cell, RNA, and DNA in the domain of biology [8–10], singer name, band name, song name in the domain of musicology [11] and product name, brand name, and product type in e-commerce and business domain [2, 3, 12].

This thesis studies the topic of product named entity recognition. From typical product named entity categories (e.g. brand name, product name, product size and product series [13, 14]), we specifically focus on the recognition of product names in short product descriptions. To support the required background for the rest of this thesis, in this chapter we give an overview on the main approaches in Named Entity Recognition and we explain the important machine learning concepts that are used in this thesis.

2.1 Named Entity Recognition

Research on named entity recognition is still at its early stages. The main NER approaches are: (1) rule-based techniques, (2) machine learning techniques, and (3) gazetteer-based techniques. Machine learning techniques assume the existence of an annotated corpus and use machine learning algorithms to learn a predictive model to recognize and classify named entities. Rule-based techniques rely on handcrafted linguistic patterns and recognize named entities by pattern matching. Gazetteer-based techniques, also known as dictionary-based techniques, rely on the use of gazetteers (i.e. dictionaries) that contain a list of predefined named enti-

ties. These lists are used to recognize and classify named entities.

2.1.1 Rule-based Approach

Rule-based approaches use handcrafted linguistic patterns and recognize named entities by applying pattern-matching. The problem is that good rules need significant effort of domain experts, and are not easily adaptable to new domains. Bick et al. [15] used a constraint grammar-based parser to recognize named entities in Danish texts. Their technique is based on a set of predefined rules and is able to recognize different named entities that also includes product names. The approach highly depends on the performance of a Danish parser; thus it is not portable to other problems, specially to multilingual problems.

In [16] the authors use a set of parser-based rules to automatically generate an annotated corpus which later is used to train a Hidden Markov Model (HMM) named entity classifier. Some of their proposed rules are used for the recognition of product named entities. For example the following rule:

$$\text{Has\_AMod(handheld)} \Rightarrow \text{PRO}$$

is one instance of a Has_AMod(X) $\Rightarrow$ PRO rule family which states that the name which is modified by the ”handheld” is most likely to be a product named entity PRO. The following rules are other instances of this rule family:

$$\text{Has\_AMod(fuel-efficient)} \Rightarrow \text{PRO}$$
$$\text{Has\_AMod(well-sell)} \Rightarrow \text{PRO}$$
$$\text{Has\_AMod(valueadd)} \Rightarrow \text{PRO}$$

As another family of rules, they propose that the object of some verbs have higher chance to be a product named entity. From this rule family the following rules can be derived:

$$\text{Object\_Of(refuel)} \Rightarrow \text{PRO}$$
$$\text{Object\_Of(vend)} \Rightarrow \text{PRO}$$

A similar idea is used to create more rule families such as:

$$\text{Has\_Predicate(accelerate)} \Rightarrow \text{PRO}$$
$$\text{Has\_Predicate(collide)} \Rightarrow \text{PRO}$$
$$\text{Possess(patch)} \Rightarrow \text{PRO}$$
$$\text{Possess(rollout)} \Rightarrow \text{PRO}$$

In general rule-based approaches have two main drawbacks: (1) they need a list of hand-crafted linguistic rules and (2) they are language dependent. Therefore, they are not suitable for the multilingual named entity recognition (i.e. the problem of this thesis).

2.1.2 Machine Learning Approach

One approach to tackle the named entity recognition problem is to formulate it as a classification problem that can be effectively solved by applying a wide range of machine learning techniques. This section explains the main groups of machine learning techniques that have been used for Named Entity Recognition.

Supervised methods

Supervised machine learning methods are a class of algorithms that learn a predictive model by looking at an annotated training set. The main learning algorithms are: Decision Trees [17], Neural Networks [18], Ensemble learning methods such as Random Forest [19] and AdaBoost [20], Support Vector Classifiers (SVC) [21], Maximum Entropy Models (ME), Hidden Markov Models (HMM) [22] and Conditional Random Fields (CRF) [23]. The performance of these techniques is widely studied in NER for general named entity categories such as person, location, and so on [17–23]. Although they can reach near-human performance for general name entity recognition, their major drawback is that these techniques require a sufficiently large annotated dataset in order to induce an accurate predictive model. In the rest of this section, we discuss some of the researches that use supervised learning specifically for the task of PNER. For the applications of supervised learning techniques in the domain of NER, we refer to references [17, 18, 21–23].

Pierre [24] developed an English named entity recognition system and used it to recognize product named entities in a large collection of product reviews for audio equipments (e.g. Speakers). They specifically used Naive Bayes and Boolean classifiers for knowledge discovery on automatically generated metadata. They defined four metadata facets: category (including 11 product categories), subcategory (including 49 product subcategories), products, rating (including “Good” and ”Bad” ratings). They trained a Naive Bayes classifier for each facet. Then they use these classifiers to automatically generate metadata for product reviews. Their corpus contained 47923 individual product reviews. They used half of the corpus as their training set and the other half as the testing set.

Luo et al. [25] develop a PNER technique based on introducing domain ontology features to the CRF models. As an example they consider Notebook products. First they construct a domain ontology for these products. Then to construct features of the CRF model, they define three feature groups: word context features, part of speech features, and ontology features. According to their evaluations, the latter outperforms the other feature groups (specifically much better results in terms of recall measure).

Semi-supervised machine learning

To overcome the cost of providing a large annotated training set, semi-supervised or weakly supervised learning approaches have been developed. These techniques are focused on the automatic construction of annotated corpus. They begin with a small annotated corpus and extend it using the co-training [26–29] and bootstrapping [30] techniques.

The central idea of co-training is to separate features into multiple orthogonal views. For example in the task of NER, one view utilizes the context evidence and the other view relies on the dictionary evidence. The classifiers corresponding to different views learn from each other iteratively. Blum et al. [26] shows that co-training can be very efficient such that in the extreme case only one labeled data sample is needed to learn the classifier. Compared to bootstrapping techniques, co-training suffers from error propagation which is the result of iterative learning used in this technique [31].

Niu et al. [16] used a bootstrapping approach in named entity classification. They first learn some parsing-based named entity rules from a small annotated corpus, then these rules are applied on a large unannotated corpus to automatically generate a large annotated corpus which later is fed into a Hidden Markov Model named entity learner. In this sense their approach is the combination of machine learning and rule-based approaches. They also apply their named entity classifier to a dataset with 2000 product named entities. Their classifier is able to reach 63.7% precision and 72.5% recall with F_1 score of 69.8%. However, their technique has two main drawbacks: (1) it highly depends on the performance of English grammar parser and (2) it is difficult to extract parser-based name entity rules for the coverage of different product named entities.

Gazetteer-based Approach

To the best of our knowledge the use and the effectiveness of gazetteers for the task of PNER has not been studied yet. However, there are some works that use gazetteers to improve the performance of NER [32, 33]. Generally, gazetteer-based techniques assume the existence of a domain specific dictionary which can be used to identify specific types named entities. Therefore, the main challenge lays in the construction of a comprehensive dictionary for a particular domain. In this direction, in [9] the authors propose a learning approach with minimal supervision to construct dictionaries for different named entity types (in particular for biomedical named entity types such as viruses and diseases).

2.2 Concepts of Machine Learning

This section gives a short introduction to the main machine learning concepts that are used in this thesis. This covers feature engineering including feature construction and different techniques for feature selection, learning models that are used in this thesis (i.e. linear and non-linear SVC, Random Forest, and Adaboost), cross-validation and the over-fitting problem, and automated machine learning frameworks.

2.2.1 Feature Engineering

Feature engineering is a set of techniques that includes the process of constructing the set of candidate predictive variables for the model (i.e. *feature construction*) and reducing the constructed candidate variables to a subset of most relevant variables (i.e. *feature selection*) [34,35]. We dedicate the rest of this section to briefly discuss these two steps.

Feature Construction

The goal of feature construction is to create a strong set of predictive variables, so-called *features*. This is a vital step in the machine-learning-based solutions, no matter which learning algorithm they use. In fact, much of the success of machine learning algorithms depends on the quality of the constructed features. Features are sometimes obvious and sometimes they are not so trivial. In general feature construction is difficult and creative process in which under-specified, ill-formed raw data should be shaped into a set of predictive variables.

Feature Selection

In some applications the size of feature set may grow dramatically. This may result in a number of problems such as over-fitting, dramatical increase in the computational overhead, and performance loss. To tackle these problems several feature selection methods have been proposed. This section first explains why feature selection methods are needed specially in this thesis and then discusses the main feature selection techniques.

The Need for Feature Selection

In classification problems that deal with text data (e.g. the problem of this thesis) the number of features tends to increase dramatically. This is because of the existence

of features with type string that are typical in this category of problems.

The string features can be divided into two main categories: *categorical features* (also called *nominal features*) and *ordinal features*. The value of a categorical feature belongs to a finite set of predefined categories. For example the part-of-speech of the current token is a categorical feature. Similarly an ordinal feature may take a value from a predefined set of categories; however, this time there is an intrinsic ordering among the predefined categories. For example assume the token length as a feature where instead of an integer we only care if a token is long, medium or short. This defines an ordinal feature, because there is an ordering between the three categories: short is smaller than medium and medium is smaller than large.

When we work with text data, many features that are constructed have the type of string. The problem is that the learning algorithms only accept binary features (i.e. features with only True or False value) or numerical features (either integer or floating point features). Thus the string features should be encoded into binary or numerical features. In case of ordinal features, due to their intrinsic ordering, it is possible to encode them into numerical values. However, for categorical features this encoding does not work. Because they have no natural ordering, numerical encoding would mislead the learning algorithm. For example in our part-of-speech example, we cannot encode pronouns into 1, nouns into 2 and verbs into 3, because in that encoding the average of a verb and a pronoun is noun which does not make any sense. Therefore the categorical features should be encoded into binary features. This is technically called *binarization*. Binarization suddenly increases the size of the feature set as one single string feature is encoded into hundreds or thousands of binary features.

Working with large feature sets results in two important drawbacks: (1) it dramatically increases the training time, and (2) it degrades the robustness and accuracy of the predictive model on the unseen data due to over-fitting. In practice only a subset of features significantly contributes to the performance of the predictive model [36].

Main Feature Selection Techniques

Feature selection is the process to automatically select a subset of features that correlates the best with the data [37]. Feature selection methods can be used as a filter that removes *irrelevant* and *redundant features* from the feature set. The irrelevant features are the ones that have no or very small contribution to the prediction accuracy. Redundant features also known as *dependent features* are features that have the same influence on the prediction accuracy. Irrelevant and redundant features make the model more complex, impose unnecessary computations and consequently increase the training time, and reduce the interpretability of the feature

set. They also increase the chance of over-fitting (see Section 2.2.3).

There are three groups of feature selection methods: *filters*, *wrappers*, and *embedded methods*. All these methods automatically select the most relevant subset of features. Each method includes a group of techniques that choose the same strategy for feature selection [36, 38–40].

Filter Methods. In filter methods features are chosen based on the characteristics of data without using any classifier in the process of feature selection. Filter methods are composed of two steps: first the features are ranked according to certain criteria; then in the second step, the features with the highest rankings are selected to induce the predictive model. Fisher Score (or F-test) [41], ReliefF [42], and methods based on mutual information [43] are the most representative filter-based feature selection algorithms. Among them Fisher score is one of the most widely used criteria due to its good performance [44]. In this thesis we use Fisher score for feature selection.

Fisher score. Filter-based algorithms based on Fisher score are *univariate* evaluation features. This means that features are selected, ranked and evaluated independently. Thus this method neglects the usefulness of combinations of features (i.e. evaluating two or more than two features together) and therefore it cannot distinguish between redundant and non-redundant features. The central idea in this method is: features with high quality should assign similar values to instances in the same class and different values to instances from different classes. According to this, the score for the i^{th} feature S_i is calculated as:

$$S_i = \frac{\Sigma_{j=1}^{K} n_j (\mu_{ij} - \mu_i)^2}{\Sigma_{j=1}^{K} n_j \rho_{ij}^2} \tag{2.1}$$

where μ_{ij} and ρ_{ij} are the mean and the variance of the i^{th} feature in the j^{th} class respectively, n_j is the number of instances in j^{th} class, and μ_i is the mean of the i^{th} feature [36].

Wrapper Methods. In practice features are not independent of each other; the effect of one feature may differ when it is used in combination with other features: ”a variable useless by itself can be useful together with others” [38, 44]. This is the main motivation behind the wrapper methods. They focus on finding the best combination of features (i.e. *multivariate* evaluation). Wrappers utilize the learning algorithm of interest as a black box to score the subsets of the features based on their predictive power. This method comes in three important strategies: *backward feature elimination*, *forward feature selection*, and *recursive feature elimination* [38]. The drawback of these methods is that they are computationally expensive and it

is not clear if the gained predictive power justifies the imposed computational load for a certain application (for example product named entity recognition). Wrapper methods use a learning model, as a black box, to select the best combination of features. This is regardless of the chosen predictive model; therefore the training of the predictive model based on the selected features should be done after the feature selection step. This imposes even more computational load to the method. To address this issue embedded methods have been evolved to fill the gap between filter and wrapper methods.

Embedded Methods. Embedded methods embed feature selection with classifier construction. These methods bridge the gap between filter and wrapper methods. They first use statistical measures, similar to filter methods, to select subsets of candidate features with predefined cardinality. Second they use these candidate feature sets to induce learning models; the candidate feature set with the highest accuracy is chosen and there is no need to use it to train the predictive model as it is already done in the second step of the method. In this way, the embedded methods obtain results that are comparable with those of wrapper methods in terms of accuracy while they are more computationally efficient than the wrapper methods [45, 46].

2.2.2 Learning Models

This section briefly introduces the learning models that are used in this thesis. We give an intuitive explanation for each learning model. Our objective is to give an idea on how the learning algorithm works in general. We also give references to more detailed explanations of each algorithm for further reading.

Random Forest

Random forest [47] is one of the ensemble learning methods [48, 49]. This is a category of learning algorithms in which a set of weak learners are combined to construct a single powerful learner. Random forests aggregate the result of many decision trees where each tree is trained with a randomly chosen subset of features over a subspace of the training set. Random forest is a powerful and popular classifier that is successfully applied to different applications [50]. The main reason behind the success of this classification method is not clear from the mathematical point-of-view [51, 52]. However, Breiman [47] relates this success to the *out-of-bag* strategy: based on that strategy the samples that are not used for training the current tree, are used to estimate the prediction error and then to evaluate the feature

importance. The number of and the depth of trees are free variables that can be tuned as the hyper-parameter of the classifier. Recently random forests have been also used successfully as a feature selector [53].

Support Vector Classifier (SVC)

The SVC is a discriminative classifier that mathematically works based on finding optimal hyperplanes. The hyperplanes separate different classes of data. The method is originally designed for binary and linear classification. However it is shown that the linear core of the classifier can be extended for classifying non-linear problems by a technique that is called *kernel trick*. The kernel trick uses kernel functions to map the data to a new space on which the data is linearly separable. This additional dimension is calculated by a kernel function. The main kernel functions are: Radial Basis Function (also known as Gaussian), exponential, polynomial, hybrid and sigmoidal. To enable to use SVC for multi-class classification, a number of binary support vector classifiers are combined.

The linear variant of the method has different hyper-parameters. The parameter C (also known as *soft-margin*) is the most influential hyper-parameter of the model. The soft-margin parameter enables more flexibility in choosing the hyperplanes of the classifier. It is a generalization of *hard-margin* where the optimal hyperplane is the one that is exactly in the middle of *support vectors*. Support vectors are the vectors that determine the boundaries of the samples of each class. By setting soft-margin, the user of the model can determine where between the support vectors, the hyperplanes should be placed.

Adaptive Boost (AdaBoost)

AdaBoost, the abbreviated name of Adaptive Boost, is one of the earliest boosting algorithms that belongs to the category of ensemble learning models. In this technique an increasingly complex predictor is constructed by combining weak predictors. As a result boosting enables us to create a powerful predictive model out of many weak learners (e.g. decision trees).

The algorithm starts by assigning equal weights to each data point (i.e. each token in NER applications). Then it iterates for a certain number of times. In each iteration it trains a decision tree (i.e. the weak learner) on a specific number of features. After training the decision tree, an error is calculated for each data-point based on that the weight of the data points are updated such that the data points that are mis-classified get a higher weight; so they can hopefully have more chance to be trained in the next iteration. We also calculate a weight for our weak learner. This weight is calculated based on the classification error and indicates how much

we trust this weak learner. In the next iteration we choose another set of features (randomly) and train another weak learner over that. This time we use the weighted data points that are biased towards the data-points that were missed by the previous learner. The new weights for each data point and a new weight (coefficient) for the weak learner are calculated. This is repeated until some predefined number of iterations. The number of iterations is equal to the number of weak learners which is one of the hyper-parameters of the algorithm. At the end, we have trained n weaker learners and for each one we calculated a separate weight that indicates how much we are confident on the classification of that specific weak learner. Finally the complex predictor is constructed by linear combination of the weak learners and their weights as follows:

$$y = Sign(\Sigma_{i=1}^{N} w_i * f_i(x)) \tag{2.2}$$

where y is the classification of data point x, N is the number of weak learners, f_i is the i^{th} weak learner and w_i is its weight.

2.2.3 Cross-Validation

One of the main concerns in using machine learning techniques is whether the predictive model that has been trained over a limited dataset can work with almost the same performance on the future unseen data. To resolve this issue, in machine learning methodology, the dataset is divided into: train set, validation set and test set. These sets are typically chosen randomly from the dataset. The training set is used to train the predictive model using the learning algorithm with certain hyper-parameters, The validation set is used to fine-tune the hyper-parameters of the model, and the testing set models the future unseen data and is used to evaluate the actual performance of the predictive model. The performance of the model on the testing set is assumed to be an approximation of the performance of the model on the future unseen data.

The performance metrics measured in the above-mentioned methodology are not robust in practice as the performance of the predictive model depends on how data samples are divided into the testing, validation and training sets. This is especially a critical issue when the number of features grows compared to the size of the training set (which is a typically the case in text categorization problems). In these cases, the model fits too well to the training set, so-called over-fits the training data. In this situation, the random selection of the training set does not resolve the over-fitting problem, as the performance metrics may differ from one randomly chosen training set to another. Increasing the size of train set mitigates the problem but it is expensive and may not be feasible in many applications.

Cross-validation is a commonly used approach to check how well the model generalizes to new data. Moreover, cross-validation enables the use of the whole dataset for training and testing; this is in contrast to explicitly assigning one part of the data to training and the other parts to validation and testing. In cross-validation, the dataset is divided into a number of folds. That is the reason this method is also known as K-fold cross validation. In each iteration one fold is taken as the testing set and the others are taken as the training set. This is so called *leave-one-out cross-validation* (LOOC) [54]. The predictive model is trained and the performance metrics are computed. This process repeats for all folds. The final performance metric is the average of performance metrics over all iterations. The number of folds is a hyper-parameter of the technique that can be tuned based on the application.

2.2.4 AutoML

In general to design a machine learning solution, one has to solve several decision problems such as which learning model and which feature selection technique should be used? What is the optimal number of features? And what are the optimal hyper-parameters for the chosen learning model? Automatic Machine Learning (AutoML) Frameworks (e.g. AUTO-SKLEARN [55], HYPEROT-SKLEARN [56] and AUTO-WEKA [57]) are tools that are able to automatically solve the above-mentioned decision problems. AutoML problems are also defined in the context of the CASH (Combined Algorithm Selection and Hyper-parameter optimization) problem [57]. AutoML algorithms exploit different machine learning techniques to construct more automated, robust and efficient machine learning frameworks. Feurer et al. present a precise definition for the AutoML problem in [55].

2.3 Summary

This chapter presents the main approaches to NER: (1) machine learning techniques, (2) rule-based techniques and (3) gazetteer-based techniques. Because of the ungrammatical and multilingual nature of our dataset, in this work we focus on the machine learning and gazetteer-based techniques. More specifically we investigate a set of hybrid solutions that combine machine learning and gazetteer-based techniques. This chapter also gives an introduction to the main steps and concepts used in developing machine learning solutions. We start with the most influential step, the feature engineering step (including both feature construction and feature selection) and continue with discussing main feature selection techniques and learning models. We explain three main approaches in feature selection: (1) filter methods (2) wrapper methods (3) embedded methods. In this work we employ filter

methods and embedded methods in our proposed solutions. However, because of the high computational cost of the embedded methods, our experiments is only limited to the solutions that use filter methods for feature selection. This chapter also discusses a set of learning models: linear and non-linear SVC, random forest, and AdaBoost. Later in this thesis we use these models to induce required predictive models for the task of product name recognition.

Chapter 3

Methodology

This chapter discusses our methodology. Throughout this chapter we create different parts of a machine learning framework that enables us to investigate different aspects of machine learning-based solutions for the task of product name recognition. We begin with introducing our dataset and the format of product name annotations. We continue with data preparation steps and constructing a set of relevant features. Then we discuss how automatic feature selection methods are employed to reduce the dimensionality of data. We also present the configuration of the machine-learning solutions that are investigated in this work. Each solution is composed of a feature selection method, a learning algorithm and a set of hyper-parameters. Our framework automatically selects the most effective subset of features, and optimizes the hyper-parameters of the solution. Thus, in addition to its application for experimenting machine-learning techniques, our framework is sufficiently generic to be used as an automatic machine learning framework. The important advantage of automatic machine learning frameworks is their ability to simplify the process of designing machine-learning solutions by automating most of the required confrontational decisions (e.g. the choice of learning model, feature selection method, optimal number of features and model hyper-parameters)

3.1 Dataset

This section discusses our dataset and the format and structure of product name annotations. The dataset is a set of short product descriptions from ERIKS [5]. It contains 155427 product descriptions among which, for 2091 product descriptions the product names are manually annotated. The manual annotations are provided by the company Mydatafactory [4]. Each product description in the dataset may contain one or several product names and each product name may be composed of one or multiple adjacent terms.

Product Description	CILINDERSCHR. MET ZAAGGLEUF M5 X 20 DIN84
Product Name Offsets	['1:14', '1:28', '37:42']
Product Names	PN1= CILINDERSCHR. PN2= CILINDERSCHR. MET ZAAGGLEUF PN3= DIN84
Product Names after Tagging	[('CILINDERSCHR.', 'B'), ('MET', 'I'), ('ZAAGGLEUF', 'I'), ('M5', 'O'), ('X', 'O'), ('20', 'O'), ('DIN84', 'B')]"

Table 3.1: An example of tagging product name tokens when there are overlapping product names in the product description

Annotation Format

Given that a product description is a list of terms, the objective of annotation is to determine which sequence of adjacent terms in the product description is a product name. The annotation of product names has been done by adding a list of offset pairs. Each offset pair marks one product name in the product description. The pair contains two indices indicating the index of the starting character and the index of the ending character of the annotated product name. We assume that a product description is the string S of characters (including white spaces) such that the first character has index 1 and the last character has the index $\text{len}(S)$. The following example shows how the product names are annotated in the given product description:

CILINDERSCHR. MET ZAAGGLEUF M5 X 20 DIN84,
T1 Productname 1 14 CILINDERSCHR.,
T2 Productname 1 28 CILINDERSCHR. MET ZAAGGLEUF,
T3 Productname 37 42 DIN84

where the first line is the given product description and the rest of the lines represent the annotated product names. For example the second line indicates that the first product name starts from the index 1 and ends at the index 14 in the product description string.

Overlapping Product Names

Sometimes when there are multiple product names in a product description, the offset range of one product name may completely cover the offset range of another one. In this case, we take the larger product name and drop the smaller one. The reason for this decision is that the larger product name is assumed to be more informative than the smaller one. Table 3.1 gives an example of this case, where the product name PN1 is fully covered by the product name PN2.

3.2 Data Analysis

The main objective of data analysis is to collect statistical information about the product names in our dataset. According to the analysis, 70.97% of the product names appear at the beginning of the product description. This means that the first term of the product name coincides with the first term of the product description in which it appears. Moreover, 51.5% product names are unigram, 34.2% are bi-gram, and 10.6% are trigram, and 3.7% more than tri-gram. In total 96.3% of the product names are less than trigram. Furthermore, 4.48% of the product names appear at the end of the product description (i.e. the last term of the product name coincides with the last term of the product description).

3.3 Preprocessing

Our preprocessing phase is done in three main steps: (1) punctuation replacement (2) tokenization and (3) IOB tagging. In the punctuation replacement, we replace the following set of punctuation marks with whitespace:

$$\text{punctuations} = \{\text{','} , \text{'.'} , \text{':'} , \text{'-'} , \text{'+'} , \text{'('} , \text{')'} , \text{'='} , \text{'\_'} , \text{'/'} , \text{'\textbackslash'} , \text{'*'} , \text{'['} , \text{']'} \}$$

The decision about the set of punctuation marks that has to be replaced with whitespace, depends on the dataset and the application. Thus in general it is an input to our machine learning framework. In the context of our dataset and application in this work, the punctuation replacement is useful because it normalizes the morphological structure of product names. Based on our manual inspections on the data, punctuations marks are irrelevant features for product names. This means that in many cases the dataset contains both the whitespace form and punctuated form of a product name (e.g. ”o-ring” and ”o ring” product names). For these cases punctuation replacement, yields a more normalized dataset.

In the second step we tokenize each product description using a whitespace tokenizer that yields a list of tokens as its output. In the rest of this work we use the term token to refer to each term or word in product descriptions.

3.3.1 IOB tagging

Each product description may contain multiple product names. The product names of a product description often are related (i.e. they are synonym of each other or one is the hypernym of the other one). The raw dataset has to be processed before being fed into as the input into the learning algorithms. For this purpose, we transform our raw dataset into a processed dataset in the form of (token, tag) pairs. To tag the

tokens of a given product description, we use an In/Out/Begin (IOB) representation [33, 58]. In this format unigram product names are tagged by the label 'B' while in multi-gram product names, the first token is tagged with the label 'B' and the rest of the tokens are tagged with the label 'I'. The tokens which are not part of a product name are tagged with the label 'O'. Therefore, a product name in our new representation always starts with the label 'B' followed by zero or multiple tokens with the label 'I'. Table 3.2 shows our tagging strategy. The 'B' label is specifically needed to distinguish between product names that appear adjacently in the product description (i.e. there is no non-product-name token with label 'O' between them).

Product Desc.	CARROSSERIERONDEL M6 X 30 DIN440R
Product Names	Product Name 1 (PN1) = CARROSSERIERONDEL Product Name 2 (PN2) = DIN440R
Tagged Pro. Desc.	(CARROSSERIERONDEL, 'B'), (M6,O), (X,O), (30,O), (DIN440R,B)

Table 3.2: An example showing how the tokens of a product description are tagged according to the IOB method.

3.4 Feature Construction

To construct our feature set, we follow a structural approach. We first take four classes of features: token-level features, document-level features, gazetteers-based features, frequency-based features as the basis of our feature construction step. These classes are introduced by [59] as the main feature classes for the task of NER. Next, for each class we identify a list of features that are relevant to our product name recognition task. In the rest of this section we elaborate on each feature class. In the next chapter we study the impact and usefulness of different features in our feature set when they are used to train the predictive models of our solution space.

3.4.1 Token-Level Features

Token-level features are related to the character composition of tokens [59]. Token case (i.e. upper or lower-case), numerical and special characters and different morphological features such as prefixes, suffixes are considered as main token-level features. Among them morphological features are specifically interesting for the task of product name recognition, as many product names share the same set of characters as their prefixes or suffixes.

Some of the features mentioned above may not be as discriminative as they are in other NER applications. For example the *case* of the token, is useful mostly

in the applications that follow grammatical rules while this is not the case for the product descriptions in our dataset. Product descriptions are typically ungrammatical and short. Thus we expect, for our dataset, the orthographic features such as capitalization to be very noisy; and so they do not contain considerable discriminative information. A similar hypothesis has been studied and confirmed in [60] for a dataset of queries. We evaluate this hypothesis with respect to our data in the next chapter.

Although we might not benefit from the features that require some levels of grammatical regularity, other groups of features such as morphological features, digit patterns, token length, and the token itself are considered as potentially useful features.

Table 3.3 shows the list of all features used by our predictive model. The first three rows, are variants of the case features. The next row is the token itself which is taken as a feature. Next to that there are numeric features that check if the token is a number or if it contains a numeric part. After these we have the token length as a feature. The rest of the rows in the table are the variants of the morphological feature group. They address different sub-sets of the token. The infix features are taking a sub-part of the token as a feature. We denote the sub-parts by the sequence of token letters $l_i l_j l_k$ where l_i is the i^{th} letter of the token.

3.4.2 Document-level Features

This class considers the features that appear at the document-level (i.e. in the level of product description) where each product description is considered as a separate document according to our definition. Our analysis of the training set reveals that the position of the product names in the product descriptions follows a specific spatial distribution. According to that observation, we define the document-level feature, *token position* as the index of the token t in the token-list V. V is the list of terms that is the outcome of the tokenization of the product description such that the first token in the list, is the first term in the product description. For more details about our tokenization method, we refer to Section 3.3. Note that we consider the feature "token position" as a document-level feature because its evaluation requires document-level information (i.e. it needs the position of the token in the token list).

In addition to the token position, we also consider the previous and the next tokens of the current token as additional document-level features. For this purpose we use a *windowing scheme* with windows size of five that centers at the current token. This creates four new features: the second previous token, previous token, next token, and the second next token.

Features	Description
is-capitalized	it is true if the first letter of the token is capital.
is-all-caps	it is true if all letters of the token is capital.
is-all-lower	it is true if all letters of the token is lower-case.
token	The token itself as a feature
token-numeric	it is true if the token is an integer number
token-digit	It is true if the token contains one or more than one digits
length	The length of the token
prefix1	the first letter of the token.
prefix2	the first two letters of the token.
prefix3	the first three letter of the token.
prefix4	the first four letter of the token.
suffix1	the last letter of the token.
suffix2	the last two letters of the token.
suffix3	the last three letters of the token.
suffix4	the last four letters of the token.
trimmed1	all the letters of the token except the last one.
trimmed2	all the letters of the token except the last two one.
trimmed3	all the letters of the token except the last three one.
trimmed4	all the letters of the token except the last four one.
infix2-1	$l_{n-2}l_{n-1}$ where n is the token length
infix3-1	$l_{n-3}l_{n-2}l_{n-1}$ where n is the token length
infix4-1	$l_{n-4}l_{n-3}l_{n-2}l_{n-1}$ where n is the token length
infix2-2	$l_{n-3}l_{n-2}$ where n is the token length
infix3-2	$l_{n-4}l_{n-3}l_{n-2}$ where n is the token length
infix4-2	$l_{n-5}l_{n-4}l_{n-3}l_{n-2}$ where n is the token length
infix2-3	$l_{n-4}l_{n-3}$ where n is the token length
infix3-3	$l_{n-5}l_{n-4}l_{n-3}$ where n is the token length
infix4-3	$l_{n-6}l_{n-5}l_{n-4}l_{n-3}$ where n is the token length

Table 3.3: Token-level features

The window size is a hyper-parameter and its value may differ from one application to another. In our problem, we choose the windows size based on the initial statistical analysis on the distribution of product names sizes in terms of number of tokens (see Section 3.2). Table 3.4 summarizes the list of document-level features that are present in our feature set.

3.4.3 Gazetteer-based Features

This section, presents how product name gazetteers (e.g. ETIM) are incorporated with machine learning models. This answers the first part of the research question RQ3 (see Section 1.4).

The gazetteers-based approach is one of the main NER approaches. Sometimes when gazetteers are sufficiently complete, they are used as a stand-alone named entity recognizer. However, some researches present hybrid NER approaches where the gazetteers are used in combination with machine-learning techniques [11, 33] to construct more powerful named entity recognizers. In these hybrid solutions, gazetteers are involved as a feature of predictive model. We also follow the same approach. However we extend it, by applying a windowing-scheme to gazetteer-feature. To the best of our knowledge, this work is the first work that studies the use of a product name gazetteer as a feature for the task of PNER.

Gazetteer-based features for an arbitrary term t are defined as the result of the lookup function $Gaz(t)$. The function takes a token as its input and returns true if the token matches with at least one of the tokens in one of the entries of the used gazetteer (i.e. ETIM). Same as document-level features, we use a similar windowing scheme for gazetteer-based features. This enables us to exploit the potential relationship between the neighboring terms. The window size is five and it is centered at the current token. So the window covers two tokens before and after the current token. For each token t_n, the following lookup functions are evaluated: $Gaz(t_{n-2})$, $Gaz(t_{n-1})$, $Gaz(t_n)$, $Gaz(t_{n+1})$, and $Gaz(t_{n+2})$ where Gaz is the gazetteer lookup function. Table 3.5 summarizes our gazetteer-based features.

To be able to effectively use the gazetteer-based features, gazetteer entities have to pass the same punctuation replacement step as the product descriptions. Moreover, the case of the tokens in the product descriptions and the gazetteer entities should be uniformed before matching (i.e. all to upper-case or all to lower-case).

Note that for each product description there are marginal tokens for which the window has some missing tokens. If the windows size is 5, the token t_n is marginal for $n < 2$ and $n > |D|-2$ where $|D|$ is the number of tokens in the product description where the first token is t_0. Some features of the marginal tokens are always evaluated as false because there is no previous or next token or tokens. For example for the token t_1, the feature $G(t_{n-2})$ is always false for all product descriptions.

Features	Descriptions
token position	The index of the token t in the token-list V. This indicates the position of the token in the product description according to our tokenization method
token-position = end	It is true if the current token is the last token of the product description
token-position = pre-end	It is true if the current token is the first previous token of the last token of the product description
token-position = second-pre-end	It is true if the current token is the second previous token of the last token of the product description
pre-token	The previous token
second-pre-token	The second previous token
next-token	The next token
second-next-token	The second next token
pre-token-numeric	The previous token is numeric.
second-pre-token-numeric	The second previous token is numeric.
next-token-numeric	The next token is numeric
second-next-token-numeric	The second next token is numeric.
pre-token-digit	The previous token contains a digit.
second-pre-token-digit	The second previous token contains a digit.
next-token-digit	The next token contains a digit.
second-next-token-digit	The second next token contains a digit.
pre-token-length	The length of the previous token
second-pre-token-length	The length of the previous token
next-token-length	The length of the next token
second-next-token-length	The length of the next token

Table 3.4: Document-level features

3.4.4 Frequency-based Features

Frequency-based features are a class of features that use the frequency of terms in the document as a predictive variable. Our hypothesis is that frequency-level features such as *term-frequency* and *inverse-document-frequency* are informative features that can be used to distinguish the tokens that are part of a product name (i.e. tagged with 'B' or 'I') from non-product-name terms (i.e. tagged with 'O'). More precisely we study if there is a correlation between the frequency of tokens and their tags (i.e. 'I', 'B', or 'O'). Note that we do not specifically claim that product name tokens are the most frequent tokens in the dataset. The important advantage of this

feature class is that these features potentially enable us to exploit an unannotated dataset (if it exists). In our case, we have a larger unannotated dataset that we use to construct frequency-based features.

Features	**Descriptions**
current token in gazetteer	is true if the current token exists in the gazetteers otherwise it is false
previous token in gazetteer	is true if the previous token w.r.t. the current token exists in the gazetteer otherwise it is false
second previous token in gazetteer	is true if the second previous token w.r.t. the current token exist in the gazetteer otherwise it is false
next token in gazetteer	is true if the next token w.r.t. the current token exists in the gazetteer otherwise it is false
second next token in gazetteer	is true if the second next token w.r.t. the current token exists in the gazetteer otherwise it is false

Table 3.5: Gazetteer-based features

We define the Term-Frequency (TF) feature for the token t as the frequency of the token in our unannotated dataset. The numerical value of the term-frequency feature tf for the token t is calculated by the following formula:

$$tf(t) = \frac{C(t)}{|T|} \tag{3.1}$$

where $C(t)$ is the number of occurrences of the token t in the unannotated dataset, T is the set of all tokens, and $|T|$ is its cardinality.

The Inverse-Document-Frequency (IDF) feature uses the same principle, however, instead of term frequency, the inverse document frequency [61, 62] is used. Unlike the term-frequency feature, now we consider each product description as a separate document. The numerical value of the inverse document frequency feature idf for the token t is calculated from the following formula:

$$idf(t) = \ln \frac{|D| + 1}{df(t) + 1} \tag{3.2}$$

where $df(t)$ is the number of documents (i.e. product descriptions) containing the term t in the unannotated, D is the set of all documents, and $|D|$ is its cardinality.

Table 3.6 summarizes our frequency-based features.In the next chapter we study the usefulness of these features and based on our experimental results, we evaluate the hypotheses that are posed in this section.

3.4.5 Hypotheses on Features

This section presents a list of hypotheses on the effectiveness of some of the features. We evaluate these hypotheses based on our experimental results in the next chapter. The hypotheses are:

1. Capitalization features have low discriminative power and are not effective features for the task of product name recognition (discussed in Section 3.4.1).

2. Position matters: there is a significant correlation between the position (i.e. the index) of a token and being part of a product name (discussed in Section 3.4.1).

3. Features that are constructed based on the windowing scheme are effective features (discussed in Section 3.4).

4. Tokens appearing as a part of the product names have a statistical distribution in terms of “term frequency” (tf) or “inverse document frequency” (idf). This can be used as a discriminative feature for product name tagging (discussed in Section 3.4.4).

5. Gazetteer-based features are among the effective features in our feature set (discussed in Section 3.4.3).

In the next chapter, we evaluate these hypotheses.

3.5 Feature Selection

The number of features specifically in case of text data, as well as our problem, tends to increase rapidly to thousands or ten thousands features. In our problem, the number of features grows to 66400 binary features that is almost six times larger than our training samples. This is because we work with text and the features (e.g. the feature ”token” in Table 3.3) need to be binarized (see Section 2.2.1). This large number of features leads to over-fitting. In practice, it turns out that many of these features are noisy features that do not really correlate with our target classes. These noisy features sometimes degrade the performance of our learning model; so having many features sometimes ends up in less efficient and less robust predictive model. This is confirmed by our experimental results discussed in the next chapter. According to the literature, feature selection is one of the effective methods to deal with this problem [36–40]. In this section we explain how feature selection techniques are used in our machine learning solutions.

As discussed in Section 2.2.1, feature selection is a well-known technique to reduce the dimensionality of data (i.e. the size of the feature set). There are three main classes of feature selection techniques: filter methods, wrapper methods and embedded methods. Among those feature selection methods, in this work we specifically focus on the filter and embedded methods. Filter methods are computationally efficient; however they are less accurate as they only consider the importance of the one feature, independent of the other features. The wrapper and embedded methods select the best combination of the features; so they potentially result in a better feature set compared to the filter methods but they are computationally intensive. Between embedded and wrapper methods, embedded methods are more efficient with almost the same feature quality. So between embedded and wrapper methods we choose to investigate embedded methods.

The design of our solution space is based on incorporating both embedded and filter methods; however, our initial experiments reveal that the computational cost of embedded methods is so high that is not feasible for us to generate sufficient experimental results. For completeness, we present our methodology for both feature selection methods (i.e. embedded and filter methods); however, the next chapter, where the experimental results are discussed, only covers the solutions that use filter methods as their feature selection technique.

From the different feature selection techniques in the group of filter methods, we use F-test in this work. F-test is a statistical test based on variance analysis. It is also the most popular feature selection technique in filter methods [44]. This test enables us to determine the features that correlate the best with the target classes. The test assigns a score to each feature (known as Fisher score) based on which a certain subset of important features (i.e. the features with higher scores) are selected. For more background on this topic we refer to Section 2.2.1.

Embedded methods combine filter and wrapper methods. Our embedded feature selector is a combination of F-test and a learning model (i.e. linear or non-linear SVC or Random Forest or AdaBoost). In this feature selection method, filtering of features is done in two steps. We first select a subset of relevant features using a given filter method (i.e. F-test in this work). This reduces the size of feature set. Second for all the possible subsets of the features with a given cardinality that are taken from the already reduced feature set, we train a learning model (e.g. Random Forest or linear SVC, or etc.). At the end the subset with the best performance is chosen as the selected feature set. As the model has already been trained for the best feature set, there is no need to repeat the training step again, and the already trained model is directly used as the predictive model.

Table 3.8 shows our solution space where feature selectors and learning models are combined to construct a solution. The solutions in the rows 2, 4, 6 and 8 use

embedded feature selection while the other rows use filter feature selection method (namely F-test).

3.6 Learning Models

Apart from the quality of the features, the learning models that are used to train the predictive model also play an important role in the overall performance of a machine learning solution. This work investigates the impact of different learning models on the final performance of the product name recognition task. We specifically study: linear and non-linear support vector classification [63, 64], ensemble methods with *bagging* strategy such as random forests [47, 53] and ensemble methods with *boosting* strategy such as AdaBoost [65, 66]. We also investigate the effect of hyper-parameter optimization on the performance of the learning model.

As discussed in Section 2.2.2, SVC is a binary classification algorithm. However, it can be extended to be used as a multi-class classifier by using for example the *crammer-singer* scheme [67][1]. As our problem is a multi-class classification problem, throughout this work we only use multi-class SVC models. To learn more about the different approaches to multi-class SVC, we refer to [68].

3.6.1 Hyper-parameter Optimization

Tuning hyper-parameters of the learning models might be challenging as they may vary in a wide range and they may have influence on each other. The common method, so-called GridSearch, is to choose a set of candidate values for each hyper-parameter based on the intuitive understanding of the hyper-parameter's role on the learning power of the model; then exhaustively search for the combination of the hyper-parameters with the best performance [69].

To compare different combinations of hyper-parameters, a cross-validated scoring function is used. This means that we use cross-validation methodology to split the training set into a certain number of folds. Then we take one fold as the (new) testing set [2] and the rest as the (new) training set. The model is trained using the first set of the hyper-parameters on the new training set. Then the model is evaluated on the new testing set. This is repeated for each fold. At the end, the average of the scores in all folds are taken as the final score for that hyper-parameter combination.

[1] In the sklearn library, this can be done by setting the *multi-class* parameter of the algorithm to "crammer-singer" in case of a linear kernel or by setting the *decision-function-shape* parameter to "ovo" (one-vs-one) in the case of non-linear kernels.

[2] This is the part of our original training data that is used as a testing set for hyper-parameter tuning.

Using the procedure above, we can assign to each hyper-parameter combination a cross-validated score. Finally we sort all the hyper-parameter combinations based on their score and choose the best combination.

3.7 Automatic Machine Learning Framework

This section discusses the machine learning framework developed in this work. We use this framework to study the effectiveness of various features, feature selection techniques, classifiers, and fine-tuning hyper-parameters and number of features. However, in addition to the experimental purposes, the framework is applicable to automatic development of machine learning solutions. This addresses the problem that is known as the AutoML problem [55]. The rest of this section explains our machine learning framework. The core steps of the framework are presented in the form of pseudo code in Listing 1 and 3. Listing 2 shows how the framework is used.

3.7.1 The Skeleton of the Framework

The pseudo code in Listing 1 presents the skeleton of the framework. As input, the AutoML algorithm receives a processed dataset, a list of solutions and an over-fitting threshold. The framework ranges over a given list of solutions and selects the solution with the best performance. The framework uses GridSearch (see Section 3.7.4) and EvaluateSol (see Section 3.7.3) algorithms to optimize hyper-parameters of the solutions.

We explain this process with more details. For each solution, the algorithm iterates over a range of feature percentages starting from 0.5 percent to 100 percent of the features (line 8). This range can be adapted by the user of the algorithm to make it computationally feasible for a specific hardware platform that is used for the experiment. In our experiments, due to limitations in the computational power, we use a smaller range of feature percentages with unequal step sizes. We start with smaller step sizes at the beginning and we increase the step size as the feature percentage grows. This is the range of feature percentages that we use in our experiments: $\{0.01, 0.02, 0.04, 0.06, 0.08, 0.1, 0.3, 0.5, 0.8, 0.9, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5, 8, 8.5, 9, 9.5, 10, 10.5, 11, 11.5, 12, 12.5, 13, 13.5, 14, 14.5, 15, 15.5, 17, 20, 25, 40, 60, 70, 80, 90, 100\}$. Note that the loop over the features, breaks as soon as the over-fitting condition at line 12 is satisfied. This provides some speed ups, as in many cases the learning models over-fit the training set when the dimensionality of data grows (i.e. in large feature sets). Moreover, the iterations of the both loops in the algorithm are independent making them very suitable for parallel execution.

For each feature percentage the GridSearch algorithm searches for the optimal hyper-parameters (see Section 3.7.4, to learn more about GridSearch). The outcome of GridSearch are the optimal hyper-parameters that are stored in the hyper-parameter variable of the current solution (i.e. S[i].CHP). Given the percentage of features and tuned hyper-parameters, the algorithm evaluates the solution S[i] with optimal hyper-parameters and given percentage of features (line 11). Next when the difference between the training and testing errors exceeds the user-defined over-fitting threshold, the algorithm exits the feature loop (lines 8-13) and the optimized hyper-parameters and the last percentage of features are taken as the best hyper-parameter for the i^{th} solution. After repeating this process for all solutions in the list, at the end the algorithm selects the solution with minimum F_1 error as the best solution and returns it as its output while the tuned hyper-parameter and optimal number of features have been stored in the returned object. For brevity, we use constant values for step size and maximum percentage of features (line 11), however, in a more generic framework, these parameters are taken as the inputs of the algorithm.

3.7.2 Dataset Preparations

In order to use the AutoML algorithm, some application-specific preparation steps have to be performed on the input raw dataset. These steps are not part of the generic AutoML framework as they vary from one application to another. For example the format of raw data, how it can be tokenized, how the data samples are going to be represented by features strongly depend on the application. This section explains how the inputs of the AutoML algorithm are prepared in our PNER application. The preparation steps include: preprocessing of raw input dataset and feature construction. These steps are presented in Listing 2.

The preprocessing step (line 4) transforms the raw data into the processed data to be used in the AutoML algorithm. In the raw data, product descriptions are strings attached by a list of integer intervals, each one indicating the start and end index of the product names in the product description string. The preprocessing step transforms these raw annotated product descriptions into a vector of tagged tokens (called processed dataset here-forth). More details are explained in Section 3.3.

The outcome of preprocessing step is the matrix product description $X_{1\times t\times|\mathbf{D}|}$ and the matrix of class labels $Y_{1\times t\times|\mathbf{D}|}$ where t is the number of tokens per product description and $|\mathbf{D}|$ is the number of product descriptions in the dataset. As the number of tokens per product description may vary from one product description to another, we set t to be the maximum number of tokens per product description in our dataset. Product descriptions with less tokens assumed to be padded with dummy tokens and tags. Note that this is only for sake of simplifying the presentation of the algorithm; the real implementation is done based on token lists with unequal sizes.

```
1  Algorithm: AutoML
2  Input:   Dataset, //preprocessed dataset
3           S( classifiers , CHP, feature−selector, EHP, folds), //solution list
4           Threshold //over-fitting threshold
5  Output: Solution
6
7  For i ∈ [1..|S|]:   //ranges over the solutions
8     For f∈ [0.5..100] steps 0.5:   //ranges over the percentage of features f
9        S[i].EHP = f
10       S[i].CHP = GridSearch(S[i].classifier , f , Dataset)
11       Metrics = EvaluateSol(Dataset, S[i])
12       if (Metrics.F1TestErr − Metrics.F1TrainErr) > Threshold:
13          break
14 indx = FindMinErrIndx_{j=1}^{|S|}(Metrics[j].F1TestErr)
15 return S[indx]
```

Listing 1: Proposed Automatic Machine Learning (AutoML) Framework

The feature construction step (line 5) builds up a feature vector for each token. The tags of the tokens are not relevant to this step; so we only pass the token vectors to the function Feature_Construction (line 5). The categories of features used to construct the feature vector are discussed in detail in Section 3.4. The outcome of this step significantly increases the dimensionality of the data (see Section 3.4). Therefore for $X_{1\times t\times|\mathbf{D}|}$ as input, the step generates $X'_{f_1\times t\times|\mathbf{D}|}$ where f_1 is the number of binary features generated for each token. It can increase up to thousands of features.

Then we construct a solution space and store it in the solution vector S (line 6). Each element of S is a solution configuration that includes: a classifier (e.g. Linear SVC), the hyper-parameters of the classifier, a feature selector (e.g. F-test), the hyper-parameters of the feature selector, and the number of cross-validation folds. The solution space and the hyper-parameters used in this work is defined in Table 3.8 and Table 3.7 respectively. However, among the solutions presented in Table 3.8, due to limited computational power, our experiments only cover those solutions that use non-embedded (i.e. F-test) feature selection method.

3.7.3 The Steps of the Evaluation Algorithm

The algorithm presented in Listing 3 is used to evaluate the machine learning solutions. The algorithm is composed of five main steps: feature selection, training,

prediction, post-processing, and evaluation. We briefly explain the function of each step and the notation we use in the algorithm and refer to the related section for more details.

The algorithm implements a cross-validation methodology (lines 10-23). The processed dataset is split into F folds where in each iteration one fold is used for testing and the rest is used for training the model. So in iteration i, if the test fold is $X^{(i)}$, then the training set is $X - X^{(i)}$ (i.e. the whole dataset except the test set). The number of folds is an input parameter that can be determined by the user of the algorithm. However, the most common number of folds is ten; meaning that in each cross-validation iteration, nine folds are taken as the training set and one fold is taken as the testing set. In our experiments in the next chapter, we set the number of folds to ten. For more information about cross-validation we refer to Section 2.2.3.

The feature selection step (lines 13-17) finds the best subset of features to be trained. For filter feature selection methods, we need to determine an *estimator* that in our case is F-test. In the case of embedded feature selection, the given classifier is used for both feature selection and training the model. So the estimator would be irrelevant. The outcome of this step is $X''_{f_2 \times t \times |\mathbf{D}|}$ where f_2 ($f_2 << f_1$) is the number of the selected features. The estimator of the feature selection step is trained over the training set of each fold. Training and prediction are done in lines 18-20 that finally yield a list of predicted classes $\mathbf{Y}'_{1 \times t \times \frac{|\mathbf{D}|}{F}}$. After Post-processing (see Section 3.8.1) in line 21, the performance metrics of the current fold are computed in line 22. At the end, the average performance of the learning model in all folds is returned as the outcome of the algorithm (line 24).

```
1  Input:  Raw Dataset, Threshold
2  Output: A classifier with optimized hyper−parameter
3
4  X_{1×t×|D|}, Y_{1×t×|D|} = Preprocessing(Raw_Dataset)
5  X'_{f1×t×|D|} = Feature_Construction(X_{1×t×|D|})
6  S = Solution_Space_Construction()
7  Dataset = (X'_{f1×t×|D|}, Y_{1×t×|D|})  //preprocessed dataset
8  return AutoML(Dataset, S, Threshold)
```

Listing 2: Preparation steps for AutoML Framework

3.7.4 GridSearch

We use GridSearch to optimize the hyper-parameters of the learning model in our solution space. In this method, we exhaustively search a range of predefined values for each hyper-parameter. The goal is to find the optimal combination of hyper-parameters with which the predictive model has its best performance. Table 3.7 shows the predefined ranges for each hyper-parameter. The details of the algorithm is standard and we do not discuss it any further. For more details we refer to [70].

3.7.5 Solution Space

This section presents our solution space. Each solution is a combination of feature selection methods and learning models. Our goal is to study which solution configuration obtains the best performance. For each configuration, there is a set of hyper-parameters that have to be tuned before running the algorithm. The set of solution configurations that we study in this work is presented in Table 3.8.

Learning model	Hyper-parameter Space
L-SVC	C= 0.0001,0.001,0.01,0.1,1,10,100,1000,10000 $class-weight = \{$balanced, w(O)=1,w(B)=1,w(I)=1$\}$ where balanced = adjust weights inversely proportional to class frequencies in the input data
NL-SVC	C= 0.0001,0.001,0.01,0.1,1,10,100,1000,10000 $class-weight = \{$balanced, w(O)=1,w(B)=1,w(I)=1$\}$ where balanced = adjust weights inversely proportional to class frequencies in the input data $\gamma =$ 0.01,0.1,1,10,100,1/f $f =$ number of features $kernel =$ RBF (Gaussian), sigmoid
Random Forests	d= 10,25,50, 100,150,200 $ntree =$ (5,300), step=10 $max-features =$ sqrt(f), log2(f),(f /3) $f =$ number of features
AdaBoost	d= 10,25,50,100,150 $ntree =$ 15,25,50,75,100,150,200

Table 3.7: The search space of hyper-parameters for different learning models in our solution space

Algorithm: EvaluateSol
Input: Dataset, // processed dataset
Solution_Configuration (classifier ,
CHP, //tuned classifier hyper− parameters
f−selector, // feature selector
EHP, //feature selector hyper−parameters
F //number of folds)
Output: Performance Metrics

$\mathbf{X}'_{f_1\times t\times|\mathbf{D}|} = \{\mathbf{X}'^{(1)}_{f_1\times t\times\frac{|\mathbf{D}|}{F}}, \ldots, \mathbf{X}'^{(F)}_{f_1\times t\times\frac{|\mathbf{D}|}{F}}\}$=Split$(\mathbf{X}'_{f_1\times t\times|\mathbf{D}|}, F)$
$\mathbf{Y}_{1\times t\times|\mathbf{D}|} = \{\mathbf{Y}^{(1)}_{1\times t\times\frac{|\mathbf{D}|}{F}}, \ldots, \mathbf{Y}^{(F)}_{1\times t\times\frac{|\mathbf{D}|}{F}}\}$=Split$(\mathbf{Y}_{1\times t\times|\mathbf{D}|}, F)$
For $i \in$ [1.. F]:
$\mathbf{X}''_{f_2\times t\times|\mathbf{D}|}$= Feature_Selection($\mathbf{X}'_{f_1\times t\times|\mathbf{D}|}$,
$\mathbf{Y}_{1\times t\times|\mathbf{D}|}$,
f−selector($\mathbf{X}'_{f_1\times t\times|\mathbf{D}|} - \mathbf{X}'^{(i)}_{f_1\times t\times\frac{|\mathbf{D}|}{F}}$,
$\mathbf{Y}_{1\times t\times|\mathbf{D}|} - \mathbf{Y}^{(i)}_{1\times t\times\frac{|\mathbf{D}|}{F}}$),
EHP)
LM = Learning_Model(classifier, CHP)
LM.Train($\mathbf{X}''_{f_2\times t\times|\mathbf{D}|} - \mathbf{X}''^{(i)}_{f_2\times t\times\frac{|\mathbf{D}|}{F}}$,$\mathbf{Y}_{1\times t\times|\mathbf{D}|} - \mathbf{Y}^{(i)}_{1\times t\times\frac{|\mathbf{D}|}{F}}$)
$\mathbf{Y}'_{1\times t\times\frac{|\mathbf{D}|}{F}}$ = LM.Test($\mathbf{X}''^{(i)}_{f_2\times t\times\frac{|\mathbf{D}|}{F}}$)
$\mathbf{Y}''_{1\times t\times\frac{|\mathbf{D}|}{F}}$ = Post_Processing($\mathbf{Y}'_{1\times t\times\frac{|\mathbf{D}|}{F}}$)
Metrics[i] = Evaluate($\mathbf{Y}''_{1\times t\times\frac{|\mathbf{D}|}{F}}$, $\mathbf{Y}^{(i)}_{1\times t\times\frac{|\mathbf{D}|}{F}}$)

Metrics = $Avg_{i=1}^{F}$(Metrics[i])

Listing 3: Algorithm used for evaluation of solutions

Our solution space includes eight solutions. Table 3.8 shows these solutions where each row represents the configuration of each solution. Each configuration determines which feature selection and which learning model are used in the solution. As discussed in Section 3.5, we investigate filter and embedded feature selection methods. In filter methods we use the statistical F-test technique. F-test has only one hyper-parameter that is the number of features that should be selected by the statistical test. This is denoted by f_1. Embedded feature selectors first apply the statistical F-test to select an initial set of features with size f_1; then they use a learning model to select a subset of features with size f_2 where $f_2 < f_1$. The

same learning model (with the same set of hyper-parameters) is used to train the predictive model. Hence, the set of hyper-parameters is the same as the hyper-parameters of the learning algorithm. Note that in embedded methods the learning model that is used for feature selection is also used to train the predictive model. The hyper-parameters of the learning algorithms are discussed in detail in Section 2.2.2. For the SVC model with linear kernel we tune the hyper-parameters C, while in the SVC model with non-linear Gaussian kernel both C and γ are optimized. For the tree-based ensemble learning models (i.e. Random Forests and AdaBoost) the hyper-parameters are $ntree$ determining the number of trees (also known as number of estimators), d determining the maximum depth of the trees.

Solutions	**Configurations**
L-SVC1	FS: F-test LM: LinearSVC
L-SVC2 (embedded)	FS: F-test and linearSVC LM: linearSVC
NL-SVC1	FS: F-test LM: Non-linearSVC
NL-SVC2 (embedded)	FS: F-test and Non-linearSVC LM: Non-linearSVC
RF1	FS: F-test LM: Random Forests (RF)
RF2 (embedded)	FS: F-test and Random Forests (RF) LM: Random Forests
AB1	FS: F-test LM: AdaBoost (AB)
AB2 (embedded)	FS: F-test and AdaBoost (AB) LM: AdaBoost (AB)

Table 3.8: Candidate Solution Space

3.8 Evaluation Method

This section discusses our evaluation method. Generally we stick to the standard performance metrics such as precision, recall, and combined metrics such as F_1 score. However, in NER context, these measures are defined at two different levels: (1) phrase-level measure and (2) token-level measure [71]. The first measure is sensitive to the correct detection of the boundaries of the named entities, in our case to the boundaries of product names, while the second measures relaxes this require-

ment. Thus the predictions are counted in the token-level and not in the phrase-level (or product-name-level in our case). We discuss these measures further in this section, but before that we first need to discuss the reasons why we exclude the 'O' class from our calculations.

Excluding the 'O' class from Evaluation. The main objective of our classifier is to recognize product names (i.e. the correct tagging of the tokens in classes 'B' and 'I'). Therefore correctly classified tokens in class 'O' should not be counted in the performance of the classifier. To make it clearer, assume that there is product description with 50 tokens, containing only one product name with two tokens (i.e. there is only one token with tag 'B' and one token with tag 'I' in the product description and the rest of the tokens are in class 'O'). If a classifier predicts that all 50 tokens are in class 'O', its precision is 96% while we cannot consider it as a good classifier, because it obviously missed the only product name in the product description and the fact that it was able to correctly tag all the tokens in class 'O' does not add any value to this classifier as a product name recognizer. This gets more important when the number of tokens in class 'O' are many more than the other two classes, which is indeed the case in our annotated dataset. To address this issue, we customize the standard metrics such that only the 'B' and 'I' classes are taken into account.

Token-level Measurement. For each product description, we calculate true positives, false positives, and false negatives, only for the tokens tagged with either 'B' or 'I'. Based on those values the precision and recall for each product description pd are calculated using the following equations:

$$\mathsf{Precision}_{\mathsf{pd}} = \frac{\Sigma_{\mathsf{i}\in\{B,I\}}\mathsf{TP}_{\mathsf{i}}}{\Sigma_{\mathsf{i}\in\{B,I\}}\mathsf{TP}_{\mathsf{i}} + \Sigma_{\mathsf{i}\in\{B,I\}}\mathsf{FP}_{\mathsf{i}}} \tag{3.3}$$

$$\mathsf{Recall}_{\mathsf{pd}} = \frac{\Sigma_{\mathsf{i}\in\{B,I\}}\mathsf{TP}_{\mathsf{i}}}{\Sigma_{\mathsf{i}\in\{B,I\}}\mathsf{TP}_{\mathsf{i}} + \Sigma_{\mathsf{i}\in\{B,I\}}\mathsf{FN}_{\mathsf{i}}} \tag{3.4}$$

where the set $\{B, I\}$ is the set of classes (i.e. token tags), TP_{i}, FP_{i}, FN_{i} are the number of true positives, false positives and false negatives computed for the class i respectively.

At the end, when precision and recall for each product description are calculated, the final precision and recall for a solution is computed as the average of the precision and recall of all product descriptions in the testing set respectively. This can be calculated using the following equations:

$$\mathsf{Recall} = \frac{\Sigma_{t=0}^{N_{pd}}\mathsf{Recall}_t}{N_{pd}} \tag{3.5}$$

$$\text{Precision} = \frac{\Sigma_{t=0}^{N_{pd}} \text{Precision}_t}{N_{pd}} \tag{3.6}$$

where N_{pd} is the total number of product descriptions in the testing set.

Phrase-level Measurement. Phrase-level measurement evaluates how successful the classifier is in tagging full product names. Thus, a prediction is correct only if all tokens of the product name are predicted correctly. To calculate phrase-level measures, we develop our own algorithm that checks the sequence of tags and matches them with the corresponding sequence in the ground truth. More specifically the algorithm finds the sequence of tags starting with a 'B' tag followed by zero or multiple 'I' tags in the predicted data and matches it against the ground truth. Only if the whole sequence matches completely, it is counted as one successful prediction.

We explain how these two measures (i.e. token-level and phrase-level measures) are calculated by bringing an example in Table 3.9. The first row of the table in the example shows the tagging of an example product description taken from the testing set (ground truth). The product description is shown as a list of (token,tag) pairs where t_1 is the first token of the product description and t_{12} is the last token. The second row shows the same product description assumed to be tagged by one of our solutions. The third row shows the precision and recall calculated based on token-level and the fourth row shows the same metrics calculated based on phrase-level different computation methods.

In our evaluations we use both token-level and phrase-level metrics. We believe that each of them evaluates the performance of the predictive model from its own specific perspective. The phrase-level metrics shows how good a solution is in tagging a complete product name while the token-level is more relaxed and gives an insight into the performance of a classifier in the partial tagging of the product names.

Ground-truth	(t1,B),(t2,I)(t3,I)(t4,O),(t5,O),(t6,O) (t7,B),(t8,I)(t9,O),(t10,O),(t11,O),(t12,O)
Prediction	(t1,B),(t2,I)(t3,I)(t4,O),(t5,O),(t6,O) (t7,B),(t8,O)(t9,O),(t10,O),(t11,O),(t12,O)
token-level	Precision =(2+2)/((2+0)+(2+0)=4/4=1 Recall= (2+2)/((2+0)+(2+1))=4/5=0.8
phrase-level	Precision = 1/(1+0)=1 Recall=1/(1+1)=1/2

Table 3.9: Example of Evaluation Methods

3.8.1 Post-processing

The post-processing is the final step before performance evaluation. This steps takes the prediction result produced by the solution and improves the quality of classification by applying a set of predefined rules. Our post-processing step uses only one rule that stems from the fact that in IOB tagging system, it is meaningless to have a product name prediction that starts with a token tagged by the label 'I'. The question is how this can be repaired. Multiple options can be considered: (1) flip the token label to 'B', (2) flip label of the previous token (that has already been labeled with 'O') to 'B' and keep the label of the current token intact. The rationales behind the rules are different. in the first case, we assume the current token has been the initial token that is wrongly tagged as 'I'. So the predicting model has failed to distinguish between the 'I and 'B' classes. In the second rule, we assume that the predictive model correctly tagged the current token as 'I' but it failed to tag the previous token correctly, so the failure is to distinguish between the 'B' and 'O' classes. According to our experiments and visual inspections of the results, the second rule yields a better performance.

Features	Descriptions
token-tf-range[0-0.2)	It is true if the term frequency for the current token is in the range of 0 to 0.2.
token-tf-range[0.2-0.4)	It is true if the term frequency for the current token is in the range of 0.2-0.4.
token-tf-range[0.4-0.6)	It is true if the term frequency for the current token is in the range of 0.4-0.6.
token-tf-range[0.6-0.8)	It is true if the term frequency for the current token is in the range of 0.6-0.8.
token-tf-range[0.8-1.1]	It is true if the term frequency for the current token is in the range of 0.8-1.1.
token-tf-range-greater-1.1	It is true if the term frequency for the current token is greater than 1.1.
token-idf-range-less-3.8	It is true if the inverse-document-frequency for the current token is less than 3.8
token-idf-range[3.8-4.8)	It is true if the inverse-document-frequency for the current token is in the range of 3.8-4.8.
token-idf-range[4.8-5.8)	It is true if the inverse-document-frequency for the current token is in the range of 4.8-5.8.
token-idf-range[5.8-6.8)	It is true if the inverse-document-frequency for the current token is in the range of 5.8-6.8.
token-idf-range[6.8-7.8)	It is true if the inverse-document-frequency for the current token is in the range of 6.8-7.8.
token-idf-range[7.8-8.8)	It is true if the inverse-document-frequency for the current token is in the range of 7.8-8.8.
token-idf-range[8.8-9.8)	It is true if the inverse-document-frequency for the current token is in the range of 8.8-9.8.
token-idf-range[9.8-10.8)	It is true if the inverse-document-frequency for the current token is in the range of 9.8-10.8.
token-idf-range[10.8-11.8)	It is true if the inverse-document-frequency for the current token is in the range of 10.8-11.8.
token-idf-range[11.8-13]	It is true if the inverse-document-frequency for the current token is in the range of 11.8-13.
token-idf-range-greater-13	It is true if the inverse-document-frequency for the current token is greater than 13

Table 3.6: Frequency-based features

Chapter 4

Results

This chapter discusses our experimental results. We compare the performance of the machine learning solutions presented in Table 3.8). However, due to limited computational power, we only address the solutions that use non-embedded feature selectors. Then we explain how the optimal number of features is selected and what is the effect of hyper-parameter optimization on the performance of predictive models. We also analyze the effectiveness of different feature groups introduced in the previous chapter and evaluate the hypotheses presented on the feature set. Throughout this chapter we answer the research questions of this research (see Section 1.4).

4.1 Evaluation of Solutions

This section compares the performance of the studied solutions (i.e. the solutions with non-embedded feature selectors in Table 3.8). Table 4.1 show token-level and phrase-level F_1 score of each solution. We refer to Table 3.8 for more details on the configuration of the studied solutions. The table 4.1 also shows the optimal hyper-parameter values for each solution. The table also lists the optimal number of features as one of the solution hyper-parameters. In the next section we discuss, how the optimal number of features are selected for each solution.

Based on the results, the NL-SVC1 solution achieves the best performance both in terms of token-level and phase-level measures. The results also reveal that the difference between maximum and minimum scores is 6%. This implies that role of learning models does not create a considerable performance difference at least on the studied dataset.

Discussion. Our expectation was that the complex learning models such as NL-SVC, RF and AB may induce more powerful learning models, however, the results

reveal that this is not really the case and in fact the less complex L-SVC model works as good as the other learning models. When we analyze the training errors of the four learning models, we see that the RF and AB models are able to effectively learn the patterns of the training data. So they achieve training errors of almost 1 percent for AB and 7 percent for RF. However for these models the testing error remains relatively high and does not decrease proportionally with the training error. We relate these results to three factors: (1) insufficient training samples, (2) weak feature set, and (3) non-optimal hyper-parameters. The third factor is less likely to play an important role. The reason is that the hyper-parameters of the learning models are tuned over a decent range of values. We believe that the second item also does not have a major effect. The reason is that we use a large feature set that addresses a wide range of features. Moreover, the constructed feature set is sufficiently strong that the learning models are able to effectively learn the pattern of training data. Therefore, we conclude that among the listed factors, the first factor (i.e. insufficient training samples) has the largest influence on the performance of the learning models.

Note that in tuning continuous values, there is no guarantee to find the global optimum by only searching over discrete values. However, searching a broad range of hyper-parameter values, one can more or less ensure that the selected hyper-parameters are very close to the optimal values.

In the next sections, we discuss that although the models are relatively similar in terms of performance, they are not equal in terms of robustness to the changes in the number of features and also the choice of non-optimal hyper-parameters.

Linearly separable dataset. The high dimensionality of our data does not allow us to have a visualization of dataset samples in the feature space. However, based on the performance result of L-SVC and NL-SVC we can conclude that the training samples of our dataset are linearly separable in the constructed feature space. The rationale behind this is the fact the L-SVC and NL-SVC have almost equal performances.

4.2 Determining the Optimal Number of Features

Section 2.2.3 discusses how we determine the optimal number of features for each solution. A number of features is optimal if the model generated based on that is neither over-fitted nor under-fitted the training data. It is difficult to determine the optimal number of features precisely. However, the analysis of training and testing errors gives a good approximation. In this section, we perform this analysis for each

Solutions	Token-level F_1-score	Phrase-level F_1-score	Optimal Hyper-Parameters
L-SVC1	0.77	0.64	f= 8% of total features C = 0.1 $Kernel$ = linear
NL-SVC1	0.78	0.65	f= 4.5% of total features, C = 1000, γ =1/4000 $Kernel$ = RBF (Gaussian)
RF1	0.76	0.62	f = 2% of total features, $ntree$ =25, d =50, $nfeature$ = f/3
AB1	0.73	0.59	f = 4% of total features, $ntree$ =25, d =50, $base - estimator$ =DecisionTree

Table 4.1: Performance comparison of studied solutions in terms of F_1 score

solution and determine the optimal number of features for each solution. We also demonstrate the over-fitting problem for the studied solutions.

Figure 4.1 (a-d) shows the testing and training errors for the solutions L-SVC1, NL-SVC1, RF1, and AB1 respectively. For all solutions, the statistical F-test is used as the feature selector and the tuned hyper-parameters in Table 4.1 are used. Each graph in the figure, shows the training and testing error in terms of token-level and phrase-level measures where the X-axis is the percentage of features used to train the learning model; and Y-axis is the prediction error that is computed from the following equation $E = 1 - F_1$. For more details on how the F_1 is computed for phrase-level and token-level measures, we refer to Section 3.8.

In graph 4.1 (a), L-SVC1 for both measures, the training and testing errors decrease by the number of features; however, not with the same rate. The training error decreases dramatically by the number of features while the testing error, after its initial reduction, remains constant. Large differences between the training and testing error are an indication of over-fitting. The graph shows that the model over-fits the train data at large feature numbers, especially for the full feature set (i.e. 100% of features). This supports the ”rule of thumb” that states that the learning models over-fit the training data when the number of features is significantly larger than the number of training samples.

It is difficult to determine an exact point where the model starts to over-fit the training data. However, according to the graph, choosing 8% of the feature set gives a good approximation for the optimal number of features for the solution L-SVC1.

This is the minimum number of features at which the predictive model reaches its best performance on the unseen data. As the size of feature set is relatively small with respect to the training sample, and the difference between training and testing error is in an acceptable margin, we can safely ensure that the model is not over-fitted at that point. With this number of features, our predictive model is sufficiently complex to be trained effectively over the training data, and at the same time it is not too complex to over-fit the details of the training data.

Graph (b) in Figure 4.1 shows how the training and testing errors change with the number of features for the solution NL-SVC1. The solution has the same configuration as the solution L-SVC1 except that it uses a non-linear SVC model with RBF (Gaussian) kernel. Based on the error analysis, we select 4.5% as the optimal number of features for the solution NL-SVC1.

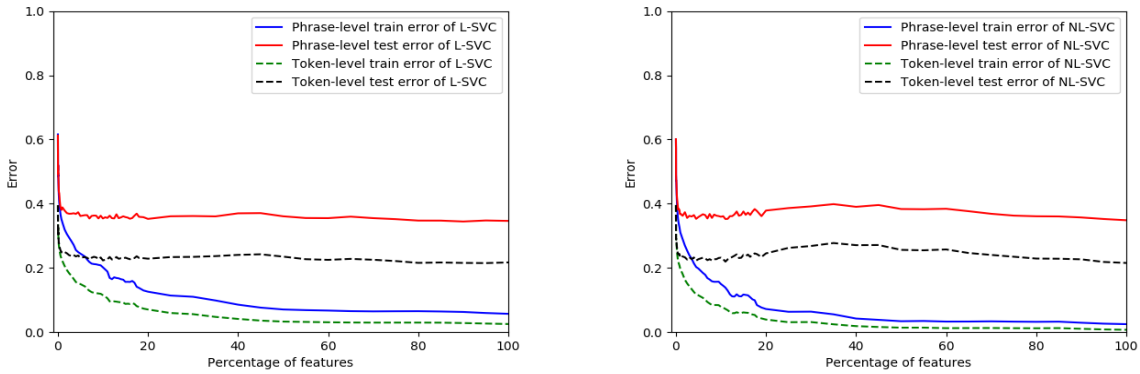
Graph (c) in Figure 4.1 shows the training and testing errors for solution RF1. Compared to SVC models, Random Forest fits much better and with fewer features to the training data, however, the performance of the model on the unseen test data is not better than linear and non-linear SVC models. In fact the RF model quickly over-fits the training data, when only about 2% of the features is used. This makes the SVC models preferable over the Random Forest model in our application, as they are less vulnerable to over-fitting compared to the Random Forest model.

Graph (d) in Figure 4.1 shows the training and testing errors for solution AB1. The behavior of the model in different number of features is very similar to the Random Forest model. However, AdaBoost has a bit weaker performance compared to the Random Forest model.

4.3 The Effect of Hyper-parameter Optimization

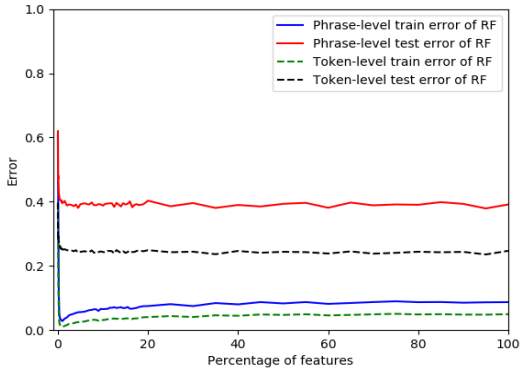
Each learning model in our solution space has a set of hyper-parameters. The hyper-parameters are tuned for our dataset to improve the performance of the predictive model on unseen data. Hyper-parameters play an influential role on the performance of learning models [72]. However, tuning of hyper-parameters is a computational-intensive task, so the question is that how much we can gain from it? This section answers this question in the context of our PNER application, by analyzing the performance of linear and non-linear SVC, Random Forests, and AdaBoost classifiers with or without hyper-parameter tuning.

We first evaluate the performance of the solutions L-SVC1, NL-SVC1, RF1 and AB1 with default hyper-parameters.The default values of the hyper-parameters for each learning model are taken from sklearn library [73] (i.e. we use the default values used by sklearn). Next, we repeat the evaluation for the same set of solutions, however, this time we use the optimal values for hyper-parameters. By comparing

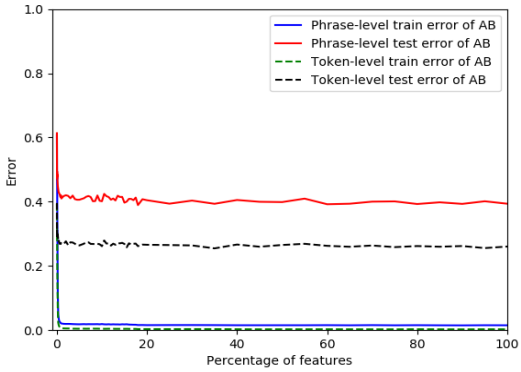


(a) L-SVC1

(b) NL-SVC1



(c) RF1



(d) AB1

Figure 4.1: Training and testing error analysis for solutions L-SVC1, NL-SVC1, RF1, AB1 (feature selector: F-test, learning model). The graphs show how the training and testing error change when the percentage of features increases.

these evaluations, we can conclude how much each solution can benefit from hyper-parameter tuning. For more details on our method for hyper-parameter optimization, we refer to Section 3.7.

Table 4.3 shows the results of the experiment. The results are generated for the optimal number of features for each learning model as presented in Table 4.1. The default and tuned hyper-parameters for each solution are presented in Table 4.2.

According to the table, as expected, all models have better performance when they are trained with the tuned hyper-parameters. However, the effect of tuning is not the same for different learning models. Some models, namely nonlinear SVC and AdaBoost, benefit more from the hyper-parameter tuning than the others. In fact these two models under-fit the training data when they are trained with default hyper-parameters. This means that they fail to learn the patterns in the data (i.e. the relation between the features and the tag labels). So they cannot even be trained

Solution	Tuned-HP	Default-HP
L-SVC1	$C = 0.1$ $class - weight =$ w(O)=w(I)=w(B)=1	$C = 1.0$ $class - weight =$ w(O)=w(I)=w(B)=1
NL-SVC1	$C = 1000$, γ=1/4000 $class - weight =$ w(O)=w(I)=w(B)=1	$C = 1.0$, γ=1/f, f is the number of features $class - weight =$ w(O)=w(I)=w(B)=1
RF1	$ntree =$25, $d =$50, $random - state =$42, $max - nfeature =$ (f/3), f is the number of features	$ntree =$10, $d =$None, $random - state =$None, $max - nfeature =$ sqrt(f), f is the number of features
AB1	$ntree =$ 25, $d =$ 50, $base - estimator =$ DecisionTree	$ntree =$ 50, $d =$None, $base - estimator =$ DecisionTree

Table 4.2: Default and tuned hyper-parameters

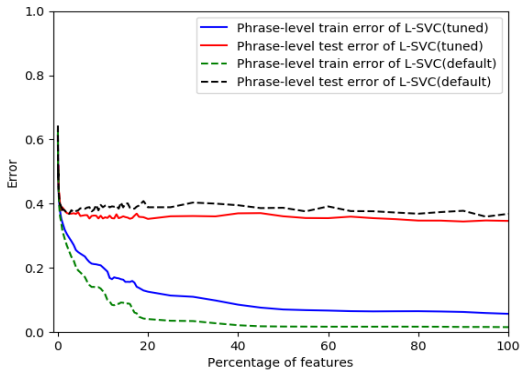
Solution	Tuned-HP	Default-HP
L-SVC1	0.64	0.63
NL-SVC1	0.65	0.54
RF1	0.62	0.61
AB1	0.59	0.53

Table 4.3: Comparison of solutions with default and tuned hyper-parameters in terms of F_1 measure

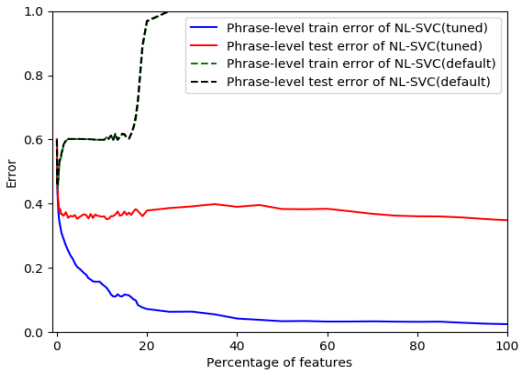
effectively (i.e. very high train error even with very large feature set (see Figure 4.2)).

Moreover, one observation that can be made based on the Figure 4.2 is the relationship between the robustness of the learning models against the changes in the number of features and the hyper-parameters of the model. For example, the non-linear SVC model (NL-SVC1) with tuned hyper-parameters is robust against the changes in the number of features while the performance of the same model trained with default hyper-parameters highly depends on the used number of features. This behavior is illustrated in graph (b) in Figure 4.2. Another important factor is the learning model itself. For example, as it is shown in graph (a) in Figure 4.2, the linear SVC model (L-SVC1) is robust against the changes in the number of features; no matter if it is trained with the default or tuned hyper-parameters.

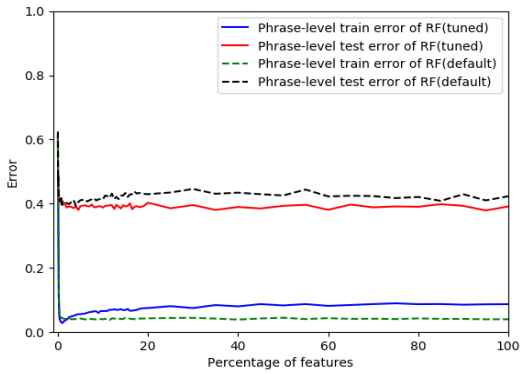
One conclusion of this experiment is that tuning hyper-parameters for the solutions AB1 and NL-SVC1 is necessary as they under-fit the training data (i.e. they fail to effectively learn the patterns in the training data) when they are trained with the default hyper-parameters. For the solutions L-SVC1 and RF1, hyper-parameter optimization provides a better performance, however it can be ignored in case of



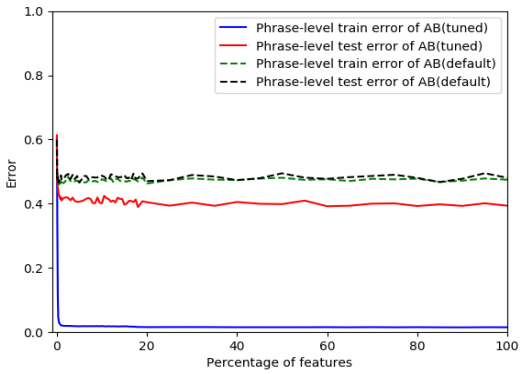
(a) L-SVC1



(b) NL-SVC1



(c) RF1



(d) AB1

Figure 4.2: The behaviour of learning models when they are trained with tuned and default hyper-parameters

strict limitations on the computational power. Another conclusion of this experiment is that, when learning models are trained with optimal hyper-parameters, they are more robust against the changes in dimensionality of data (i.e. number of features). These conclusions are only valid in the context of this study. However, the developed framework is sufficiently general that can be used to draw more general conclusions if it is supported with more annotated datasets.

4.4 Feature Analysis

The previous chapter presented a set of features that were used to construct a predictive model for product name recognition. Features were organized in four main groups: token-level features, document-level features, frequency-based features, and gazetteer-based features. To achieve a better understanding about our feature set, in this section we systematically analyze the effectiveness of different feature

groups and we evaluate the hypotheses presented in Section 3.4.5.

In this experiment, for each candidate feature group we compare the performance of the solution NL-SVC1 (the best predictive model in our feature space), with two feature sets: (1) the feature set that includes all feature groups and (2) the feature set excluding the candidate feature group. In this way we measure the effectiveness of one feature group (i.e. the candidate feature group) in comparison with other features. This method to analyze the effectiveness of features is used also in other researches such as [74]. Additionally, we add another perspective to this experiment by presenting the top 50 most effective features ranked based on the Fisher score[1] [44] in Table 4.5. We also present 50 selected features among the top 200 most effective features in the same ranking in Table 4.6. Later in this section we use these tables to compare the effectiveness of different feature groups and also to evaluate the hypotheses proposed in Section 3.4.5.

Table 4.4 shows the contribution of each feature group to the overall performance of the solution NL-SVC1 in terms of precision, recall and F_1 score. According to the table, document-level features are the most effective feature groups with a contribution of 14.8%. Token-level features are in the second position with 6.8% contribution. The contribution of frequency-based and gazetteer-based features is 0.5% and 1% and is negligible compared to other feature groups.

The reason behind the significant contribution of document-level features is due to the features that are related to the position of the tokens. This is confirmed by our complementary experiment (i.e. feature ranking using Fisher score) where the features related to "token positions" appear at the positions, 1, 9, 44, 97, 98, 105, 109, 112, 114, 178 in table 4.6. The contribution of the token-level features is mostly from: (1) the morphological features such as the use of "DIN", "RING" and "VIS" as the prefix or suffix (at the positions 10, 25, 33, 35, 95, 152, 160, 173, 194 in table 4.6) and (2) numeric features at the positions 6, 8 and 24.

To understand better the reason behind the poor performance of gazetteer-based features, we analyze the product name coverage of ETIM to measure the completeness of the used gazetteer with respect to our dataset. According to our measurement 16% of the annotated product names in our dataset are mentioned in ETIM dictionary. This means that ETIM dictionary is relatively incomplete with respect to our dataset. We believe the low coverage of ETIM dictionary is the main reason behind the poor contribution of gazetteer-based features to the overall performance of the predictive model. The question how the contribution of gazetteer-based features to the performance of machine-learning-based solutions improves by the completeness of gazetteers is an interesting open question that can be seen as one of the possible future work of this thesis.

[1] For more details about Fisher score see Section 2.2.1.

Solution	precision	recall	F_1 score	contribution to F_1 score
NL-SVC1-AF	0.666	0.644	0.646	-
NL-SVC1-ExTok	0.578	0.594	0.578	6.8%
NL-SVC1-ExDoc	0.572	0.492	0.498	14.8%
NL-SVC1-ExFreq	0.661	0.638	0.641	0.5%
NL-SVC1-ExGaz	0.659	0.631	0.636	1%

Table 4.4: The effectiveness of different feature groups

It is important to notice that features work in combination with each other. Sometimes their contributions to the overall performance of the model overlap and sometimes add up. Thus, it is incorrect to conclude that gazetteer-based features or frequency-based features have small discriminative power. In fact our complementary experiment (i.e. feature ranking using Fisher score) reveals that some of the features from frequency-based feature and gazetteer-based feature groups are present in the top 200 features ranked based on Fisher score. These are the features at the positions 26, 47, and 103 for gazetteer-based features, and at the positions 84, 87, 106, and 126 for frequency-based features in Table 4.6. This means that some of the features in these two groups have relatively good discriminative power.

At the end of this section we validate the list of hypotheses on the relevance and effectiveness of some families of features presented in Section 3.4.5.

Hypothesis: capitalization features are not effective features for the task of product name recognition.

Evaluation: this hypothesis is not correct. Despite the fact that product descriptions in our dataset are ungrammatical, ”is-all-lowercase” and ”is-capitalized” features are present at the positions 27 and 196.

Hypothesis: position matters; there is a significant correlation between the position (i.e. the index) of a token and being part of a product name.

Evaluation: this hypothesis is correct. Based on Table 4.6, features related to the position of tokens are in the ranks: 1, 9, 44, 97, 98, 105, 109, 112, 114, 178. This shows that the position of features is a useful discriminative features.

Hypothesis: features that are constructed based on the windowing scheme are amongst the effective features.

Evaluation: this hypothesis is correct. According to Table 4.6, the features created based on the windowing scheme, are present in the ranks: 6, 7, 8, 12, 13, 26, 40, 72, 76, 119, 145, 152, 160, 183, 186 and 188. Therefore we conclude that this family of features contains effective features and windowing scheme, as expected, is useful extension to the features in other feature groups (i.e. token-level features or gazetteer-based features). Thus, the hypothesis is strongly confirmed by our

observations.

Hypothesis: tokens appearing as a part of the product names have a statistical distribution in terms of ”term frequency” (TF) or ”inverse document frequency” (IDF) that can be used as a discriminative feature.

Evaluation: this hypothesis is correct. Table 4.6 shows that for the tokens that appear in the product names, the IDF values in the range of 4.5-5.8, 5.8-6.8, 10.8-11.8 and 9.8-10.8 appear at the positions 84, 87, 106, 126 respectively.

Hypothesis: gazetteer-based features are effective features.

Evaluation: this hypothesis is correct. Three features from the group of gazetteer-based features are present in the ranking at the positions: 26, 47, and 103.

Another observation is that the feature selection method was able to select interesting morphological patterns among all the possible permutations of characters. For example the feature ”prefix3=DIN” with rank 10, has been automatically selected by the F-test among all the possible permutations of a three-character prefix.

Feature	Rank	Feature	Rank
token-position=0	1	second pre token in gazetteer	26
pre-token=”	2	is-all-lower	27
pre-token-length=”	3	second-next-token-length=”	28
second-pre-token-length=”	4	second-next-token=”	29
second-pre-token=”	5	length=2	30
pre-token-numeric	6	token-numeric	31
pre-token-digit	7	suffix4=DIN	32
second-pre-token-numeric	8	token=DIN	33
token-position=1	9	prefix4=DIN	34
prefix3=DIN	10	suffix3=DIN	35
trimmed4=”	11	trimmed2=D	36
previous token in gazetteer	12	infix2-2=D	37
second-pre-token-digit	13	infix4-2=D	38
prefix1=D	14	infix3-2=D	39
infix2-3=”	15	second-next-token-numeric	40
trimmed3=”	16	infix3-1=DI	41
infix3-3=”	17	infix4-1=DI	42
infix4-3=”	18	suffix2=IN	43
prefix2=DI	19	token-position=end	44
infix4-2=”	20	next-token-length=”	45
infix2-2=”	21	next-token=”	46
trimmed2=”	22	infix2-1=DI	47
infix3-2=”	23	next-token-numeric	48
token-digit	24	suffix2=NG	49
pre-token=DIN	25	infix2-1=IN	50

Table 4.5: List of the top 50 most effective features ranked based on Fisher score

Feature	Rank	Feature	Rank
token-position=0	1	token-position=2	97
pre-token-numeric	6	token-position=5	98
pre-token-digit	7	length=7	104
second-pre-token-numeric	8	token-position=pre-end	105
token-position=1	9	token-idf-range(9.8-10.8)	106
prefix3=DIN	10	length=9	108
previous token in gazetteer	12	token-position=4	109
second-pre-token-digit	13	token-position=3	112
token-digit	24	token-position=6	114
pre-token=DIN	25	pre-token=Zeskant	119
second previous token in gazetteer	26	prefix3=933	123
is-all-lower	27	token-idf-range(5.8-6.8)	126
length=2	30	length=14	136
token-numeric	31	current token in gazetteer	143
token=DIN	33	second next token in gazetteer	145
suffix3=DIN	35	token=tapbout	151
second-next-token-numeric	40	pre-token=din	152
token-position=end	44	pre-token=VIS	160
next-token-numeric	48	token=VIS	173
next-token-digit	56	token-position=7	178
pre-token-length=2	72	second-next-token-digit	183
pre-token-length-3	76	pre-token=staal	186
token-idf-range(4.8-5.8)	84	next-token=933	188
token-idf-range(10.8-11.8)	87	prefix3-VIS	194
suffix4=RING	95	is-capitalized	196

Table 4.6: List of the selected 50 feature among the top 200 most effective features ranked by Fisher score (used for the evaluation of feature hypotheses)

Chapter 5

Conclusions and Future Work

This chapter presents a summary of the main contributions of this thesis and ends with discussing future research directions.

5.1 Conclusions

Hybrid NER Solutions. The first contribution of this thesis is to design a set of hybrid solutions where both machine learning and gazetteer-based approaches are combined. For this purpose, product name gazetteers such as ETIM are incorporated into the predictive model in the form of a set of gazetteer-based features. This answers the first part of the research question RQ3 (i.e. how can gazetteers be incorporated into the predictive model?). We specifically use AdaBoost, Random Forest, linear and non-linear SVC as the learning algorithms. Moreover, to answer the second research question RQ2 (i.e. among the learning models chosen for this study which one induces a better predictive model?), we investigate and compare the performance of different solutions to understand the role of the learning algorithms. According to the experimental results, the performance of the solutions varies between 59% for the solution that uses AdaBoost and 65% for the solution that uses non-linear SVC in terms of F_1 score. This shows that in the context of our experiments, the choice of the learning algorithm does not have a large impact on the final performance of the induced model.

AutoML Framework. To be able to investigate different aspects of the proposed solution space, we developed a machine learning framework. The framework is used to automatically determine the optimal number of features based on a user-defined over-fitting threshold, optimize the model hyper-parameters and evaluate the performance of different solutions for a given (processed) annotated dataset. By optimizing the hyper-parameters of each solution, we make sure that they are

all compared in their optimal settings. We use our framework to investigate and compare the performance of solutions, however, the framework is sufficiently generic to be used as an automatic machine learning (AutoML) framework. In this way most of the required decisions in the process of designing a machine-learning solution are automated. Thus making machine learning-based solutions for a new application is reduced to pre-processing the raw input data and constructing an effective feature set. The AutoML framework is able to automatically select the optimal feature set, optimize the hyper-parameters and select the best learning algorithm for a given processed dataset.

Effectiveness of Different Feature Groups. We use a structural approach in constructing our feature set. Accordingly the features are organized in four groups: token-level features, document-level features, gazetteer-based features, and frequency-based features. In our experiment, we measure the contribution of each feature group to the overall performance of the non-linear SVC solution (i.e. the solution with the best performance). According to our results, the most effective feature groups are the document-level features with 14.8% contribution to the F_1 score and token-level features with 6.8% contribution. The gazetteer-based features with 1% contribution and frequency-based features with 0.5% contribution are the less effective feature groups. These measurements answer research question RQ1 (i.e. what are the main discriminating features representing a predictive model for product names in our dataset?) and the second part of the research question RQ3 (i.e. to what extent can this improve the performance of product name recognition?).

Additionally, we dig into the reasons behind the relatively poor performance of gazetteer-based features. For this purpose, we measure the product name coverage of the used gazetteer, ETIM, with respect to our dataset. The result reveals a product name coverage of 16% (i.e. only 16% of product names in the annotated dataset are mentioned in ETIM). This explains why the performance of gazetteer-based features is not so considerable compared to other feature groups. To complete these analyses, we rank all the selected features based on the Fisher score. This adds another perspective to the effectiveness of features where some of the gazetteer-based and frequency-based features appear in first 50 top most features in the ranking. From this we can conclude that these features have good discriminative power but they overlap with some other features, thus their contributions to the performance of the predictive models are not significant when they are compared with other feature groups

The Effect of Hyper-parameter Tuning. Optimization of hyper-parameters is a computationally intensive task. However, it is not clear when hyper-parameters

optimization causes significant improvement on the performance of the predictive model. In general this depends on the dataset and learning algorithm. According to our experiments, the performance of all learning models used in this study improves when they are trained with the tuned hyper-parameters. However, the effect of tuning is not the same for different learning models. Some models, such as non-linear SVC and AdaBoost, benefit more from the optimization of their hyper-parameters. In fact these two models under-fit the training data when they are trained with their default hyper-parameters. This answers the research question RQ4 (i.e. what is the role of hyper-parameter optimization?).

Suggestions for Mydatafactory. As explained before, this work is motivated by the company Mydatafactory. The company is interested in product name recognition especially from short and multilingual product descriptions. In this work, we use a dataset of product descriptions that is provided and partly annotated by the company. Therefore, the result of this research is more useful for the company compared to similar researches that have used different datasets. Specifically, this thesis suggests a hybrid product named entity solution that combines the gazetteer-based and machine learning approaches. The performance of the hybrid solution can be improved in two ways: first by expanding the training set, and second by enriching the used gazetteers. Moreover, the research questions of this work address some of the important decisions that have to be made in the process of designing a product name recognition system. The answers presented in this work, provide useful hints and directions that can be useful for the company to have better design choices.

5.2 Future Work

This section summarizes some promising future research directions.

Relation Extraction. Assume that there are two or multiple datasets where some relations between the product descriptions have already been identified. Then one may use our product name recognition system to tag the product names in the product descriptions and then investigate how the relationship between the product descriptions may imply a relationship between the recognized product names. For example from the fact that a product description, that describes a request for a product, matches with a product description in a supplier catalog, we may infer that the product names recognized in those product descriptions are synonym (i.e. they refer to the same physical product or service). Similarly other types of relations such as hypernym or part-of (i.e. a product is part-of another product) can also be ex-

tracted. As an additional step, the relations between product names extracted in this way, may be used to construct a Word-Net or an ontology for the domain of product names, or may be employed to enrich current product name gazetteers.

Extensions to the Machine Learning Framework. The proposed machine learning framework is able to compare the performance of a given set of machine learning solutions. This can be extended such that instead of comparing different machine learning solutions, the framework constructs an ensemble out of the given solutions. Then it automatically combines the predictions of different solutions using a voting mechanism. In this way the predictions of different learning models complement each other; that may result in a stronger predictive model.

Gazetteers Extracted from Wikipedia. One can study the effectiveness of a multilingual gazetteer extracted from Wikipedia, for the task of product name recognition. As an initial idea, it can be considered if a token in the product description is linkable in Wikipedia. For this purpose, different wikification [75] tools such as TAGME [76–78] or Dexter [79] can be utilized. However, based on our initial investigations both tools are still limited in the range of the languages that they support. TAGME supports English, Italian, German and Dexter works only with an English dump of Wikipedia.

Bibliography

[1] M. Levene, *An Introduction to Search Engines and Web Navigation*, 2nd ed. Wiley Publishing, 2010.

[2] F. Liu, J. Zhao, B. Lv, B. Xu, and H. Yu, "Product named entity recognition based on hierarchical hidden markov model," in *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*, 2005.

[3] Y. Yao and A. Sun, "Product name recognition and normalization in internet forums," *SIGIR Symposium on IR in Practice (SIGIR Industry Track)*, 2014.

[4] "mydatafactory co." [Online]. Available: https://www.mydatafactory.com/

[5] "Eriks co." [Online]. Available: https://www.eriks.co.uk/

[6] D. M. Bikel, S. Miller, R. Schwartz, and R. Weischedel, "Nymble: a high-performance learning name-finder," in *Proceedings of the fifth conference on Applied natural language processing*. Association for Computational Linguistics, 1997, pp. 194–201.

[7] "Information extraction course at stanford university." [Online]. Available: http://web.stanford.edu/~jurafsky/li15/lec6.induce.pptx,AccessedonOct.2017.

[8] B. Settles, "Biomedical named entity recognition using conditional random fields and rich feature sets," in *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications*. Association for Computational Linguistics, 2004, pp. 104–107.

[9] R. Bunescu, R. Ge, and R. J. Mooney, "Extracting gene and protein names from biomedical abstracts," *Unpublished Technical Note*, 2002. [Online]. Available: Availablefromhttp://www.cs.utexas.edu/users/ml/publication/ie.html

[10] K. Humphreys, G. Demetriou, and R. Gaizauskas, "Two applications of information extraction to biological science journal articles: Enzyme interactions and protein structures," in *Pac symp biocomput*, vol. 5, no. 505-516, 2000.

[11] S. Liljeqvist, “Named entity recognition for search queries in the music domain,” *KTH Thesis*, 2016. [Online]. Available: http://www.divaportal.se/smash/get/diva2:1010104/FULLTEXT01.pdf

[12] F. Luo, H. Xiao, and W. Chang, “Product named entity recognition using conditional random fields,” in *Business Intelligence and Financial Engineering (BIFE), 2011 Fourth International Conference on*. IEEE, 2011, pp. 86–89.

[13] F. Liu, J. Zhao, B. Lv, B. Xu, and H. Yu, “Product named entity recognition based on hierarchical hidden markov model,” in *Proceedings of the 4thSIGHAN Workshop on Chinese Language Processing*, 2005.

[14] D. P. Putthividhya and J. Hu, “Bootstrapped named entity recognition for product attribute extraction,” in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2011, pp. 1557–1567.

[15] E. Bick, “A named entity recognizer for danish.” in *LREC*, 2004.

[16] C. Niu, W. Li, J. Ding, and R. K. Srihari, “A bootstrapping approach to named entity classification using successive learners,” in *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*. Association for Computational Linguistics, 2003, pp. 335–342.

[17] S. Sekine, “Nyu: Description of the japanese ne system used for met-2,” in *Proc. of the Seventh Message Understanding Conference (MUC-7*, 1998.

[18] F. Dernoncourt, J. Y. Lee, and P. Szolovits, “Neuroner: an easy-to-use program for named-entity recognition based on neural networks,” *arXiv preprint arXiv:1705.05487*, 2017.

[19] R. Speck and A.-C. N. Ngomo, “Ensemble learning for named entity recognition,” in *International semantic web conference*. Springer, 2014, pp. 519–534.

[20] X. Carreras, L. Marquez, and L. Padró, “Named entity extraction using adaboost,” in *proceedings of the 6th conference on Natural language learning-Volume 20*. Association for Computational Linguistics, 2002, pp. 1–4.

[21] R. Sasano and S. Kurohashi, “Japanese named entity recognition using structural natural language processing,” in *Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-II*, 2008.

[22] D. M. Bikel, S. Miller, R. Schwartz, and R. Weischedel, “Nymble: a high-performance learning name-finder,” in *Proceedings of the fifth conference on*

Applied natural language processing. Association for Computational Linguistics, 1997, pp. 194–201.

[23] A. McCallum and W. Li, "Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons," in *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*. Association for Computational Linguistics, 2003, pp. 188–191.

[24] J. M. Pierre, "Mining knowledge from text collections using automatically generated metadata," in *PAKM*, vol. 2. Springer, 2002, pp. 537–548.

[25] F. Luo, H. Xiao, and W. Chang, "Product named entity recognition using conditional random fields," in *Business Intelligence and Financial Engineering (BIFE), 2011 Fourth International Conference on*. IEEE, 2011, pp. 86–89.

[26] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in *Proceedings of the eleventh annual conference on Computational learning theory*. ACM, 1998, pp. 92–100.

[27] M. Collins and Y. Singer, "Unsupervised models for named entity classification," in *1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999.

[28] D. Pierce and C. Cardie, "Limitations of co-training for natural language learning from large datasets," in *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, 2001.

[29] D. Yarowsky, "Unsupervised word sense disambiguation rivaling supervised methods," in *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 1995, pp. 189–196.

[30] P. K. Mallapragada, R. Jin, A. K. Jain, and Y. Liu, "Semiboost: Boosting for semi-supervised learning," *IEEE transactions on pattern analysis and machine intelligence*, vol. 31, no. 11, pp. 2000–2014, 2009.

[31] R. Sousa and J. Gama, "Comparison Between Co-training and Self-training for Single-target Regression in Data Streams using AMRules," in *IOTSTREAMING@PKDD/ECML*, 2017.

[32] W. W. Cohen and S. Sarawagi, "Exploiting dictionaries in named entity extraction: combining semi-markov extraction processes and data integration meth-

ods," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 89–98.

[33] J. Kazama and K. Torisawa, "Inducing gazetteers for named entity recognition by large-scale clustering of dependency relations," *Proceedings of ACL-08: HLT*, pp. 407–415, 2008.

[34] I. Guyon and A. Elisseeff, "An introduction to feature extraction," *Feature extraction*, pp. 1–25, 2006.

[35] H. Liu and H. Motoda, *Feature extraction, construction and selection: A data mining perspective*. Springer Science & Business Media, 1998, vol. 453.

[36] J. Tang, S. Alelyani, and H. Liu, "Feature selection for classification: A review," *Data Classification: Algorithms and Applications*, p. 37, 2014.

[37] M. A. Hall, "Feature selection for discrete and numeric class machine learning," in *Proc. 17th Int'l Conference. Machine Learning*, 2000, pp. 359–366.

[38] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[39] Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization," in *ICML*, vol. 97, 1997, pp. 412–420.

[40] M. A. Hall, "Correlation-based feature selection for discrete and numeric class machine learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*, ser. ICML '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 359–366. [Online]. Available: http://dl.acm.org/citation.cfm?id=645529.657793

[41] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000.

[42] M. Robnik-Šikonja and I. Kononenko, "Theoretical and empirical analysis of relieff and rrelieff," *Machine learning*, vol. 53, no. 1-2, pp. 23–69, 2003.

[43] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 27, no. 8, pp. 1226–1238, 2005.

[44] Q. Gu, Z. Li, and J. Han, "Generalized Fisher Score for Feature Selection," in *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, ser. UAI'11. Arlington, Virginia, United States: AUAI Press, 2011,

pp. 266–273. [Online]. Available: http://dl.acm.org/citation.cfm?id=3020548.3020580

[45] H. Liu and L. Yu, "Toward integrating feature selection algorithms for classification and clustering," *IEEE Transactions on knowledge and data engineering*, vol. 17, no. 4, pp. 491–502, 2005.

[46] S. Ma and J. Huang, "Penalized feature selection and classification in bioinformatics," *Briefings in bioinformatics*, vol. 9, no. 5, pp. 392–403, 2008.

[47] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[48] T. G. Dietterich *et al.*, "Ensemble methods in machine learning," in *Multiple Classifier Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 1–15.

[49] T. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization," *Machine learning*, vol. 40, no. 2, pp. 139–157, 2000.

[50] K. Moorthy and M. S. Mohamad, "Random forest for gene selection and microarray data classification," *Bioinformation*, vol. 7, no. 3, p. 142, 2011.

[51] G. Biau, L. Devroye, and G. Lugosi, "Consistency of random forests and other averaging classifiers," *Journal of Machine Learning Research*, vol. 9, no. Sep, pp. 2015–2033, 2008.

[52] P. Büchlmann and B. Yu, "Analyzing bagging," *Annals of Statistics*, pp. 927–961, 2002.

[53] R. Genuer, J.-M. Poggi, and C. Tuleau-Malot, "Variable selection using random forests," *Pattern Recognition Letters*, vol. 31, no. 14, pp. 2225–2236, 2010.

[54] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 1137–1143. [Online]. Available: http://dl.acm.org/citation.cfm?id=1643031.1643047

[55] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems*, 2015, pp. 2962–2970.

[56] B. Komer, J. Bergstra, and C. Eliasmith, "Eliasmith c. hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn," in *In: Proceedings of SciPy*, 2014.

[57] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-weka: Combined selection and hyperparameter optimization of classification algorithms," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013, pp. 847–855.

[58] E. F. T. K. Sang and J. Veenstra, "Representing text chunks," in *Proceedings of the Ninth Conference on European Chapter of the Association for Computational Linguistics*, ser. EACL '99. Stroudsburg, PA, USA: Association for Computational Linguistics, 1999, pp. 173–179. [Online]. Available: https://doi.org/10.3115/977035.977059

[59] D. Nadeau and S. Sekine, "A survey of named entity recognition and classification," *Lingvisticae Investigationes*, vol. 30, no. 1, pp. 3–26, 2007.

[60] S. Rüd, M. Ciaramita, J. Müller, and H. Schütze, "Piggyback: Using search engines for robust cross-domain named entity recognition," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics, 2011, pp. 965–975.

[61] K. Sparck Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, vol. 28, no. 1, pp. 11–21, 1972.

[62] H. P. Luhn, "A statistical approach to mechanized encoding and searching of literary information," *IBM Journal of research and development*, vol. 1, no. 4, pp. 309–317, 1957.

[63] N. Cristianini, J. Shawe-Taylor, and H. Lodhi, "Latent semantic kernels," *Journal of Intelligent Information Systems*, vol. 18, no. 2, pp. 127–152, 2002.

[64] N. Cristianini and J. Shawe-Taylor, *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[65] T. Hastie, S. Rosset, J. Zhu, and H. Zou, "Multi-class adaboost," *Statistics and its Interface*, vol. 2, no. 3, pp. 349–360, 2009.

[66] Y. Freund and R. E. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *European conference on computational learning theory*. Springer, 1995, pp. 23–37.

[67] K. Crammer and Y. Singer, "On the algorithmic implementation of multiclass kernel-based vector machines," *Journal of machine learning research*, vol. 2, no. Dec, pp. 265–292, 2001.

[68] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.

[69] I. Braga, L. P. Carmo, C. C. Benatti, and M. C. Monard, "A note on parameter selection for support vector machines," in *Proceedings of the 12th Mexican International Conference on Advances in Soft Computing and Its Applications - Volume 8266*, ser. MICAI 2013. New York, NY, USA: Springer-Verlag New York, Inc., 2013, pp. 233–244. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-45111-9_21

[70] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[71] L. Ratinov and D. Roth, "Design challenges and misconceptions in named entity recognition," in *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics, 2009, pp. 147–155.

[72] T. Horváth, R. G. Mantovani, and A. C. P. L. F. de Carvalho, "Effects of random sampling on svm hyper-parameter tuning," in *Intelligent Systems Design and Applications*, A. M. Madureira, A. Abraham, D. Gamboa, and P. Novais, Eds. Cham: Springer International Publishing, 2017, pp. 268–278.

[73] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[74] M. Hasan, A. Kotov, A. I. Carcone, M. Dong, S. Naar, and K. B. Hartlieb, "A study of the effectiveness of machine learning methods for classification of clinical interview fragments into a large number of categories," *Journal of Biomedical Informatics*, vol. 62, pp. 21 – 31, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S153204641630034X

[75] J. Tang, "Wikification: Entity annotation with wikipedia." [Online]. Available: https://cs224d.stanford.edu/reports/Tang.pdf

[76] “TagMe Tool.” [Online]. Available: https://tagme.d4science.org/tagme/

[77] “TagMe Documentations.” [Online]. Available: https://services.d4science.org/web/tagme/documentation

[78] “TagMe Tool.” [Online]. Available: http://acube.di.unipi.it/tagme

[79] D. Ceccarelli, C. Lucchese, S. Orlando, R. Perego, and S. Trani, “Dexter: an open source framework for entity linking,” in *ESAIR’13, Proceedings of the Sixth International Workshop on Exploiting Semantic Annotations in Information Retrieval, co-located with CIKM 2013, San Francisco, CA, USA, October 28, 2013*, 2013, pp. 17–20. [Online]. Available: http://doi.acm.org/10.1145/2513204.2513212