Linköping University | Department of Computer and Information Science
Master thesis, 30 ECTS | Datateknik
2017 | LIU-IDA/LITH-EX-A--17/026--SE

Automating Text Categorization with Machine Learning:
Error responsibility routing in a multi-layer hierarchy

Ludvig Helén[1] and **Alexander Persson**[2]

[1]Department of Computer and Information Science, Linköping University
[2]Department of Computer Science and Engineering, Chalmers University of Technology

Supervisor : Jonas Wallgren
Examiner : Ola Leifler

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Upphovsrätt

Copyright

© **Ludvig Helén**[1] and **Alexander Persson**[2]

[1]Department of Computer and Information Science, Linköping University
[2]Department of Computer Science and Engineering, Chalmers University of Technology

Abstract

The company Ericsson is taking steps towards embracing automating techniques and applying them to their product development cycle. Ericsson wants to apply machine learning techniques to automate the evaluation of a text categorization problem of error reports, or *trouble reports* (TRs). An excess of 100,000 TRs are handled annually.

This thesis presents two possible solutions for solving the routing problems where one technique uses traditional classifiers (Multinomial Naive Bayes and Support Vector Machines) for deciding the route through the company hierarchy where a specific TR belongs. The other solution utilizes a *Convolutional Neural Network* for translating the TRs into low-dimensional word vectors, or *word embeddings*, in order to be able to classify what group within the company should be responsible for the handling of the TR. The traditional classifiers achieve up to 83% accuracy and the Convolutional Neural Network achieve up to 71% accuracy in the task of predicting the correct class for a specific TR.

Acknowledgments

The work conducted in this thesis was made possible thanks to Ericsson AB. We are grateful for the supervision and help given by company staff Michael West and Henric Stenhoff, as well as the cheerful bunch in team CATalytics with whom we have shared office space during this time.

We would like to express our sincerest gratitude to Chalmers examiner Richard Johansson and Linköping University examiner Ola Leifler.

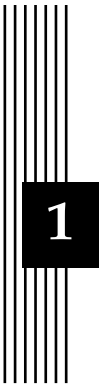
We thank Chalmers academic supervisor Selpi and Linköping University supervisor Jonas Wallgren for their involvement and great feedback throughout the project. We also thank Chalmers master thesis coordinator Birgit Grohe for her input and time given to enable a joint thesis work between Chalmers and Linköping University.

Lastly, we extend special thanks to our peers Ammar Shadiq, Daniel Artchounin, Joacim Linder and Ellinor Rånge for their company and thoughtful oppositions during the thesis work.

Contents

List of Figures

List of Tables



1 Introduction

1.1 Motivation

Machine learning is a popular topic in the field of computer science which enables the development of models capable of making intelligent predictive decisions for unseen scenarios. As data gathering has surged over the last few years together with available information, machine learning techniques enable the training of more accurate machine learning models. It is one of the reasons why there has been a rising trend in popularity. Machine Learning techniques are applied to solve a range of tasks. A few examples are: face detection, weather forecasting, classifying genetics, anti-spam detection or translating text from one language to another.

The company Ericsson is now taking steps towards embracing machine learning techniques and applying them to their product development cycle. More specifically, they would like to use machine learning techniques to automate the evaluation of *trouble reports* (TRs). More than 100,000 TRs are handled annually and stored in a database.

A TR consists of a free text description of an issue or bug together with system log data regarding intellectual property, for example base stations, radio and antennas, of Ericsson. The current method of evaluating and assigning TRs is an arduous, bureaucratic and costly process where experts manually analyze the reports to identify the product team responsible for correcting the issue, resulting in long lead times between meetings at great cost for the company.

The challenge is to accurately predict which company product team a TR belongs to, based on predefined fields and error descriptions. In addition to merely making this classification, the implementation should preferably derive the route through the company hierarchy which the TR would normally take.

In recent years, Convolutional Neural Networks (CNN), a technique mainly used for image recognition has found advances in sentiment analysis and text classification, even with a relatively straight forward architecture [1, 2]. This technique points to that it can be suitable for understanding the content of a TR, as well as performing a classification to the team who should solve it.

The task of assigning TRs to the right company product team can be formulated as a text categorization problem. The nature of the data suggests that a preprocessing technique can be

applied. In this case, a preprocessing technique means refining the vocabulary encountered across trouble reports by removing words deemed superfluous.

1.2 Aim

The work targets the same domain as the prototype developed by Jonsson et al. [3], with a focus area of applying a Convolutional Neural Network and more traditional classifiers to the free text in order to predict the routing of TRs. If manual handling can be replaced with a confident automated error handling system this would lead to shorter handling time and thus cost. By investigating and applying different supervised text categorization and preprocessing techniques, a variety of models with respective prediction accuracies can be evaluated.

1.3 Research Questions

In order to automatically predict the route of TRs based on the description confidently, the following research questions are posed:

1. How can a preprocessing technique when handling trouble reports improve the prediction accuracy? What are its characteristics?

   The question refers to reasoning about different preprocessing techniques and how they affect the performance of a predictive model.

2. How can the prediction accuracy of assigning responsibility in an automated system be improved by using a hierarchical approach or Convolutional Neural Networks over traditional classifiers in a flat approach?

   The question presented delimits to using a Convolutional Neural Network model with frugal tuning of the hyperparameters. This is done in order to decide the feasibility of using a Convolutional Neural Network for classifying TRs.

1.4 Delimitations

The work will not include the design of new algorithms, but rather experiments with the utilization of existing ones. The TR dataset provided will be composed of free text descriptions belonging to TRs.

2 Theory

2.1 Machine Learning

Machine learning refers to the art of developing models with predictive behavior, which are trained by observing patterns in some data source. A model is trained when a learning algorithm is used on a set of data (*training set*) in order to make predictions on new data [4]. Machine learning is a topic involved and used in different areas of the modern society, where models are constantly predicting choices to make, be it the targeted ads in social media applications or an e-mail client determining whether an incoming email should be classified as spam or not. Since a massive amount of data is being collected today, a desire for automating complex tasks arises. Giving machines the responsibility of finding subtle patterns in data and solving problems faster than a human ever could, even without interference, gives machine learning the potential of providing solutions to problems which have earlier been deemed too challenging.

2.1.1 Supervised Learning

By taking a certain input, called training data, a model can be trained and then be used to classify new example data [4]. In order for the model to be able to distinguish different inputs, the input is divided into *features*. Features can take various shapes, they can for example be binary or categorical values. The input samples are marked with class labels in order to distinguish them and the model is then said to be trained with *supervised learning*. This is applicable to several different types of problems in combination with the use of various techniques [5]. The approaches used for this work are explained further in Section 2.1.2.

2.1.2 Classification

Attempting to automatically classify textual content has been around since Maron [6] implemented a technique for automatic classification of documents. Today there exists a wide range of areas where classification is useful and an accurate prediction gives an idea of the relationship and interconnectivity between objects.

In order to compute what class a certain input belongs to, a need to differentiate between two or more classes exists. A computation is performed that allows a *classifier* to make a

prediction. A general depiction of a classifier dividing a population of data into two classes in a 2D-plane is shown in Figure 2.1.

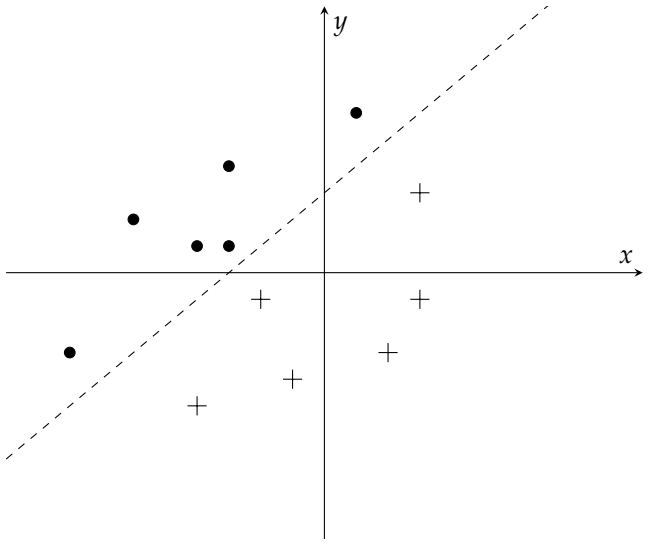


Figure 2.1: A classifier (dashed line) dividing a population of data into two classes (filled dots and plus signs).

Classifiers are a core part of machine learning systems. Classifiers can be applied to various problem domains and can classify different types of data, ranging from binary values to documents with words. They analyze input samples and compare them by their features. Feature values are stored in vectors called *feature vectors*, one set of features per vector. It works as an agglomerating tool to decide where a certain input most likely belongs to. There exist different types of established models in order to represent text as feature vectors. One model is the *Bernoulli document model* which represents documents with binary values based on the presence of a word (for example an index of the vector is set to 1 if a word is present and 0 if it is not present). Another model is the *Multinomial document model* which represents documents as integer elements which are derived from the frequency of word's occurrence within a document [4]. However, the work by McCallum and Nigam [7] has shown that the Multinomial model in many cases outperforms the Bernoulli model when both models assume a Naive Bayes assumption (Section 2.1.5).

2.1.3 Multi-class Classification

The classification task to assign input to one of two categories, one positive and one negative, is known as binary classification. For example, to predict whether an email is spam or if it is not spam. Some classifiers can only handle this configuration, so to accommodate additional elements in the category space some modifications must be made. For example, Support Vector Machine classifiers (explained in Section 2.1.4) can use an approach called "one-vs-all", where a classifier for each category is trained to separate the elements of that class from the ones in every other class. When new input is run through these classifiers, the category for the corresponding classifier that outputted the highest score is chosen [4].

2.1.4 Support Vector Machines

Support Vector Machines (SVM) are classifiers able to predict the category of previously unseen input. They have been trained by supervised learning algorithms and labeled training data. SVMs calculate a hyperplane that separates points in space extracted from input data,

which allows the model to classify new instances depending on what side of the plane they belong to [8]. The classifier can be expressed mathematically as

$$y_i = \text{sgn}(\mathbf{w}^T \mathbf{x}^{(i)} + b) \tag{2.1}$$

where $\mathbf{w}^T \mathbf{x}^{(i)} + b$ describes the hyperplane that is composed of its normal vector $\mathbf{w}$, the given input data $\mathbf{x}^{(i)}$ and bias parameter b.

As seen in Figure 2.2, a set of data points from each class that are closest to the decision boundary are known as the *support vectors* and the hyperplane is positioned between them in a way so that the distance to the support vectors are maximized. The intuition is that this makes the classifier good at generalization and will be more capable of correctly classifying new data points [8].

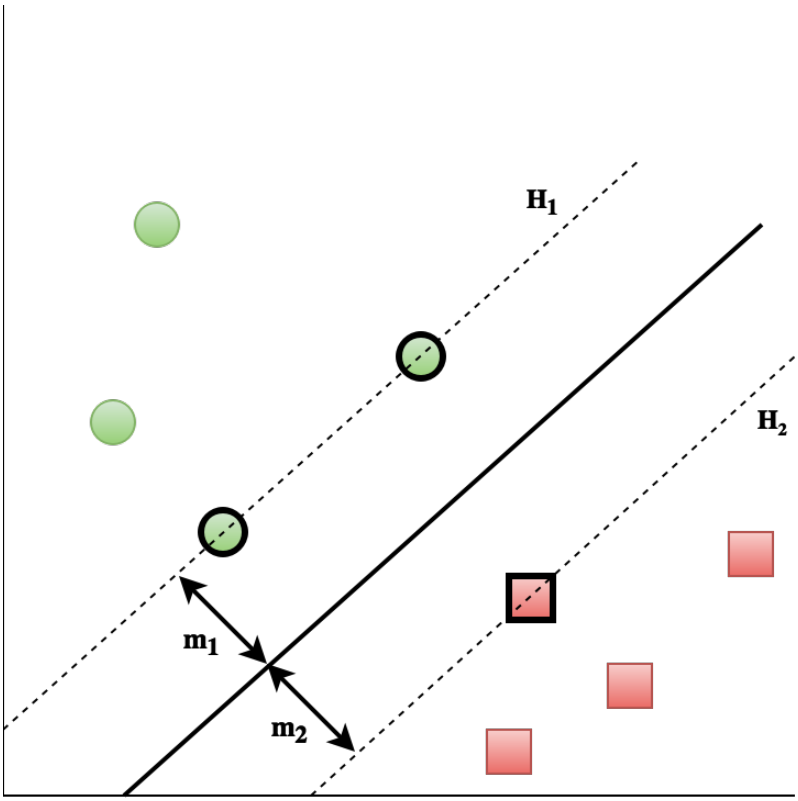


Figure 2.2: An SVM model (bold line) that separates data points from two classes (green circles and red squares). The hyperplanes H_1 and H_2 go through the support vectors (data points highlighted with bold borders) from each class and the decision boundary is placed in-between so that margins m_1 and m_2 are maximized.

SVM classifiers can be linear, but modifications can be made in order to produce non-linear classifiers with the help of kernel functions. With kernel functions, data points can be projected into a higher dimensional feature space, allowing them to become linearly separable. It should be noted that this mapping is not computed explicitly. The projection generally increases the error of generalization of the classifier but it can be mitigated by providing the algorithm with more training data [9].

2.1.5 Naive Bayes Classification

Naive Bayes classifiers belong to the group of probabilistic classifiers which produce a probability distribution over classes, rather than simply outputting which class a document belongs to [4]. A naive Bayes classifier is based on Bayes theorem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \tag{2.2}$$

which gives the conditional probability of some event A given an observation B. By (naively) assuming that the features are conditionally independent, a joint probability model is obtained and the notation can be customized to fit into a text categorization context

$$P(c|d) \propto P(c) \prod_{1 \leqslant k \leqslant n_d} P(w_k|c), \tag{2.3}$$

where the probability of some class c being the correct classification for a document d is defined by the product of the prior probability of class c and the multiplied conditional probabilities of every word w_k (from the n distinct words in d) given c. It is regarded as a simplistic approach that usually performs worse than state-of-the-art ones, but has proven to be competitive because of its simplicity [10]. It was appointed as one of the top data mining algorithms in 2008 [10] and is often included in experiments as a baseline [11, 12, 5]. There are several variants of the Naive Bayes classifier, each making different assumptions about the feature distribution. Some of these variants are more suited for text classification tasks than others. For example, utilizing a Multinomial Naive Bayes classifier is useful when there is meaningful information gained from knowing that some terms are present in numerous places in a document [13]. To classify a document, the class c^* from the set of classes C that was predicted with the highest probability is chosen.

$$c^* = \arg\max_{c \in C} \left(P(c|d)\right) = \arg\max_{c \in C} \left(P(c) \prod_{1 \leqslant k \leqslant n_d} P(w_k|c) \right). \tag{2.4}$$

For Multinomial Naive Bayes, the conditional probability $P(w|c)$ that a word belongs to a certain class is assumed to be

$$P(w|c) = \frac{T_{cw}}{\sum_{w' \in V} T_{cw'}}, \tag{2.5}$$

where T_{cw} is the amount of times a word w occurs in documents corresponding to class c. Dividing this value with the same computation for all other words $T_{cw'}$ yields the relative frequency of word w which is used as the estimate for the probability.

2.1.6 Hierarchical Classification

For many text categorization tasks it is sufficient to consider the classification in a flat context. However, there are situations where it is desired to apply classification models on hierarchical architectures because of the sheer amount of classes. By engineering a hierarchical structure based on the contents of the documents and the relationships between them, such as in Figure 2.3, it is possible to achieve results that better match reality [14, 15].

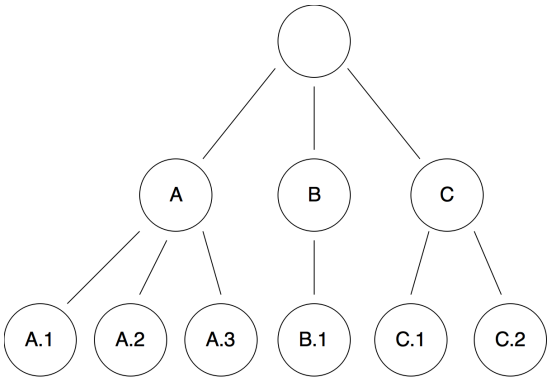


Figure 2.3: A hierarchical class space featuring classes A, B, C and their descendants. The root node is inserted to connect the graph.

There are a few traditional approaches for hierarchical classification; one can train a classifier on each parent node and use them to route an input example from the root node down to a leaf node, you can train a classifier on all nodes, or you can consider the hierarchy as a whole (known as "big-bang") and obtain a model that incorporates decisions from all classifiers [16]. Classification can be done on either leaf-only nodes (mandatory leaf-node prediction) or parent nodes may be included as well (non-mandatory leaf-node prediction), depending on the architecture of the classifiers. Hierarchical classification has been applied in several domains, such as text categorization [14] and bioinformatics [17].

2.2 Convolutional Neural Networks

The following section is divided into several parts in order to give a coherent overview of what Convolutional Neural Networks are and how they can be used as a tool for classification. The parts cover:

1. An introduction and description of how Convolutional Neural Networks work,
2. The domains and work where the networks are used,
3. A detailed description of the various core parts that together form the networks.

Neural networks are self-tuned classifiers, meaning they adjust themselves after time according to training data. The networks are inspired by biological experiments in the visual cortex and each building block in a neural network are often referred to as *neurons*. These neurons are highly interconnected, forming a complex network of signals [18]. An example of a fully connected neural network can be depicted in Figure 2.4. *Convolutional Neural Networks* (CNN) are characterized by only being selectively connected between various types of layers, unlike traditional networks. Only at the last layer the neurons are fully connected and that is where the actual classification takes place. CNNs are suited for image-like data representations [19]. When a CNN is sliding a window of focus, called a filter, over the input, the network can learn to identify patterns in the different parts of the image. This process is known as the model performing *convolutions*.

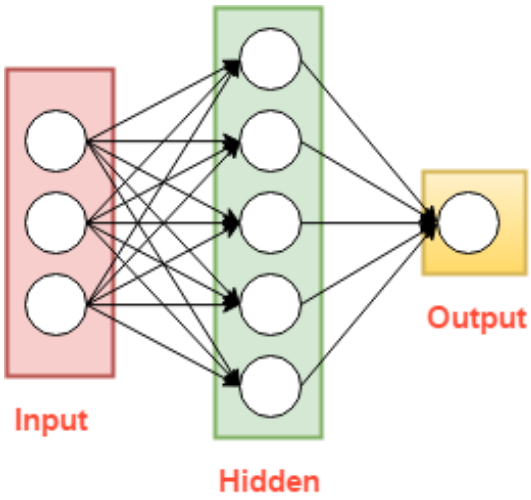


Figure 2.4: A fully connected Neural Network with one input layer, one hidden (takes input from input layer and proceeds to send to output layer) layer and one output layer. Each neuron in the network is connected to every neuron in the previous layer.

LeCun et al. [19] describe an implementation in their work of the CNN called *LeNet*, which captures the general structure of how various parts together form a functional neural network for recognizing images. It is mentioned how a CNN consists mainly of three ideas in order to find out differing factors for an input:

1. **The use of local receptive fields** - Using local fields or sub-parts enables the capturing of important deviations for a domain, e.g. edges or endpoints in an image. Combining these layers will help the consecutive layers to find important features which possibly can be useful to the whole domain. These are identified with the help of a filter.

2. **Sharing weights for filters** - When local fields for a domain are captured they are stored in a respective plane called a *feature map*. All units in the feature map share the same set of weights since the units of a specific feature map need to be consistent in the variances it detects for a domain (it looks for the same feature but in different positions). A convolutional layer is composed of several feature maps, which all can have varying weights. This enables multiple features to be detected within the examined domain. This process can be depicted further in Figure 2.5.

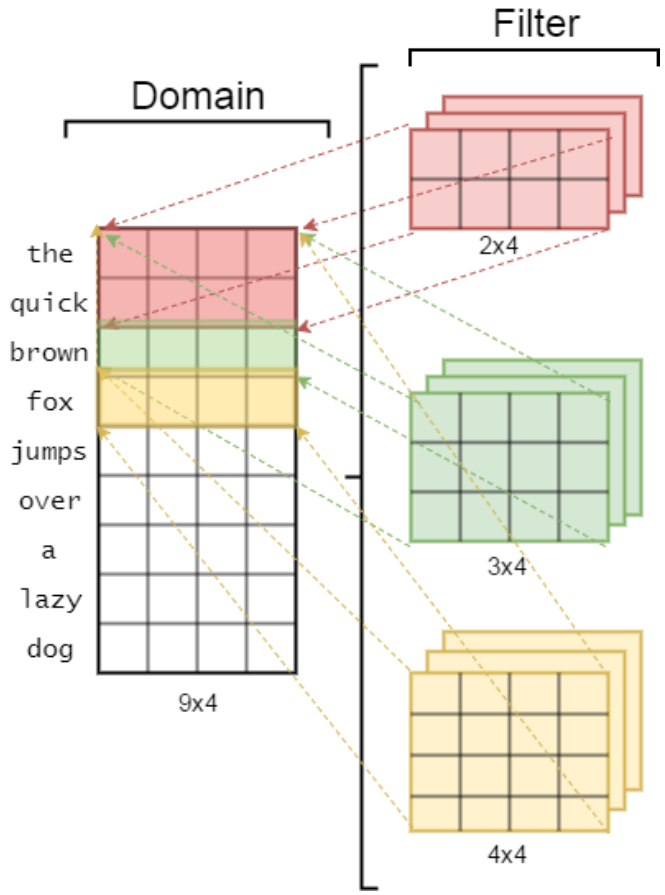


Figure 2.5: Three varying filters performing convolutions over a domain to find deviations in order to create feature maps.

3. **Collecting subsamples** - If a feature is found, its relation to other features is the important factor to consider rather than its position. It is desired to identify points of interest i.e. the whereabouts of endpoints and/or edges of a domain. Focusing on the exact position is dangerous in the sense that inputs, be it training sets or evaluation data, has a chance of varying from each other. A common method when constructing CNNs is to shrink the spatial resolution of the feature map. This is performed by collecting subsamples, also known as *pooling*. This refers to the procedure where a local averaging of the feature maps is performed. This reduces the resolution of the feature map while increasing its resilience towards interferences such as shifts and prevents the network from overfitting. In addition to this, it also reduces computational cost. This is illustrated in Section 2.2.3 and Figure 2.8. When all of these steps are combined, a full CNN can be established. A high level system of a CNN performing classifications on images is shown in Figure 2.6.

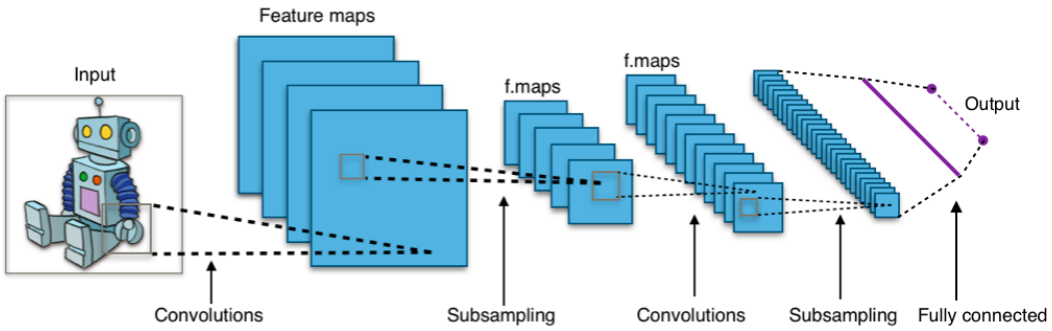


Figure 2.6: A Convolutional Neural Network performing the steps of classifying an image of a toy robot.[1]

2.2.1 Hyperparameters

There exist an abstracted set of parameters which can be set to decide different specifications for a model called *hyperparameters*. They can be seen as settings which tune the behavior of a model. Hyperparameters can be set both by humans or with the help of algorithms. Examples of hyperparameters can be the learning rate (how fast the network adapts to training data) of a model or the number of training iterations that will be performed [20]. Details about the hyperparameters used in the CNN architecture for this work are further described in Section 3.3.2 and detailed examples of hyperparameter values can be found in Table 3.4.

Work done by Bengio [20] suggests guidelines for how to set hyperparameters when training neural networks and deep architectures. Bengio mentions that picking well-selected hyperparameters is equally important as selecting a model.

2.2.2 Activation Functions

In a paper written by LeCun et al. [21] it is described how feedforward neural networks (information moving in one direction, from input to output) map a fixed-size input to fixed-size output. This is done when a computed sum from the inputs from the previous layer is sent to an *activation function*. There exist several ways to do this and one which was common in the past has been the smoothing non-linear functions, or activation functions, such as

$$f(z) = \tanh(z). \tag{2.6}$$

The tanh function takes a z, where $z \in \mathbb{R}$ and *squashes* (sets it to a finite interval) it into the range of $[-1, 1]$. This can be rewritten as

$$\tanh(z) = 2\sigma(z) - 1\,. \tag{2.7}$$

This is related to the *logistic sigmoid* function

$$\sigma(z) = \frac{1}{1 + \exp(-z)}. \tag{2.8}$$

The logistic sigmoid function is a squashing function which takes a real-valued input and squashes it to a range between $[0, 1]$ as shown in Figure 2.7.

[1] *Typical CNN* by Aphex34. Licensed under CC BY-SA 4.0

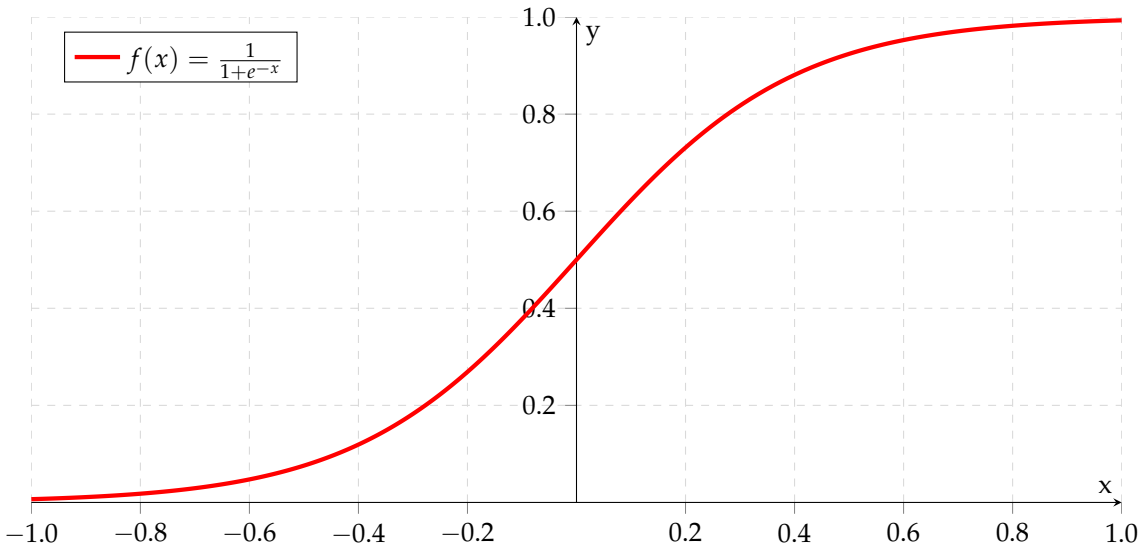


Figure 2.7: Graph of the sigmoid function ranging between a finite interval of [0,1].

For the case when $K > 2$ classes, new value x, class C_k and class C_j where j and k may or may not be equal. Class-conditional densities $p(x|C_k)$ and class priors $p(C_k)$ exists:

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{\sum_j p(x|C_j)p(C_j)} = \frac{\exp(z_k)}{\sum_j \exp(z_j)}. \tag{2.9}$$

This is know as the *normalized exponential* or the *softmax function* and is a smoothed version of the *max* function, since if $a_k \gg a_j$ for all $j \neq k$, then $p(C_k|x) \simeq 1$ and $p(C_j|x) \simeq 0$. Softmax is a part of logistic regression (estimating a binary response for a variable based on one or more features) where it is desired to handle several classes. Logistic regression is used for binary classification; $x \in \{0, 1\}$, whereas the softmax function instead allows the classifier to squash a K-dimensional vector of arbitrary real-valued scores to a vector between ranges $[0, 1]$ that sums up to 1 [4].

The non-linear function called *Rectified Linear Unit* (ReLU) refers to a half-wave rectifier:

$$f(z) = \max(z, 0). \tag{2.10}$$

This means setting all negative values of an output to 0. The help of ReLU in a deep supervised network typically allows for faster training without the need of unsupervised pretraining [22].

2.2.3 Max Pooling

When a network has convoluted with its filter over a region, it is desired to reduce the size of the filters, but still, keep points of interest in the convoluted filters. This is where *max pooling* becomes suitable. Not only does this reduces the number of parameters in the network, it also reduces the spatial size of the network. This is effective towards overfitting since some values are discarded which decreases the model's chances to tailor itself to the training data [19]. Figure 2.8 depicts max pooling applied to a 4×4-matrix.

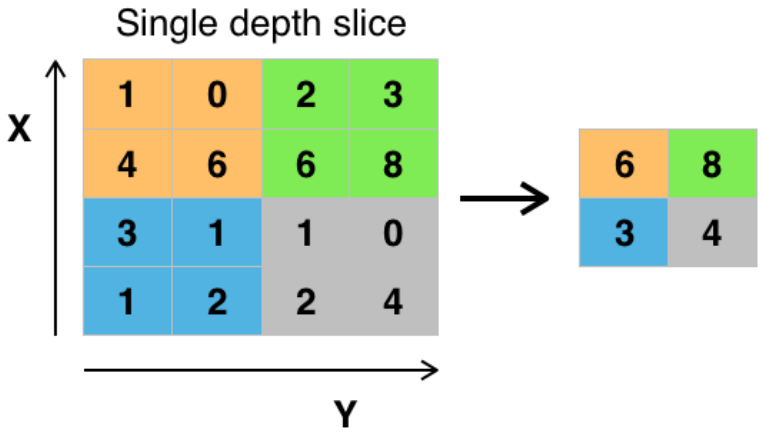


Figure 2.8: Max pooling of a single depth slice within a Convolutional Neural Network.[2]

There exist several variations of pooling techniques such as k-pooling, max pooling, average pooling and L_2-norm pooling. Max pooling has shown good results in other work [23] and can be described by

$$z_j = \max\{|u_{1j}|, |u_{2j}|, ..., |u_{Mj}|\}, \tag{2.11}$$

where z_j is the j-th element of the pooled feature map $\mathbf{z}$. Element u_{ij} is the i-th row and j-th column of a feature map $\mathbf{U}$, and $\mathbf{M}$ is the size of the region being pooled.

2.2.4 Dropout

Srivastava et al. [24] discuss an approach to prevent neural networks from *overfitting*, which refers to the event where a classifier becomes too tailored to its training data. When sparse data are present, relationships between output and input can consist of noise in the sampling which leads to undesirable connections which cause overfitting. Strivastava et al. [24] discuss the complex task of training several, varying networks which require an individual tuning of hyperparameters. This requires not only experimentation and a comprehensive bulk of training data but also a lot of computational power which is needed in a time critical system. Srivastava et al. propose a technique called *dropout* in order to tackle the issues. Figure 2.9 depicts the difference between a regular network and a network applying dropout. The general idea is by erasing or *dropping* units randomly in a neural network, its incoming and outgoing connections can be removed from the architecture. This leads to a pruned network which mitigates the risk of overfitting. A parameter is introduced for assigning the probability of a certain unit being dropped, where Srivastava et al. [24] and Zhang et al. [25] both find that 0.5 (50 %) is a value close to optimal in several networks. The work proceeds to present results showing that an implementation of CIFAR-100[3], which is an established dataset in the machine learning community consisting of images divided into subclasses, outperforms other similar configurations. The works by Goodfellow et al. [26] and Zeiler et al. [27] show that Convolutional Neural Networks with max pooling (described in Section 2.2.3) and dropout perform better than recent similar work in image recognition.

[2] *Max pooling* by Aphex34. Licensed under CC BY-SA 4.0
[3] https://www.cs.toronto.edu/ kriz/cifar.html/

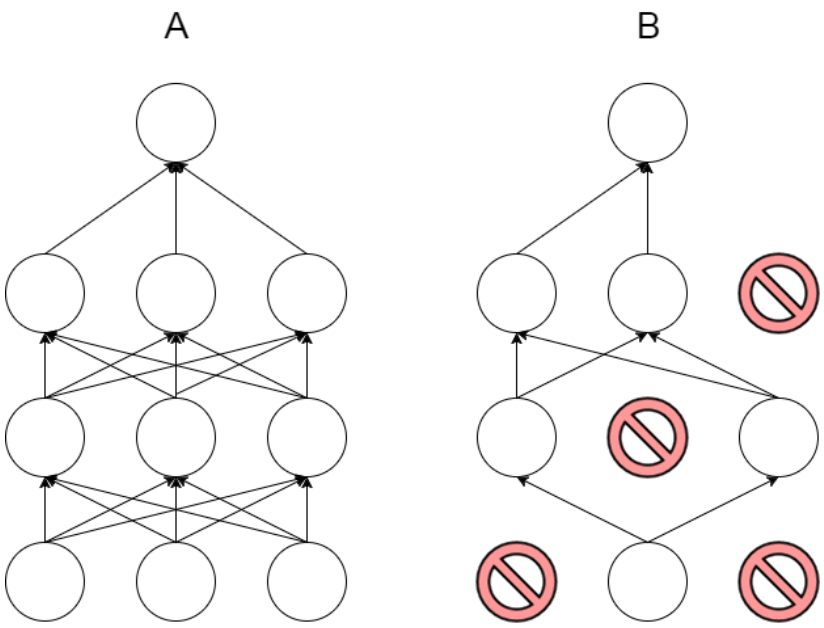


Figure 2.9: **A**: A regular fully connected network. **B**: A fully connected network applying dropout where random units have been dropped (Red mark).

However, even though applying dropout significantly increases the error rate in some domains, Srivastava et al. find that the beneficial differences dropout brings to the area of text categorizations are less than that of the image one. A conducted experiment shows that the improvement in the error rate was barely 1.5%. One of the major drawbacks of using an architecture together with dropout is that each computation trains a different architecture (since stochastic units are dropped at each iteration, the network will likely have an unique set of active neurons each iteration), which increases the total time of training a model. This gives rise to a striking trade-off between overfitting a model and its corresponding training time.

2.2.5 Error Backpropagation

In order to minimize the error of a feedforward neural network the gradient of the weights in a network needs to be evaluated. This can be done with the help of *Backpropagation*, which means that the error is evaluated between each iteration in a training procedure. After the error is evaluated, it is a question of optimization.

Many algorithms train iteratively i.e. they run in a number of steps in order to, for example, minimize the error function. After the evaluation is done, an adjustment technique is preferred in order to improve the result continuously [4]. One such technique is *Gradient Descent*, first developed by Rumelhart et al. [28], where the evaluations are used to calculate adjustments needed to be done to the weights in order to optimize the result (minimizing the loss of the error function). One of the greatest benefits of using backpropagation is the computational efficiency due to its scaling capabilities. This is related to the weighted sum in a feedforward network. The formula calculating the weighted sum is described by

$$a_j = \sum_i w_{ji} z_i, \tag{2.12}$$

where z_i is the input which sends a connection forward to unit j, and w_{ji} is the weight for the specific connection. A single evaluation needs $\mathcal{O}(W)$ for a large W, where W is the total num-

ber of weights in a network. A network often has a greater number of weights, transcending the number of units and due to the sum computation (2.12), each term will need exactly one multiplication and one addition, leading to efficient $\mathcal{O}(W)$ computations [4].

The procedure of backpropagation can be summarized as the following: Techniques such as Gradient Descent are used in order to optimize the error function. In order to optimize the error function gradients of the network are required, which are computed with the help of backpropagation.

2.3 Natural Language Processing

The following section is divided into subsections related to preprocessing of documents and how natural language can be processed and worked with.

2.3.1 Tokenization

When working with text, the sequence of characters is split in such a way that each word or term is represented as a so called *token* (hence the term *tokenization*) [13]. A naive way of obtaining these tokens would be to split the document contents by every whitespace. Although this would be acceptable in certain cases, it fails in capturing more intricate word relationships. For instance, the consecutive words *North* and *Carolina* should most likely be represented by the token *North Carolina* rather than a separate token for each word.

2.3.2 Stemming

Stemming is the process of identifying varieties in inflexions of words and exchanging them with their word stem. Reducing words to this base form can be useful, making the task of text classification easier. There may not be need to keep counts of both the words *run* and *running* for example, so all occurrences of *running* can be stemmed to its base form *run* [29]. This also comes with the benefit of shrinking the feature space.

2.3.3 TF-IDF

The process of finding and returning relevant information from a source of data is known as information retrieval [30]. In the field of text categorization, a technique commonly used to help map documents to categories is *TF-IDF* (Term Frequency - Inverse Document Frequency) [31]. This is a compound technique composed of two parts:

Term frequency refers to the number of times a term (word) occurs in a given text. *Inverse document frequency* reveals how much information a word really provides in relation to the whole collection of documents. Abundant words such as "the" and "and", which are likely to occur in a typical document, contribute little to the purpose of classification since they will likely occur many times across all documents, while more specific words related to the topic of the text are vastly more meaningful [31].

By multiplying both quantities TF-IDF is acquired, a measurement that can be used as weighting in a classification context. The precise definition varies depending on how to represent term frequency and inverse document frequency, but a commonly used equation for computing it is

Since a prediction system is reliant on how the classifier operates, a lot of work goes into detail of how a classifier is designed. With regard to classification of text, work done by Mariam Thomas et al. [32] mentions that data preprocessing when working with text are of importance.

$$\text{TF-IDF}_{w,d} = f_{w,d} \cdot \text{IDF}_{w,D} = |\{w \in d\}| \cdot \log \frac{N}{1 + \text{DF}_w}, \tag{2.13}$$

where $f_{w,d}$ equals the number of times a word w appears in a document d belonging to collection of documents D. The weighting for a word will be equal to the term frequency times the log of the inverse document frequency of a collection of N documents [33].

2.3.4 Skip-gram

Representing words distributed in a vector space can help algorithms to increase their performance by groupings words which share similarity. The Skip-gram model was introduced by Mikolov et al. [34] and predicts a context based on the current target word. Each word which is processed with the model is the input of a log-linear classifier with a continuous projection layer which can predict words within a specific range before and after the current word. Mikolov et al. proceed to mention that more distant words from a targeted word tend to be less related than its closer neighbors.

An example of the Skip-gram model can be the prediction of the target context:
It is nice to and *you*, where the missing target word predicted to be *meet*.

2.3.5 Continuous Bag of Words

The Continuous Bag of Words (CBOW) model can be seen as the inverse of the Skip-gram model. In the sense that instead of predicting a specific word from a target context within a specific range, it instead targets a word and tries to predict the missing context.

An example of the CBOW model can be the prediction of the target context given a specific word. An example would be the given word *meet*, where the target context predicted to be *It is nice to* and *meet you*.

Mikolov et al. [34] has uploaded their work called Word2vec as a project [4], where both the implementations of Skip-gram and CBOW are available. It is mentioned that the hyperparameters set for training the model differ to the nature of the problem to be solved. The following pointers are mentioned:

- Skip-gram are slower, but works better for infrequent words since it handles context-target pairs as new observations, meaning that words that occur more seldom still gets treated accordingly by the Skip-gram model. However, CBOW is faster. Hierarchical Softmax 2.3.6 is better for infrequent words, and negative sampling are better for frequent words.
- Sub-sampling of frequent words can be beneficial in terms of accuracy and speed when working with larger datasets and should be set in a range of $1e-5$ to $1e-3$. The parameter refers to setting a threshold value for removing words of a certain frequency.
- The dimensionality of the word vectors goes by the rule that more is better, but not always. The complexity increases with the number of dimensions, and as presented in their work [34] the gain in accuracy from 300 to 500 dimensions is not as much, compared to 100 to 300.
- The sentence context window size (how large context of a sentence that is under observation) for the two models is recommended to be around 10 for Skip-gram and 5 for CBOW, but for best results should be tuned accordingly after the issue at hand.

2.3.6 Hierarchical Softmax

When computing the Softmax function, one efficient way of computing it is using the Hierarchical Softmax method [35]. This refers to the process where a binary tree represent the words of a vocabulary. All words in the vocabulary are represented as leaf units of the tree and each

[4] https://code.google.com/archive/p/word2vec/

leaf unit has a unique path from to the root unit. This path is used in the probability calculation of the words represented by a leaf unit. Mikolov et al. [36] mention that one of the main advantages of using the Hierarchical Softmax is that the complexity per training instance per context word is reduced from $\mathcal{O}(V)$ to $\mathcal{O}(\log V)$. The speedup is beneficial especially when working with larger datasets, where the computation complexity of words increases. The work performed by Morin and Bengio [37] indicates this speedup for a network at the cost of having a slightly worse generalization performance.

2.4 Evaluation of Results

In order to trust the conclusions of a classifier, one must have some quantitative assurance that its predictions are accurate more often than not. Failing to properly evaluate the classifier will lead to a flawed representation of its prediction capabilities. Instead of relying on human intuition, there are standard methods and metrics of evaluation used in machine learning that give more realistic figures of what classification performance to expect in response to general input, which are described below.

2.4.1 Cross-validation

Ten-fold cross-validation is a highly prominent method of evaluation [38]. It means the available data is split up in ten, roughly equal-sized partitions (folds), and the classifier is then trained on nine of these followed by the classification of the remaining data. The fold used for classification is then shifted and the process is repeated, for a total of ten times. By taking the average of the results, a more reliable statistical representation of the classifier's capabilities can be achieved.

2.4.2 Precision, Recall, F_1-score and Accuracy

When classification of a certain input has been performed, the results can be evaluated by reasoning about the distribution of positive and negative examples (positive if belonging to the category in question and vice versa) [39]. The fraction of correctly classified examples from all examples that were classified to one class is known as *Precision*.

$$\text{Precision} = \frac{|\{\text{True Positives}\}|}{|\{\text{True Positives}\}| + |\{\text{False Positives}\}|}. \tag{2.14}$$

The precision metric is usually paired with the metric *Recall*, which is the fraction of correctly classified examples to one class from all the examples of that category,

$$\text{Recall} = \frac{|\{\text{True Positives}\}|}{|\{\text{True Positives}\}| + |\{\text{False Negatives}\}|}. \tag{2.15}$$

In essence, precision hints at the quality of what was actually classified to a specific class, while recall shows how many of the examples in that category were correctly classified to a specific class. The two metrics share a harmonic mean which is known as the F_1*-score*.

$$F_1\text{-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{2.16}$$

By dividing the number of examples the classifier got right by the total number of examples, you get *Accuracy*.

$$\text{Accuracy} = \frac{|\{\text{True Positives}\}| + |\{\text{True Negatives}\}|}{\begin{array}{r}|\{\text{True Positives}\}| + |\{\text{True Negatives}\}| + \\ |\{\text{False Positives}\}| + |\{\text{False Negatives}\}|\end{array}}. \tag{2.17}$$

This metric is not so useful in isolation, however, since a classifier that predicts the same category every time on a dataset with examples of mostly that category will score a high accuracy. The results can instead be presented in a *confusion matrix* which collects the results and presents them. A presentation of an example classification can be seen in Table 2.1. All categories are mapped to the corresponding prediction: True Positives (TP), False Negatives (FN), False Positives (FP) and True Negatives (TN) of the specific category. Values are mapped to the cells which represent the distribution of predictions. This gives a good overview of the classifier's performance in relation to the number of test examples from each category.

Table 2.1: Example of a classification of three fruits with corresponding metrics.

Category	**TP**	**FN**	**FP**	**TN**	**Precision**	**Recall**	F_1**-Score**	**Accuracy**
Apple	3	2	0	2	1.0	0.60	0.75	0.71
Banana	2	3	1	0	0.67	0.40	0.50	0.33
Kiwi	3	3	1	1	0.75	0.50	0.60	0.50

2.4.3 Evaluation of Hierarchical Classifiers

Over the years, many different approaches have been proposed in order to successfully compare flat classification classifiers with hierarchical ones. According to Silla and Freitas [40], in order to assess whether a hierarchical classification scheme is superior for a given problem is not trivial and results will vary depending on the evaluation measures taken. They recommend metrics detailed in Kiritchenko et al. [41] referred to as hierarchical precision *hP* which can be written as

$$hP = \frac{\sum_i |\hat{C}_i \cap \hat{C}'_i|}{\sum_i |\hat{C}'_i|}, \tag{2.18}$$

and hierarchical recall *hR* which can be written as

$$hR = \frac{\sum_i |\hat{C}_i \cap \hat{C}'_i|}{\sum_i |\hat{C}_i|}, \tag{2.19}$$

where $\hat{C}_i$ is the intersection set of the predicted class and its ancestors and $\hat{C}'_i$ is the same but for the true class. These metrics can be applied to both tree and directed-acyclic-graph structures. Much like in the context of flat classification, the two metrics can be combined to calculate a hierarchical F_1-score

$$hF_1 = 2 \cdot \frac{hP \cdot hR}{hP + hR}. \tag{2.20}$$

A few years earlier, Sun & Lim [16] proposed similar extensions to the traditional evaluation metrics of precision, recall and accuracy to make them more meaningful in a hierarchical context. They present measurements such as *category similarity* (based on cosine distance between feature vectors) and *category distance* (minimum number of edges needed to be traversed to go from one node to the other) to go from a binary outcome space of correct/incorrect prediction to support partially correct predictions.

More recent work was presented by Kosmopoulos et al. [42] where they developed formal frameworks for evaluation of hierarchical classifiers, using flow networks (A type of directed graphs) and set theory.

2.5 Related Work

One of the first works of SVMs for text categorization was presented by Joachims [43], where he argued that the classifier's properties were well-suited for this particular problem. He acknowledges the fact that one is forced to deal with large feature spaces when working with text and assures SVMs handle this aspect satisfactory in regards to performance and quality. SVMs are about maximizing the gaps between the support vectors and the hyperplane (independently from the number of features). By doing so it provides an innate protection against overfitting, where the decision boundary is dictated by the data to such an extent where the classifier becomes poor at generalizing.

Dumais et al. [14]explored the notion of classifying web content in a hierarchical context. They found some performance gains when going from a flat structure with many categories to a hierarchical structure using SVMs. Since an SVM outputs the category it predicts, Dumais et al. added an additional step to the algorithm in order to produce a probability distribution over categories.

Hernández et al. [15] discuss the concept of paths in a hierarchical classification context, more specifically in tree graphs and directed acyclic graphs (graph types suitable when modeling a corporate hierarchy). The paper presents an approach consisting of running classifiers on all nodes at the same time and obtaining probabilities which indicate an optimal path. It shows promising results for classification with multiple classes connected in a hierarchy compared to traditional methods.

Lo and Wang [44] propose a technique called AmaLgam for finding files that can contain bugs. AmaLgam work integrates a technique from Google which analyzes version history and tries to compute the probability of a file to include bugs. With a dataset of 3000 bugs, the Mean Average Precision (taking the mean of the average precision) of localizing buggy files compared to existing similar work is improved by 46.1%. Four experiments are performed on the four projects *AspectJ* [5], *Eclipse* [6], *SWT* [7] and *ZXing* [8]. Lo and Wang use four parameters to focus on when working with bug reports: the id of the report, the date it was submitted, a summary of the report and a detailed description of the report (this approach therefore has similarities with the work performed in this thesis). Lo and Wang describes three steps of how they use text preprocessing techniques in order to extract relevant information from the reports: text normalization, stop word removal and stemming.

In another work, performed by Lo et al. [45], the problem of using an optimal Vector Space Model (VSM) is described to be seen as an optimization problem where a distinct weight composition perform better than others, given a certain dataset. A search-based compositional bug localization engine is proposed where the training consists of textual documents and log files of bugs. Different TF-IDF schemes were tried with 15 different variants of VSM. Example of a variant is to take the logarithm of the term frequency instead of using the natural term frequency (counting the raw occurrence of words). The authors used the same dataset used in their previous work [44]. The performed experiments show that a compositional VSM performs 16.2% better than using a regular VSM (using a standard TF-IDF weighting scheme) in localizing bug reports and the Mean Average Precision is improved by 20.6%. This indicates that a prominent technique for Information Retrieval like TF-IDF [46] also has room for optimization.

Jonsson et al. [47] presented the DOLDA-model (Diagonal Orthant Latent Dirichlet Allocation) for responsibility assignment of trouble reports. The first part comes from Diagonal Orthant Probit Model which is a linear Bayesian classifier, and Latent Dirichlet Allocation which estimates probabilities of words that belong to certain topics that make up textual content unsupervised. The DOLDA supervised learning model extracts these topics from the

[5]https://eclipse.org/aspectj/
[6]http://eclipse.org/eclipse/
[7]https://eclipse.org/swt/
[8]https://github.com/zxing/zxing

textual contents of the reports and produces a probability distribution over the classes. The results in the paper showed lower yet competitive performance in regards to prediction accuracy compared to state-of-the-art approaches, but the model output provides greater insight of why the decision was made thanks to topic modeling.

Regarding CNNs, more recent work [48, 49, 50, 51] shows state-of-the-art results for various tasks regarding classifying and identifying objects in images. In addition to this, it also shows that utilizing CNNs is a highly competitive approach compared to other existing ones today.

The use of CNN based approaches is continuously growing together with its community. Its applicability is including, but not limited to, images. In the last few years, there has been an uprise in the developments of using CNN as a tool for Natural Language Processing (Section 2.3) [52]. For example, classifying if short messages such as *tweets* (Twitter messages) [53] has been of a positive or a negative nature. Understanding the context of a sentence, tweets in this example, is known as *sentiment analysis*. With the help of k-max pooling (k being the length of the sentence and depth of the network), varying length of sentences has been handled in order to capture word relations and gave a good result on classifying questions [54]. For instance, a 25% error reduction with respect to other work, which was top at the time of the conducted work [55], is an indication that CNNs are suited for this cause.

The order in which words appear can be taken advantage of to make better predictions of *word embeddings* (low-dimensional vector representations of the original word), as well as directly applying the same CNN-like structure used for images to words with little modification and will still yield competing results [56, 1]. A similar approach has also been successful, where a semi-supervised model trained on both unlabeled and labeled data performs topic modeling and sentiment analysis [57].

Zhang et al. [25] bring forth several interesting findings to take into account when setting up a CNN for handling words. It is found that performance depends on the type of word vector representation used as well as task. Two examples of popular tools used to form these representations are *word2vec*[9] and *GloVe*[10]. Both show promising results pointing to superiority compared to one-hot vector representations (vector with binary numbers, with a single one and the rest are zeros) if there is not a vast amount of data to train the model with. Points like the size of the filter also impact the system and should be tuned accordingly to the data in the domain. The number of feature maps is linked to the training time of the model and Zhang et al. are also suggesting that 1-max pooling (Section 2.2.3) is used as it performs better than other techniques. In addition to this, it also mentions that regularization (a technique in making a model more general) does not impact the performance of the system considerably.

Kim [1] presents a model for classifying sentences which includes a one–layered CNN with a frugal tuning of the hyperparameters. The model is pretrained with initialized word vectors from the help of word2vec trained on 100 billion words from Google News.

[9] https://code.google.com/p/word2vec/
[10] http://nlp.stanford.edu/projects/glove/

3 Method

This chapter describes the work flow and the methods used in this project. The work flow is divided into three main interconnected parts. The process of the work was carried out as an iterative model including research, development and testing. Doing extensive background and planning is motivated by trying to avoid potential time-consuming pit falls when the work is enacted.

3.1 Tools and Frameworks

This section contains descriptions of the tools and frameworks used for this work.

3.1.1 Scikit-Learn

Scikit-Learn[1] offers a suite of machine learning techniques [58], including data classification. The interface allows programming in Python and streamlines the process of vectorizing input data, training classifiers and evaluating results. This allows for quick iterations through several combinations of learning algorithms and classifiers to find the most suitable candidates for our purposes. Among others, Naive Bayes and linear SVM were explored. In addition, Scikit-Learn is also compatible to use together with TensorFlow which makes it prominent candidate for the task at hand.

3.1.2 TensorFlow

The CNN is implemented in *TensorFlow* [59], an open source Python library (containing optimized C++ code). It has been developed by the Google Brain Team and it is suited for optimization problems in the area of machine learning and when working with neural nets. The library provides implementations of many useful standard algorithms and is comparable with other machine learning libraries in terms of performance. TensorFlow has support for running on both CPUs and GPUs (with the help of the parallel computing platform CUDA [2]). These factors as well as the recommendation from Ericsson supervisors made this a promising tool to use for the work.

[1] http://scikit-learn.org/
[2] https://developer.nvidia.com/about-cuda

3.1.3 MHWeb

MHWeb is a web application which allows technicians in several areas together with other relevant personnel within Ericsson, to track the progress and status of TRs. Each TR entry includes information such as a descriptive title, text describing the effects (observation text) of the bug and a notebook. The notebook is used as a tool where Ericsson personnel can submit questions and updates regarding the status of the report. Only the observation and heading text are regarded as the free text description of a TR, and thus has been extracted to be used as a source of data in this work.

3.2 Preprocessing

In order to work with adequate datasets a need for preprocessing exists. The datasets used in this work are located in a large database and are available via MHWeb. With the help of a communicating script and an API, datasets could be accessed and fetched from the database and could then be stored locally.

The TRs used for the experiments comes from two datasets. One which has been extracted from a total of 10 218 TRs and another of total 66 066 TRs. Both datasets consist of the same TRs, meaning that the larger dataset is an overlap of the smaller dataset together with additional TRs. This distribution can be depicted further in Figure 3.1. The two sets are also used in similar thesis works which are performed in parallel with this work. It is beneficial for all involved parties working with TRs at the time that they use the same set of TRs in order to compare, share and reason about findings during the work process.

As a TR is produced partly by humans, the TRs have been of varying quality and inconsistency is a looming threat when working with classification, especially in regards to the free text section of the reports. This has been one of the main challenges during the preprocessing phase. When preprocessing techniques are used in order to extract relevant data, text which contains for example single characters, symbols (#, %, $*$ and @ etc.) or other log data is neglected since it is deemed as noise in the preprocessing. This can also be observed when plotting the graph of the word embeddings. After the preprocessing is performed it results in that the remaining text becomes more sparse, which in turn means less data for the model to train on.

The number of documents belonging to each one of the 18 root (group) classes can be depicted in Figure 3.1. The documents can be sub-categorized further to achieve a multi-layer hierarchy, described further in Section 3.3.4. The purpose of the hierarchical approach is to come as close to reality in regard to the route of a trouble report and minimize its deviation, to cut down cost and resources needed to correctly classify it.

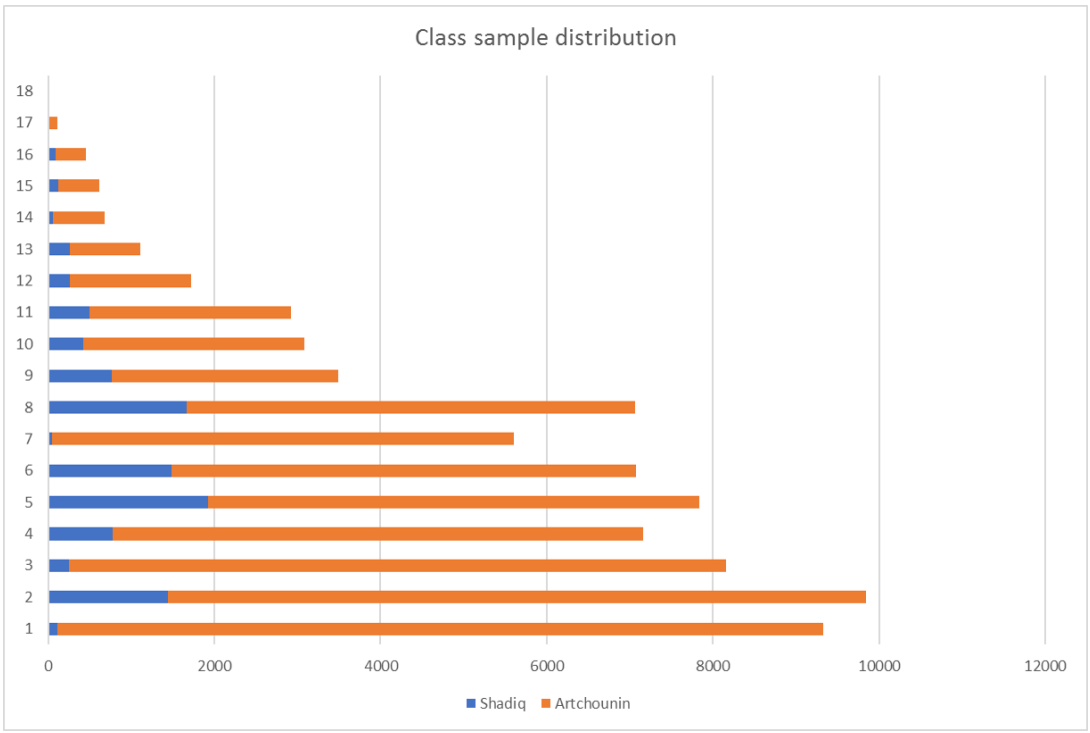


Figure 3.1: Class label distribution over two datasets.

3.2.1 Preprocessing techniques

Two types of preprocessing techniques have been used in the experiments:

- **Shadiq's Boilerplate** - A boilerplate technique based on the one presented by Kohlschutter et al. [60] has been used in order to extract human written code from the observation and heading text. This means that machine generated code is neglected and ignored and only human written text is present. Shadiq [61] reached an accuracy of 93% in extracting human written text in the smaller TR dataset in a thesis work covering the same domain as this work.
- **Artchounin's Ad Hoc Regular Expressions** - This technique uses regular expressions to remove particular words of the TRs which have been deemed noisy. This is an ad hoc solution which is specialized for the larger TR dataset performed by Artchounin [62] in a thesis work covering the same domain as this work. This technique has both human and machine text present.

In addition to this, the same textual cleaning with regular expressions used in Kim's work [1] is also used in combination with each respective preprocessing technique.

Table 3.1: Table of the datasets used for each preprocessing technique.

Preprocessing Technique	**Number of TRs**
Shadiq's Boilerplate	10 218
Artchounin's Ad Hoc Regex	66 066

3.2.2 Word Embeddings

In relation to the preprocessing of data, the text from the TRs is to be used in a format which is valid for the CNN. In order to feed the model with data from the words of the TRs, Word2vec is used to create word embeddings vectors. Words which are closer together in the word embedding space share similarities. The result of the clustering process depends on the preprocessing of TRs before they are used with Word2vec. Since the dataset is quite extensive, the preprocessing plays an important role in refining the datasets so that the model can be trained to make accurate predictions.

Various pretrained word embedding models were configured with the help of Word2vec and were used as a tool for the CNN. All experiments, using pretrained word embedding models performed better than not having it, as can be seen in Table 4.5. Throughout the experiments, different datasets were used and word embedding models were trained accordingly. The different datasets seen in Table 3.3 are consisting of the text corpus gathered from the TR samples used for training and the remaining TRs (the ones not used for training or evaluation). The different types of experiments (Stratified and Overlap) are explained further in Section 3.3.3. This explains the amount of different Word2vec models. Parameters such as *Sample* and *Iterations* has been static throughout the various models. This was done since it was desired to test how the datasets behaved with different types of models. In addition to this, it is interesting to observe differences between using a Skip-gram model or CBOW model. All different pretrained models, the configurations used and their respective input can be depicted in Table 3.2 and Table 3.3. Parameters such as *Max Window*, *Size of Embeddings* and *Sample rate* has been set after recommendation from the Word2vec developers [3].

Table 3.2: Table over Word Embedding datasets used from Shadiq and Artchounin.

Dataset Name	Words	Vocabulary size
Shadiq		
A	916 029	11 733
B	966 962	7066
Artchounin		
C	20 558 751	139 835
D	17 444 211	128 392
Stratified		
E	20 143 502	136 987
Overlap		
F	984 391	10 955
G	5 608 590	44 507

[3] https://code.google.com/p/word2vec/

Table 3.3: Table over Word Embedding configurations.

Name	Dataset	Type	Size	Max Window	Hierarchical	Sample	Iterations
SKIP1	A	SKIP	200	10	Yes	1 e^-4	15
SKIP2	B	SKIP	300	10	Yes	1 e^-4	15
SKIP3	C	SKIP	300	10	No	1 e^-4	15
SKIP4	D	SKIP	300	10	Yes	1 e^-4	15
SKIP5	D	SKIP	300	10	No	1 e^-4	15
SKIP6	E	SKIP	300	10	No	1 e^-4	15
SKIP7	F	SKIP	300	10	No	1 e^-4	15
SKIP8	G	SKIP	300	10	No	1 e^-4	15
CBOW1	A	CBOW	300	5	No	1 e^-4	15
CBOW2	B	CBOW	300	5	Yes	1 e^-4	15
CBOW3	D	CBOW	300	5	No	1 e^-4	15
CBOW4	E	CBOW	300	5	No	1 e^-4	15
CBOW5	F	CBOW	300	5	No	1 e^-4	15
CBOW6	G	CBOW	300	5	No	1 e^-4	15

3.3 Models

This section elaborates on the various types of models and classifications that are used in the work.

3.3.1 Flat Classification

When training a model, the text for each trouble report is loaded from disk and put into data arrays. Since the feature space is large, using ordinary data arrays is infeasible as they will consume too much memory. The fact that most of the cells in the array will be zero and utilize sparse data arrays can be exploited, thus keeping the memory footprint in check.

Next, the data is fed into a count vectorizer that maps each individual word and all combinations of two words together (known as 1-grams/unigrams and 2-grams/bigrams respectively) in the text to an integer value corresponding to the frequency. The vectors are then transformed using a TF-IDF transformer. This will change the word counts to numerics representing how important a particular word is in the context. Finally, the transformed matrix is passed to the linear SVM which uses the one-vs-all method to accommodate multi-class classification, or alternatively a multinomial Naive Bayes classifier. The pipeline functionality of Scikit-learn makes it easy to try out different configurations.

3.3.2 Convolutional Neural Network Architecture

The starting point of the architecture of the CNN has been developed with inspiration of the structure by Kim [1] where a relatively straightforward CNN achieves a good result. It enables creating an architecture with a solid backbone which is open for possible extensions such as adding more convolutional layers/filters/pooling etc., even though this work aims to delimit major hyperparameter tuning. The default parameters in the architecture of the CNN are based on those described in the work by Kim. Table 3.4 shows the list of hyperparameters which is the starting point of the CNN. A general overview of the architecture of the CNN can be depicted in Figure 3.2 and consists of the following parts:

- **Embedding layer** - This is where the preprocessed TRs are converted to a format suitable for convolutions.
- **Convolutional layer** - By having receptive fields (filters) performing convolutions features of interests can be captured from the embeddings.
- **Activation function** - Creating non-linearity for the domain with the help of Rectified Linear Unit.
- **1-Max Pooling** - Reducing the spatial size, while keeping interesting findings from the convolutions.
- **Softmax layer** - In order to estimate probabilities for all the classes, a softmax layer is used to interpret and estimate predictions correctly on the data from the max pooling.

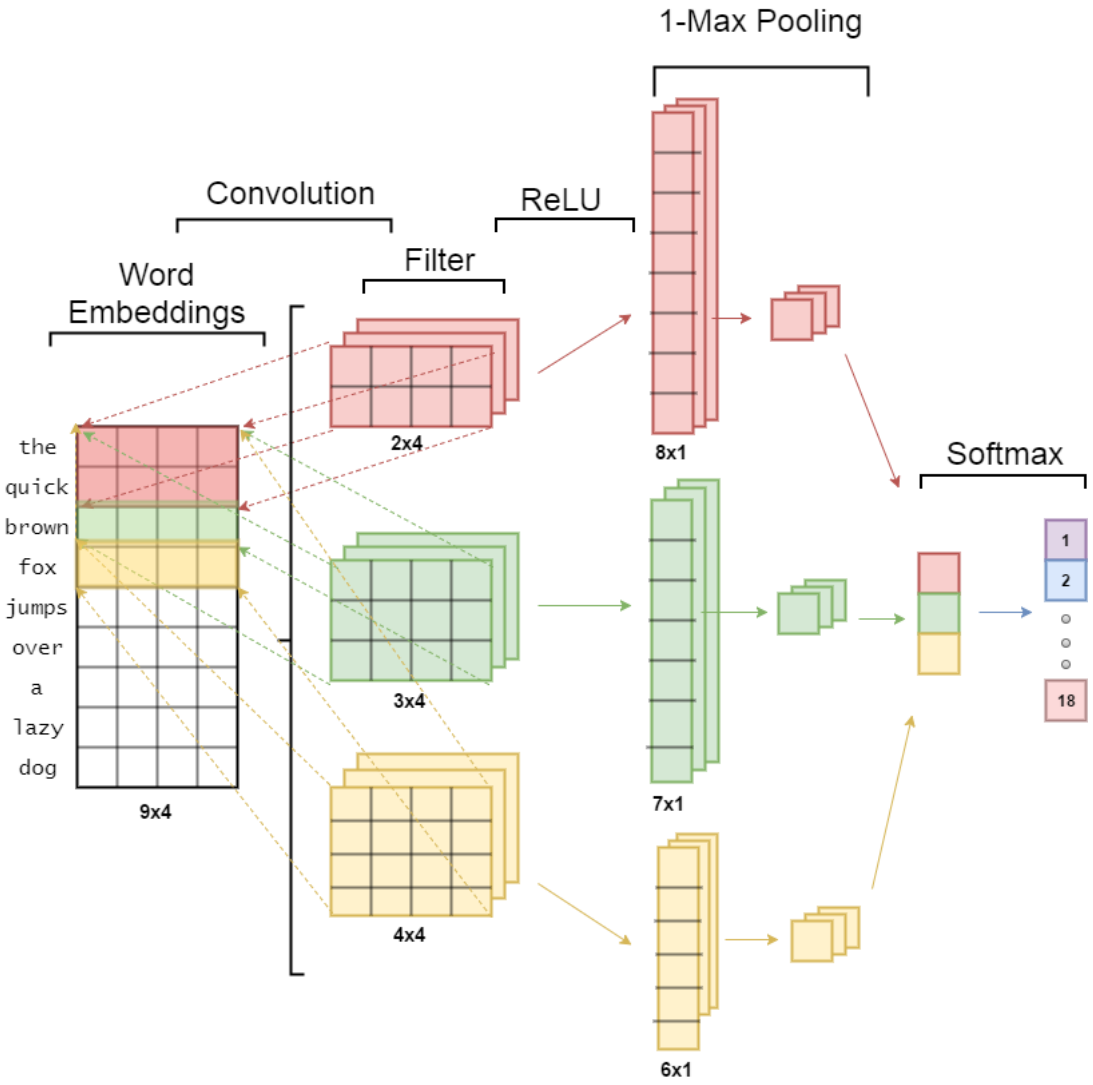


Figure 3.2: Overview of the architecture of the Convolutional Neural Network.

Table 3.4: Table with parameters used in the architecture of the CNN.

Parameters	Value	Comment
Filter sizes	3, 4, 5	Three filters of varying size.
No. filters per size	128	Each filter has 128 hidden layers.
Sequence length	Depending on dataset.	The length of sentences used.
No. groups	18	The group classes in the company hierarchy.
No. classes	153	-
Vocabulary size	Depending on dataset.	Numbers of words used for the vocabulary.
Word embedding tool	Word2vec	Tool used for converting to embeddings.
Dropout rate	0.5	Rate of dropping units within the network.
Optimization method	Stochastic Gradient Descent	-
Stride size	1	Filters moving with a certain stride.

3.3.3 Convolutional Neural Network Experiments

A series of experiments are conducted in order to find differences and patterns when working with the CNN. The purpose is to find and reason about how the quality of the model and its performance varies based on different preprocessing techniques on training samples and the models configurations.

The starting hyperparameters are tuned and adjusted for two reasons: adapting to the resources available where the CNN is trained and depending on how the result varies from experiments, meaning that if results vary depending on small tuning this will be explored further. However, some parameters such as Epochs (One forward pass and one backpropagation of all training examples) are kept static throughout the experiments. The structure of the experiments can be depicted in Figure 3.3.

The various parts involved in the experiments can be seen as separate modules which gradually are interconnected. This is done in order to adjust each module to the task at hand and to gain an understanding of existing requirements before connecting them. This is done in hopes of increasing the chances of gaining better results.

A 90/10-ratio split between train and development data has been used during the training of all models.

3.3.3.1 Shadiq and Artchounin models

The CNN experiments regarding two datasets of TRs, one preprocessed with Shadiq's and another with Artchounin's preprocessing technique. The same samples of preprocessed TRs from each dataset has been re-used throughout the experiments.

It is desired to use an even class distribution (no skewed classes) between all TR training examples, and the same amount of unseen TR for evaluating the model. This number is controlled by two things: computational resources available at the time when conducting the experiments and the class distribution of the available TR dataset (some categories having a sparse amount of TRs). It is desired to find the highest equal amount of training data from each class, as well as keeping the same amount for evaluating the model in order to find how general the model is to unseen TRs. However, all TRs which are not used for evaluation can and will be used for preloaded word embeddings. These experiments will help reason about how different configurations affects the models, as well as give indications on what preprocessing technique performs better even though they use different TRs.

3.3.3.2 Stratified Sampling

As the class distribution is skewed, it is also of interest to see how a model behaves when trained on a varying amount of TRs from each class. This is called Stratified from now on, where a percentage of the total TRs are extracted (randomly selected each time) and used for training. Two classes will be removed when conducting these experiments: the smallest class and one class which is deemed inconsistent with regards to the nature of the TRs it store. This will give an additional basis for reasons on how certain classes can impact the overall performance of a model.

3.3.3.3 Overlap

In order to draw conclusions regarding what preprocessing technique performs better than the other, an overlapped subset of TRs from both Shadiq and Artchounin will be selected. In the same way as the other CNN models, they will be trained with both Skip-gram and CBOW. By training models on an overlapped subset of TRs together with the findings from the experiments in Section 3.3.3.1, characteristics of which preprocessing technique seems to be more beneficial when working with TRs can be reasoned about.

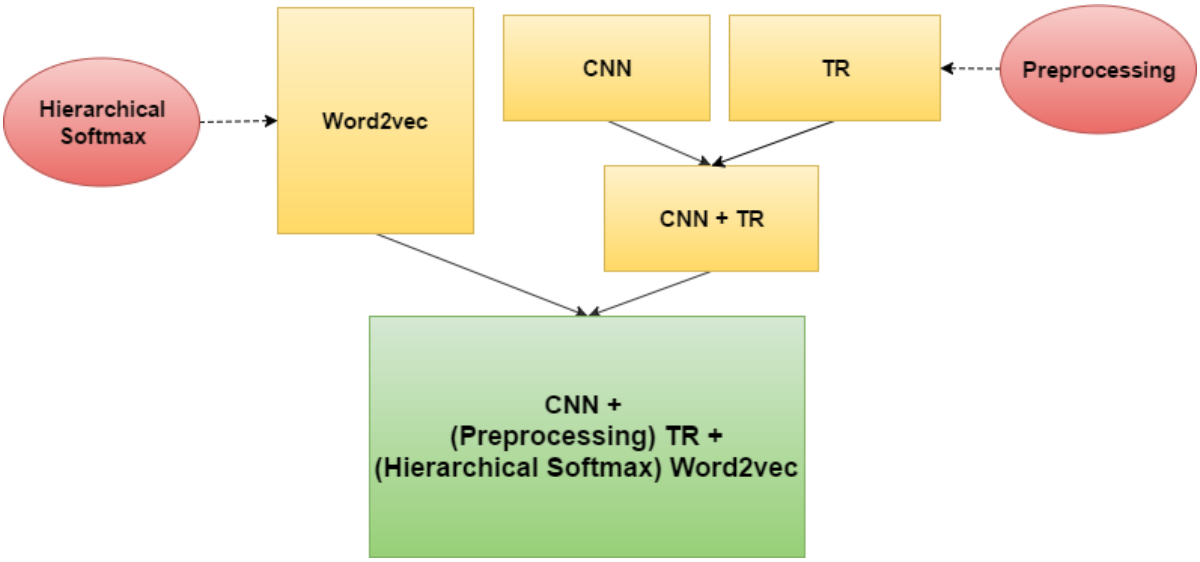


Figure 3.3: The various parts of the CNN experiments. The parts (yellow and red) can be seen as separate modules which are interconnected to form the final product (green).

3.3.4 Hierarchical Classifier

Instead of attempting to classify trouble reports in a flat context where the number of classes is large, it is sought to follow the intuition that classification can be more accurate and meaningful if the relationships between them are accounted for.

The dataset made available includes roughly 150 classes. Attempting to classify these in a flat context will likely lead to poor accuracy as there is a large amount of classes which puts the constraint that the classification must be very accurate, so instead, each class is mapped to a higher-order category based on the previously mentioned guidelines. This leaves 18 group classes in the first layer, branching down to the total of 150 classes.

By obtaining probabilities for all nodes in the hierarchy with regards to some presented example, there is an intuition that a proper combination of these probabilities will indicate

the most likely route from the root node down to a leaf node that the example should take. This is meant to serve as data for personnel overseeing the trouble report routing process. Our method of obtaining these probabilities is inspired by Hernandez et al. [15], who trained local classifiers on all parent nodes in the hierarchy.

The training is performed by dividing the class samples to their respective parent node. A classifier model is fit for each parent node in a recursive fashion, starting at the root node of the tree. As such, the root node classifies to 18 classes in the first layer, and each of these classes has a corresponding node with its own classifier to predict its descendant classes for a total of 150+ classes in the bottom layer. Figure 3.4 shows this process.

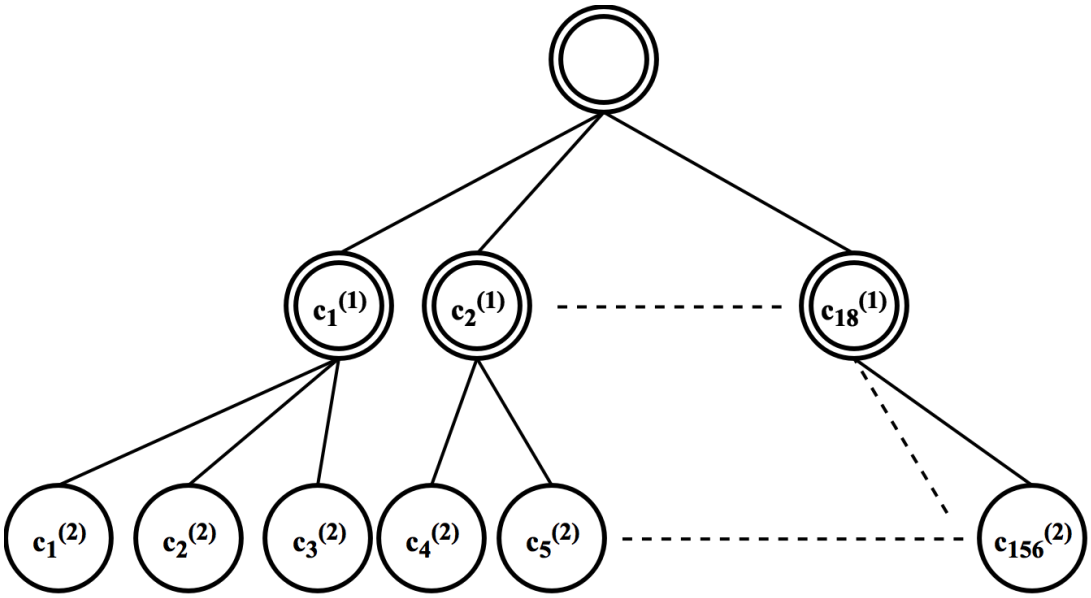


Figure 3.4: Classes represented in a hierarchical structure as tree nodes and edges. Double-lined nodes each have one classifier which is trained with data.

Another set of samples is used for prediction; one by one they are propagated through the classifiers which produce probability estimates for all their descendants. The probabilities are recursively combined per-path by multiplication. The implementation then returns the optimal path in the form of a list of nodes (one node per level), as well as a probability value that is meant to indicate the chance that the sample belongs to that path. To produce these probabilities, the probabilistic classifier Multinomial Naive Bayes has been evaluated.

3.4 Evaluation

Evaluation during the experiments with base classifiers and the hierarchical classifier was conducted by running ten-fold cross-validation 2.4.1 to keep them comparable to the flat classification experiments. Due to the relatively simple hierarchical structure of the classes, the metric used to display results is the combination of these metrics called hierarchical F_1-score, since the values for these metrics will be the same for a given sample.

The implementation of the CNN that was used utilizes partitions of the supplied dataset known as the train set, development set and test set. During training of the network, training samples from the train set propagates through the CNN and it adjusts neuron weights accordingly. Every 50th training step, it is shown all samples from the development set to evaluate training progress in relation to data it is not being trained with.

4 Results

This chapter presents results for the base classifiers from a flat classification context, as well as a hierarchical context. It also covers the results from the convolutional neural network classification.

4.1 Traditional Classifiers - 18 Classes

This section elaborates on the results for classification of the dataset in a flat classification context. Artchounin's Ad Hoc Regular Expressions dataset has been used. Table 4.1 shows classification accuracies for different configurations of features and classifiers. We note that the best performing feature configuration (highlighted in bold) includes both unigrams and bigrams, although the benefit measured compared to only unigrams is small.

Table 4.1: Classification accuracy (18 classes) for a number of classifiers with varying features. Results have been obtained by running ten-fold cross validation and taking the average across all folds.

Features	**Multinomial Naive Bayes**	**SVM (Linear Kernel)**
1-grams + TF-IDF	0.692	0.824
1&2-grams + TF-IDF	**0.715**	**0.831**

A classification report of one of the folds by the best performing model is showcased in Table 4.2. An average F_1-score of 0.84 is achieved, making the performance of the model look suitable for the cause.

Table 4.2: Evaluation results of the best performing fold in the cross-validation experiment with 1&2-grams + TF-IDF as features with a linear kernel SVM as classifier.

Class	**Precision**	**Recall**	F_1**-score**	**#Evaluate**
1.	0.56	0.37	0.45	67
2.	0.80	0.60	0.69	58
3.	0.81	0.86	0.84	830
4.	0.97	0.83	0.89	35
5.	1.00	0.14	0.25	7
6.	0.84	0.85	0.84	248
7.	0.93	0.92	0.92	837
8.	0.97	0.97	0.97	577
9.	0.73	0.85	0.79	553
10.	0.70	0.74	0.72	569
11.	0.84	0.84	0.84	629
12.	0.72	0.56	0.63	268
13.	0.78	0.78	0.78	550
14.	1.00	1.00	1.00	2
15.	0.79	0.72	0.75	249
16.	0.36	0.14	0.20	36
17.	0.94	0.94	0.94	934
18.	0.88	0.80	0.84	157
avg / total	0.84	0.84	0.84	6606

4.2 Traditional Classifiers - Preprocessing Technique Comparison

This section presents classification results for the overlapping part of Shadiq's and Artchounin's datasets. Ten-fold cross-validation was performed on the data. We observe in Table 4.3 that the SVM performs slightly better than the Multinomial Naive Bayes classifier, and that Artchounin's preprocessing technique yields some additional gain in accuracy compared to Shadiq's.

Table 4.3: Preprocessing comparison for base classifiers. The highest accuracy is marked in bold.

Classifier	**Shadiq**	**Artchounin**
Accuracy		
Multinomial NB	0.460	0.482
SVM (Linear Kernel)	0.528	**0.566**

4.3 Hierarchical Classification, 150+ Classes

This section shows results from the hierarchical classification experiments. Values are presented as both micro-averaged (true positives, false positives and false negatives summed up individually for each class which are then averaged) standard and hierarchical F_1-score, in

flat and hierarchical classification contexts. Entries for the non-probabilistic SVM classifier in regards to hierarchical classification are not applicable, since the implementation of the hierarchical model relies on its classifiers to produce probability distributions to complete evaluation.

The implementation of the hierarchical classifier consumed a lot of memory during experiments. A portion of the Artchounin dataset was used, roughly in the order of 8000 trouble reports.

Table 4.4: Classification scores expressed in standard and hierarchical F_1-score.

Classifier	**Flat Classification**	**Hierarchical Classification**
Standard F_1-score		
Multinomial NB	0.387	0.101
SVM (Linear Kernel)	0.689	*N/A*
Hierarchical F_1-score		
Multinomial NB	0.392	0.365
SVM (Linear Kernel)	0.694	*N/A*

4.4 Convolutional Neural Network

This section will describe how the CNN experiments have been performed together with their results. The conducted experiments cover two different preprocessing techniques of the TRs.

Throughout the experiments the model has kept its original architecture (filters, number of filters per filter size, convolutional layers, max pooling) static, described in Table 3.4, where other parameters such as: embedding dimension, regularization value, pretrained word embeddings and batch size modified has been tuned and configured. The parameter batch size was changed due to resource limitations when the dataset increased.

With the help of Tensorflow's visualization tool *Tensorboard*[1] the architecture of the CNN implementation can be inspected in Figure 4.1. This figure depicts a more detailed image than Figure 3.2 which hopefully gives the reader a better understanding of the networks various building blocks, even though they both represent an overview of the CNN architecture.

[1] https://www.tensorflow.org/get_started/graph_viz

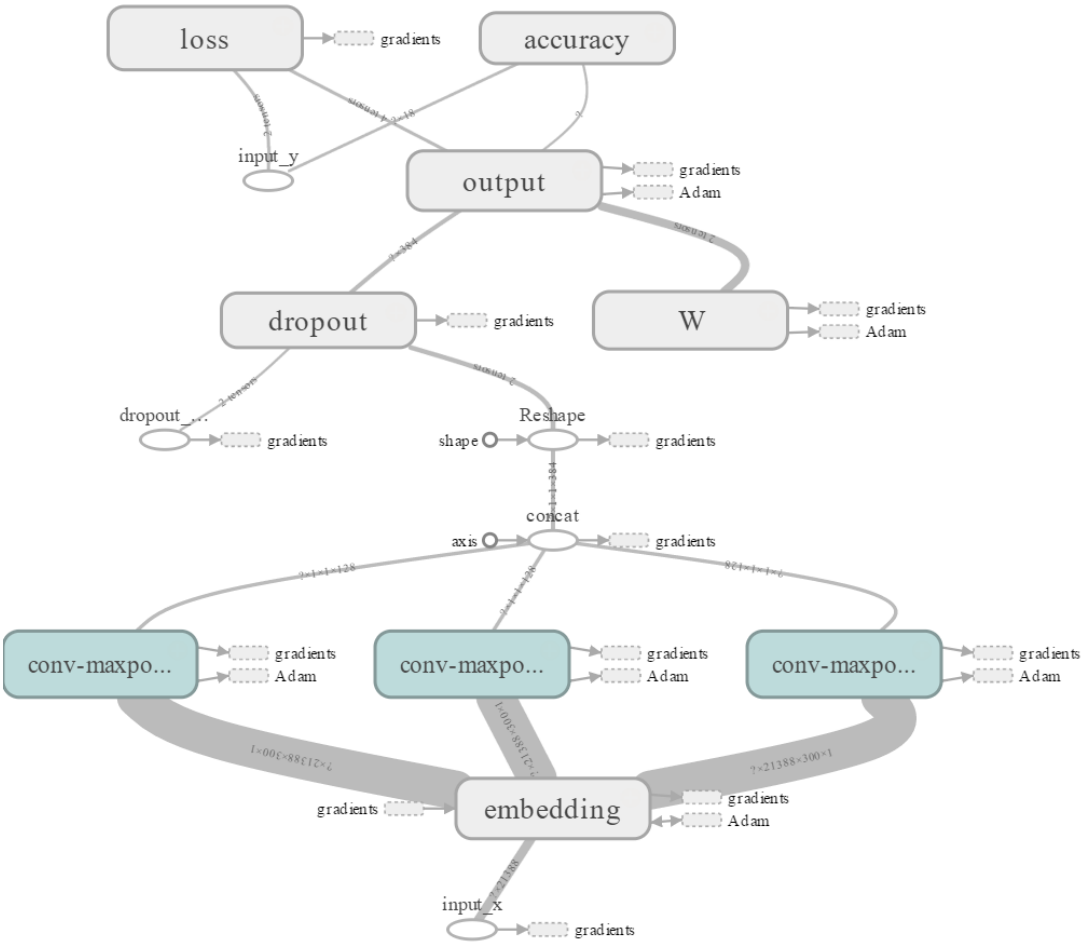


Figure 4.1: A high level representation of the CNN implementation shown with the help of Tensorboard. Starting from the bottom of the image going upward, an input enters the ***Embedding Layer*** (grey rectangle) and then passes through the three ***Convolutional Layers*** (light blue rectangles) with ***Max pooling***. The input is then concatenated (small white circles) and has ***Dropout*** (grey rectangle) applied to it. The *Output* is applied with ***Softmax*** where ***Accuracy*** and ***Loss*** are calculated together with ***Weight***(***W*** in the graph) updates (grey rectangles).

4.4.1 Experiment results

A total of 17 CNN models, 5 with Shadiq's Boilerplate, 6 with Artchounin's Ad Hoc Regular Expressions, 2 Stratified and 4 Overlap, have been trained and evaluated with different configurations. The aim has been to observe and identify factors which has been helpful in boosting the accuracy, precision, recall and F_1-score of the model, as well as giving a basis for drawing conclusions regarding the impact and characteristics between different preprocessing techniques. It was noted early in the experiment phase that with small changes in configurations from the training of one model to the other, the results varied with a considerate amount, as to why more models were developed with various configurations.

The results from the experiments is plotted into two plots. The plot in Figure 4.2 depicts the accuracy on the Y-axis and the experiment ID, in the order they were performed, on the X-axis. The blue line shows Shadiq's Boilerplate TR dataset and the red line is Artchounin's Ad Hoc Regular expressions TR dataset. The green line represents the Stratified experiments. Even though the experiments did not use the identical TR dataset, it gives an indication that Artchounin's preprocessing technique performs better in general than the one of Shadiq.

In Figure 4.3 a plot depicting accuracy together with the additional metrics Precision, F_1-score and Recall can be observed in relation with each other in order to get an overview how they are related to each other. The value of the metric is shown in Y-axis and the experiment ID, in the order they were performed, on the X-axis.

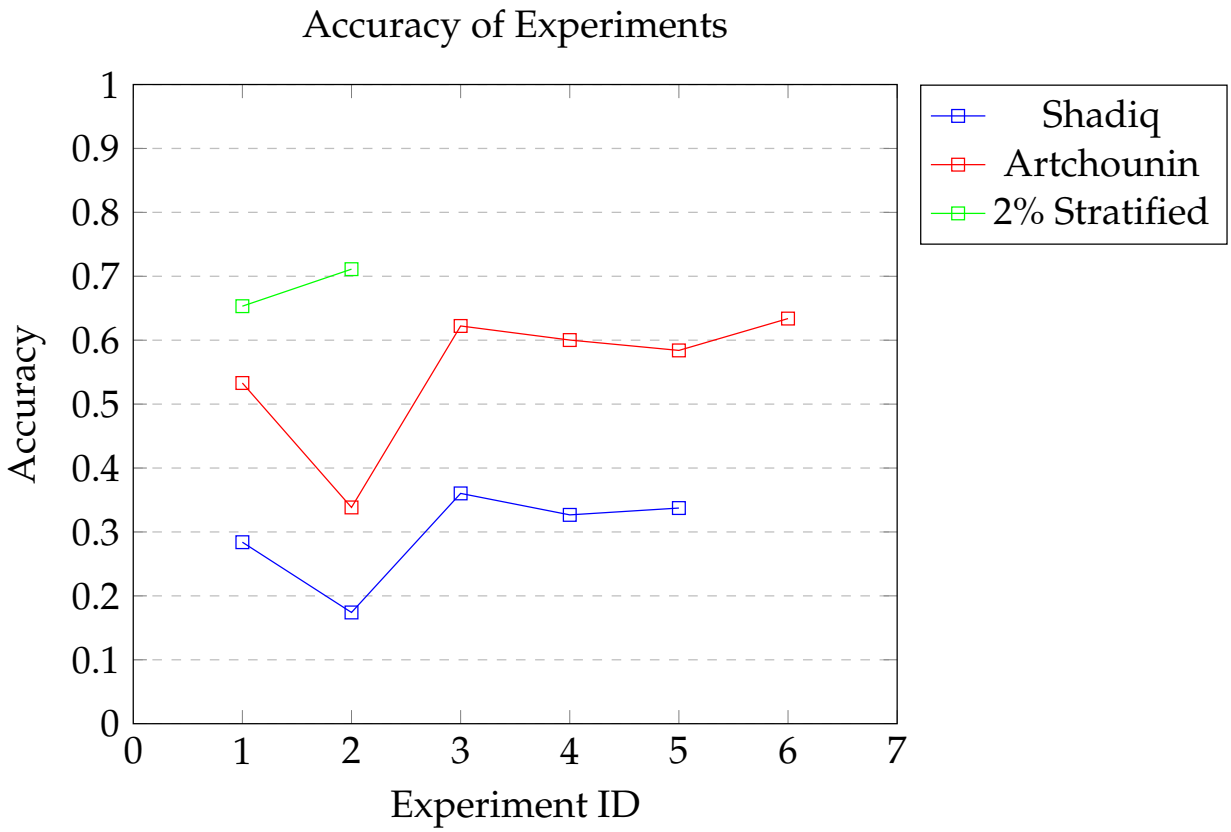


Figure 4.2: The Accuracy results from all experiments listed in Tables 4.5, 4.6 and 4.7 are shown. The Y-axis shows the accuracy value of each model and the experiment ID, in the order they were performed, on the X-axis. Artchounin's Ad Hoc Regular Expression (red) dataset outperforms the Shadiq's Boilerplate (blue) dataset one in almost all experiments. The Stratified technique (green) is performed on 16 classes.

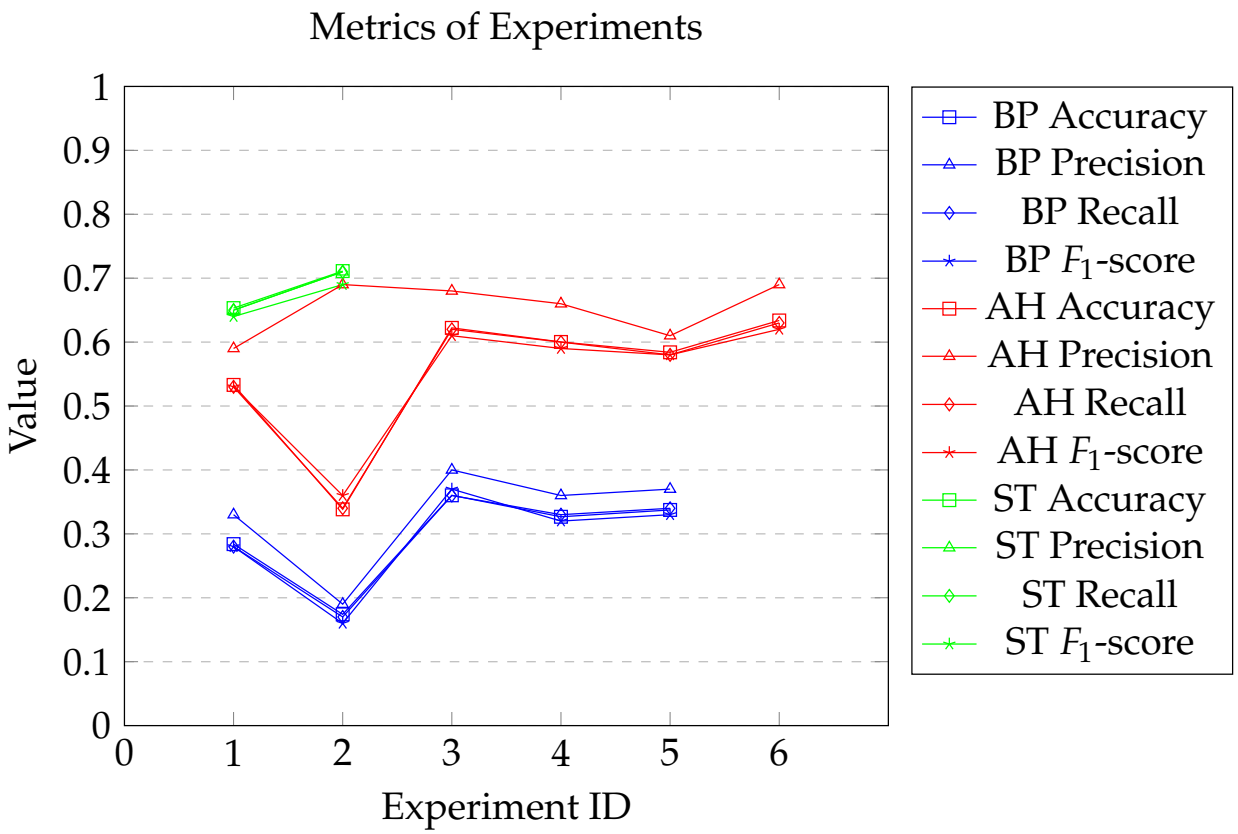


Figure 4.3: A collective plot showing all measured metrics of the CNN models performance. The value of the metric is shown on the Y-axis and the experiment ID, in the order they were performed, on the X-axis. BP (blue) in the plot is short for Shadiq's Boilerplate, AH (red) is short for Artchounin's Ad Hoc Regular Expressions, ST (green) is short for the Stratified experiments.

In order to draw conclusions of an identical TR dataset, but with various preprocessing techniques Figure 4.4 can be observed for. This shows that in the Overlapping experiments, Artchounin's preprocessing technique slightly outperforms Shadiq's with both skip and cbow preloaded word embeddings when measuring accuracy. Figure 4.5 shows an extending plot with accuracy together with the metrics precision, recall and F_1-score of the overlapping models.

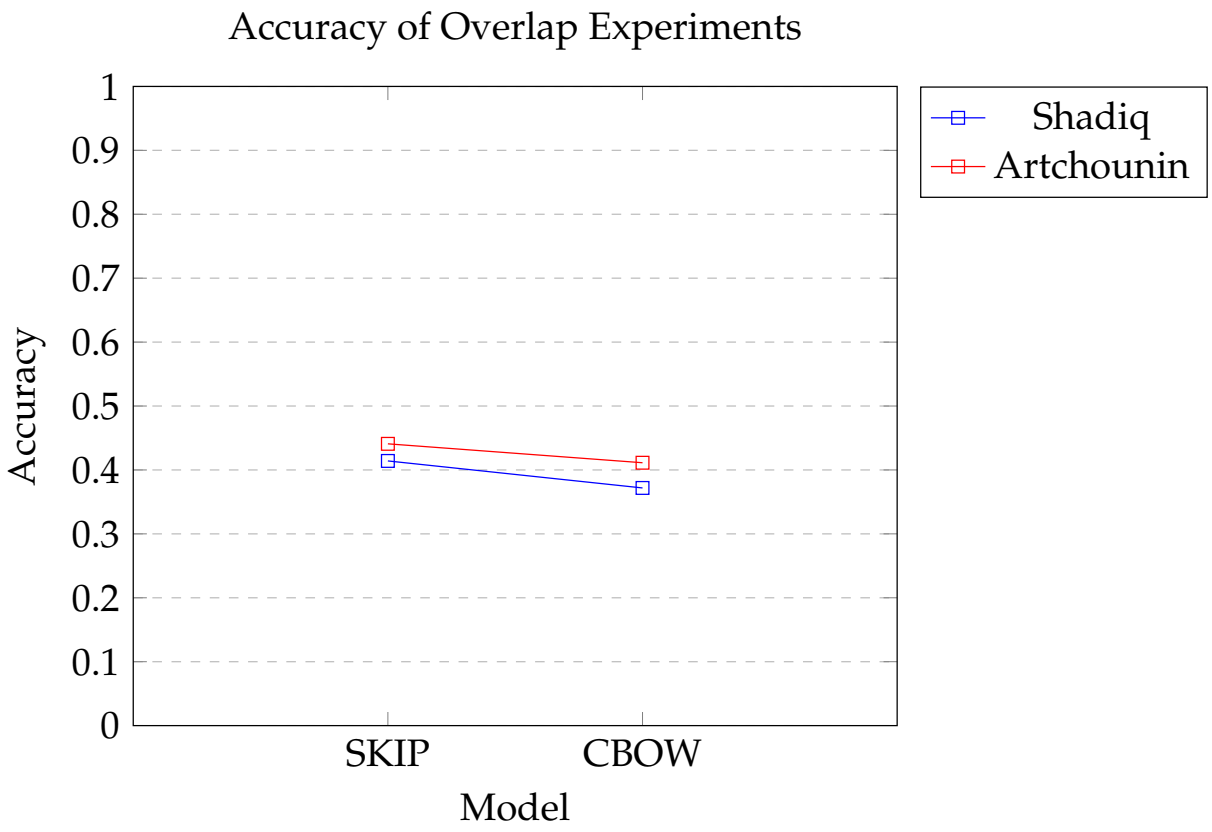


Figure 4.4: The Accuracy results from the experiments with same set of 800 TRs, indicating a difference between accuracy of Shadiq's and Artchounin's preprocessing technique. The Y-axis shows the accuracy value of each model and the experiment ID on the X-axis.

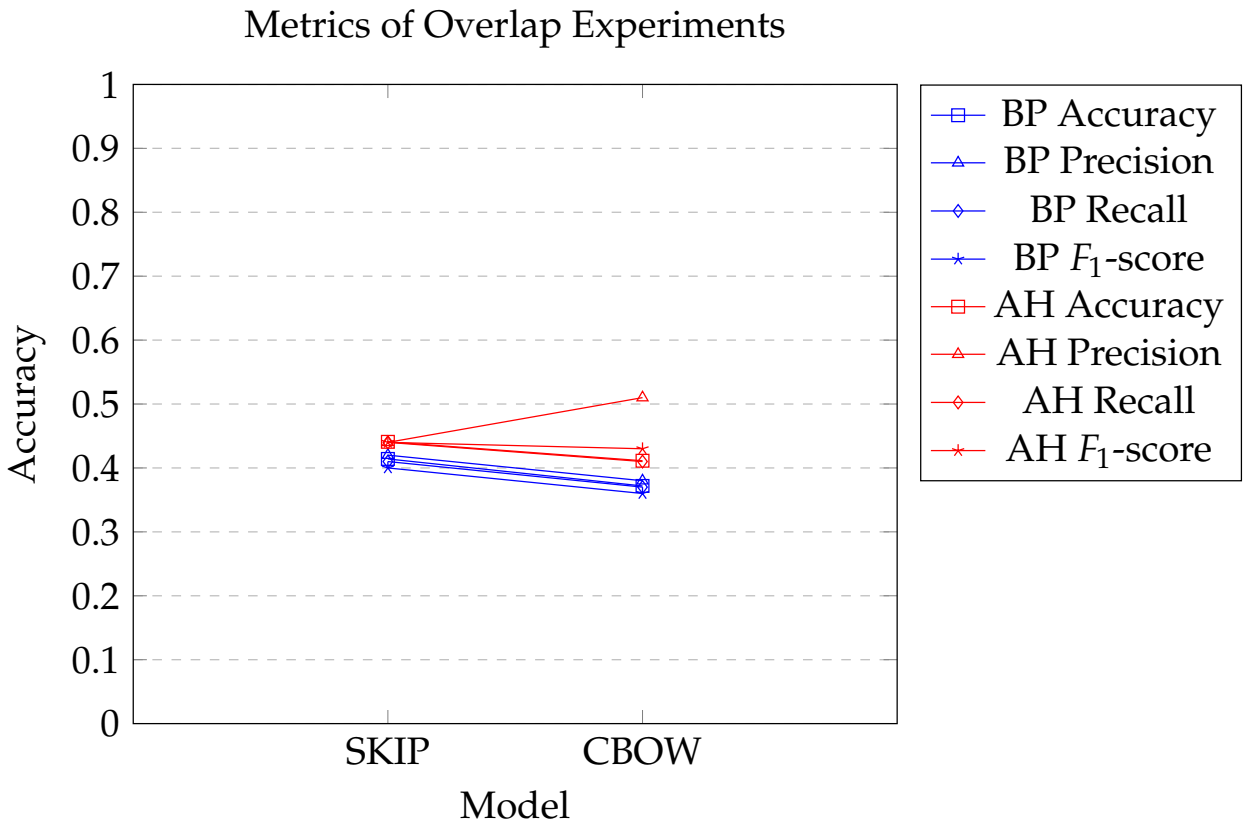


Figure 4.5: A plot showing all metrics of the overlapping CNN models performance. The value of the metric is shown on the Y-axis and the word embedding model type on the X-axis. BP in the plot is short for Shadiq's Boilerplate, AH is short for Artchounin's Ad Hoc Regular Expressions.

The Tables 4.5, 4.6 and 4.7 together show the experiments and configurations of the 17 CNN models in detail. The models acquiring the best performance in each subgroup is marked with bold text. The metrics Precision, Recall and F_1-Score are all listed in a weighted score, meaning that they are based on the average of metric for all classes divided by the total number of evaluated (listed as #Evaluate in the table) samples. Some interesting observations are:

- The accuracy of the models has varied from 17% to 63% when working with 18 classes. The main reason for this is the pretrained word embeddings and how the dataset has been preprocessed. An example of this is the Shadiq experiment with ID 2, which yielded a 17% accuracy. This experiment was not using any pretrained word embedding vectors.
- The experiments yielding the highest accuracy have been the latest one performed, where the dataset and configurations has been tuned continuously.
- The experiments which never use a preloaded word embedding (Shadiq experiment with ID 3, 5 and Artchounin experiment with ID 2) have lower accuracy than the ones which has used it.
- The experiments done with Artchounin's Ad Hoc Regular Expressions have more than 11 000 additional words in its dataset vocabulary.

- Models which are pretrained with word vectors with a Skip-gram model generally perform better than those loaded with CBOW model.
- The Stratified experiments yields over 71% accuracy, however this is done on 16 classes, meaning that the accuracy is likely to get boosted when the model has fewer classes to chose from when performing classification.
- The Overlap experiments indicates that the skip model performs best with both preprocessing techniques and that Artchounin's model is outperforming Shadiq's technique.

Table 4.5: Table over CNN experiment results part 1.

ID	Dataset	#Samples	Accuracy	Precision	Recall	F_1-score	#Evaluate	Classes
Shadiq								
1	B	655	0.283969	0.33	0.28	0.28	655	18
2	B	655	0.174046	0.19	0.17	0.16	655	18
3	**A**	**655**	**0.360305**	**0.40**	**0.36**	**0.37**	**655**	**18**
4	A	655	0.326718	0.36	0.33	0.32	655	18
5	A	655	0.337405	0.37	0.34	0.33	655	18
Artchounin								
1	C	863	0.533024	0.59	0.53	0.53	863	18
2	C	863	0.338355	0.69	0.34	0.36	863	18
3	C	863	0.622248	0.68	0.62	0.61	863	18
4	D	863	0.600232	0.66	0.60	0.61	863	18
5	D	863	0.584009	0.61	0.58	0.58	863	18
6	**D**	**863**	**0.633835**	**0.69**	**0.63**	**0.66**	**863**	**18**
Stratified								
1	E	1284	0.653201	0.65	0.65	0.64	1312	16
2	**E**	**1284**	**0.711128**	**0.71**	**0.69**	**0.71**	**1312**	**16**
Overlap								
1	F	800	0.414085	0.42	0.41	0.40	710	18
2	F	800	0.371831	0.38	0.37	0.36	710	18
3	**G**	**800**	**0.440845**	**0.44**	**0.44**	**0.44**	**710**	**18**
4	G	800	0.411268	0.51	0.41	0.43	710	18

Table 4.6: Table over CNN experiments results part 2.

ID	Batch Size	Dev. Ratio	Word Embeddings	Embedding Dim.	Filter Sizes	No. Filters
Shadiq						
1	128	0.1	SKIP1	200	3,4,5	128
2	128	0.1	-	200	3,4,5	128
3	**256**	**0.1**	**SKIP2**	**300**	**3,4,5**	**128**
4	128	0.1	CBOW1	300	3,4,5	128
5	128	0.1	CBOW2	300	3,4,5	128
Artchounin						
1	64	0.1	SKIP2	300	3,4,5	128
2	64	0.1	-	300	3,4,5	128
3	64	0.1	SKIP3	300	3,4,5	128
4	64	0.1	SKIP5	300	3,4,5	128
5	64	0.1	CBOW3	300	3,4,5	128
6	**64**	**0.1**	**SKIP4**	**300**	**3,4,5**	**128**
Stratified						
1	64	0.1	CBOW4	300	3,4,5	128
2	**64**	**0.1**	**SKIP6**	**300**	**3,4,5**	**128**
Overlap						
1	128	0.1	SKIP7	300	3,4,5	128
2	128	0.1	CBOW5	300	3,4,5	128
3	**128**	**0.1**	**SKIP8**	**300**	**3,4,5**	**128**
4	128	0.1	CBOW6	300	3,4,5	128

Table 4.7: Table over CNN experiments results part 3.

ID	Dropout	l_2 Reg. Lambda	Epochs	Vocabulary Size
Shadiq				
1	0.5	0	100	7867
2	0.5	0	100	7867
3	**0.5**	**0**	**100**	**7867**
4	0.5	0.15	100	7867
5	0.5	0	100	7867
Artchounin				
1	0.5	0.15	100	18 928
2	0.5	0.15	100	18 928
3	0.5	0.3	100	18 928
4	0.5	0.3	100	18 928
5	0.5	0.3	100	18 928
6	**0.5**	**0.3**	**100**	**18 928**
Stratified				
1	0.5	0.3	100	32 914
2	**0.5**	**0.3**	**100**	**32 914**
Overlap				
1	0.5	0.3	100	7705
2	0.5	0.3	100	7705
3	**0.5**	**0.3**	**100**	**23 569**
4	0.5	0.3	100	23 569

4.4.2 Generalization and Loss

A plot over train and development sets from two different CNN models (ID 4 and ID 5 from Artchounin in Table 4.5) can be seen in Figure 4.6. The models ran a different amount of steps but the plot shows that the training (teal) and corresponding development set (orange) are closer together, compared to the other models training (blue) and development (purple) lines, meaning not only does the model have a higher accuracy but it is more adapted to general input, thus acquiring a better prediction rate. The development data of the curves have a prediction accuracy of 77.11% for Artchounin models with ID 4 and value 74.31% ID 5. On unseen TRs the accuracy was 60.02% compared to 58.40% (listed in Table 4.5), which also confirms that the model is more general than the other.

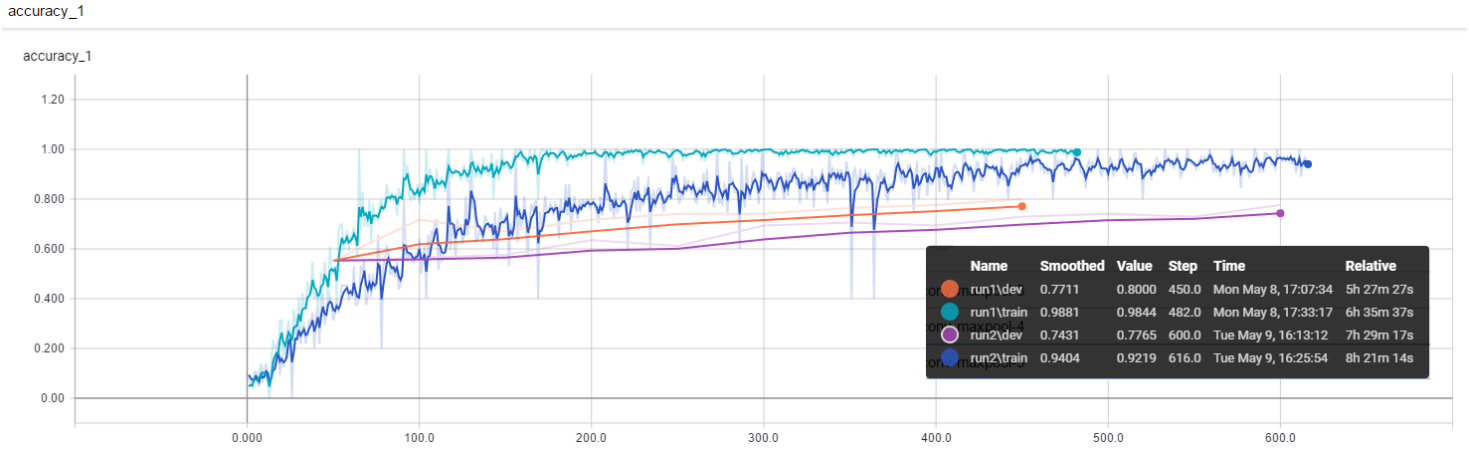


Figure 4.6: A plot showing the difference between generalization and accuracy of the models with ID 4 and ID 5 from Artchounin in Table 4.5.

The loss plot from models with ID 4 and ID 5 from Artchounin are plotted in Figure 4.7. The plot shows how the model with ID 4 (teal and orange) converges *run1* is smoother and converges better, meaning it adapts better and performs better than the model with ID 5 (blue and purple). The plot also gives an indication that the Artchounin models with ID 4 is more general and outperforms ID 5.

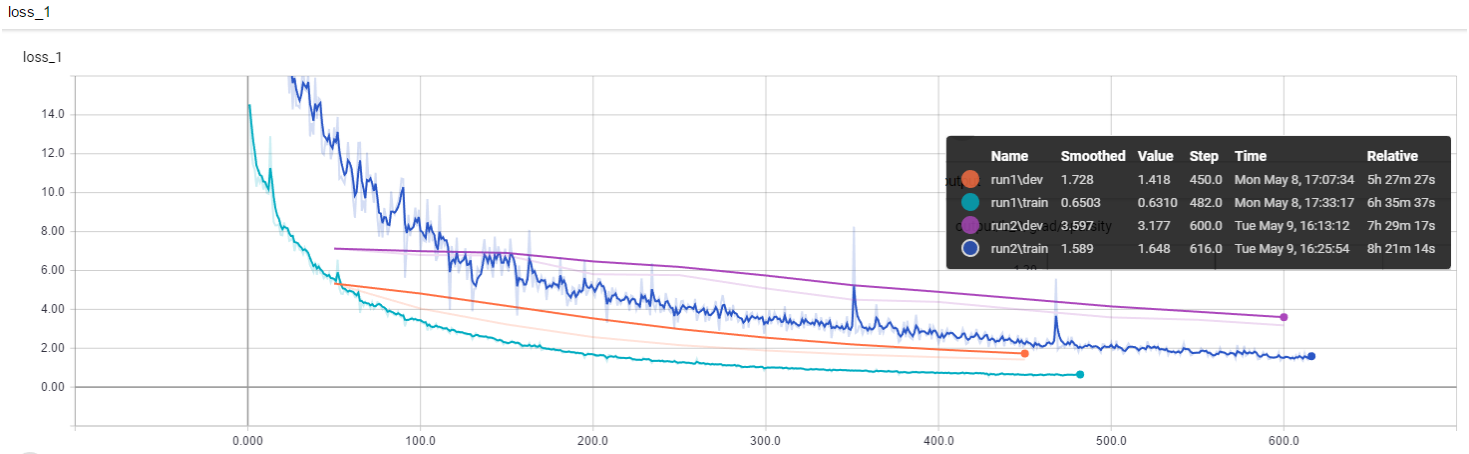


Figure 4.7: A plot showing the difference between loss and convergence of the Artchounin models with ID 4 and ID 5 from Table 4.5.

The plot in Figure 4.8 shows an example of a model which is overfitting its TR training data. As shown is that the development curve is on a steady rise until about 400 steps in, where it instead begins to fall (while the training curve still continues to rise). This gives indication that the model has become too tailored to its own training data and thus becomes less general to other input.

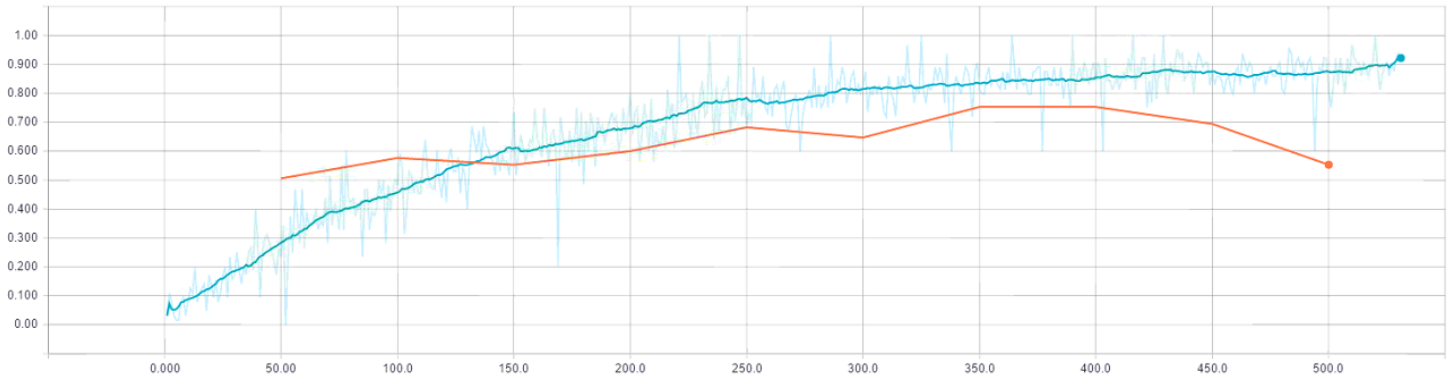


Figure 4.8: A plot showing how a model overfits its training data. The model starts to become too adapted to the training data and thus performs worse on development data.

From Table 4.5 the Artchounin model with ID 6 performs best accuracy in that subgroup. In Table 4.8 a summary over each classification performed by the model when evaluating on 863 TRs is shown. The table lists the individual metrics for each class, allowing for finding where the model performs better or worse. In addition to the summary, a detailed classification is depicted in a matrix called a *confusion matrix*, which is shown in Table 4.9. The confusion matrix depicts the exact classification of each TR for each class, giving a solid basis to reason for which classes the model e.g. has strong and weak behavior. For example in class 13, 18 TRs from class 9 are incorrectly predicted to be class 13, which is almost half of the samples of TRs for that class. This is due to the fact that class 9 and class 13 share closely related TRs. These kinds of trends together with the individual metric scores acts as a good basis when analyzing and developing the model.

Table 4.8: Evaluation results of the Artchounin model with ID 5 from Table 4.5, achieving one of the best performances of the conducted experiments.

Class	**Precision**	**Recall**	F_1**-score**	**Support**
1.	0.67	0.08	0.14	51
2.	0.61	0.82	0.70	51
3.	0.82	0.65	0.73	51
4.	0.92	0.92	0.92	51
5.	0.85	0.36	0.51	47
6.	0.65	0.88	0.75	51
7.	0.60	0.88	0.71	51
8.	0.89	0.98	0.93	51
9.	0.31	0.69	0.43	51
10.	0.56	0.45	0.50	51
11.	0.84	0.71	0.77	51
12.	0.62	0.31	0.42	51
13.	0.78	0.78	0.78	51
14.	1.00	1.00	1.00	1
15.	0.89	0.96	0.92	51
16.	0.35	0.55	0.43	51
17.	0.95	0.37	0.54	51
18.	0.71	0.60	0.65	51
avg / total	0.69	0.63	0.66	863

Table 4.9: A Confusion matrix of model with Artchounin model with ID 5. This shows a detailed overview of the predicted classification for each class when evaluating on 863 TRs. The X-axis shows the true class for TRs, while the Y-axis shows how many TRs is predicted to be that class. For example the top line shows that 47 TRs from 8 different classes are predicted to be class 1. The ideal behavior would be a diagonal line through the matrix, meaning that the model classifies every unseen TR correctly.

Class	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.	15.	16.	17.	18.
1.	4	0	0	0	0	1	1	0	28	0	0	2	5	0	0	8	0	2
2.	0	42	0	0	0	1	0	0	0	0	0	0	0	0	0	8	0	0
3.	0	2	33	0	0	2	0	0	3	3	3	0	2	0	0	2	0	1
4.	0	0	0	47	0	1	3	0	0	0	0	0	0	0	0	0	0	0
5.	0	2	1	3	17	9	1	5	0	6	0	0	0	0	0	3	0	0
6.	0	1	0	0	3	45	0	1	0	0	0	0	0	0	0	1	0	0
7.	0	3	0	0	0	0	45	0	2	0	0	0	1	0	0	0	0	0
8.	0	1	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0
9.	2	3	0	0	0	0	0	0	35	0	0	0	4	0	0	5	0	2
10.	0	1	2	0	0	3	4	0	1	23	2	4	1	0	1	9	0	0
11.	0	0	1	0	0	3	7	0	0	1	36	0	0	0	0	2	1	0
12.	0	8	1	0	0	0	1	0	5	2	0	16	10	0	0	8	0	0
13.	0	0	0	0	0	0	0	0	18	1	0	1	27	0	0	4	0	0
14.	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
15.	0	0	0	1	0	0	1	0	0	0	0	0	0	0	49	0	0	0
16.	0	4	0	0	0	3	0	0	4	4	1	1	4	0	0	28	0	2
17.	0	2	2	0	0	1	11	0	4	1	1	0	0	0	5	0	19	5
18	0	0	0	0	0	0	1	0	12	0	0	2	3	0	0	2	0	30

5 Discussion

In this chapter, the method of how the study was carried out is discussed together with the results which have been produced.

5.1 Method

This section discusses the various parts of the Method. It discusses how the chosen preprocessing, approach and experiments has been working throughout this work.

5.1.1 Preprocessing

The process of forming a preprocessing technique was initially planned to be done by the authors. As the discussion went on with the authors of parallel works, we found it intriguing and interesting to investigate the differences between two preprocessing techniques from Shadiq and Artchounin which have significant differences, which both were likely to provide interesting results. One of the most important realizations when it comes to preprocessing has been understanding the importance of preprocessing of data when developing a classifying model. One might possess big data volumes to work with, but unless it is refined, there is an imminent chance that it will result in low performance when developing a classifying model.

The use of pretrained word embeddings has improved results drastically compared to not using them. For example, the accuracy more than doubled between Shadiq experiment with ID 2 and 3 from Table 4.5 where the major difference was the use of preloaded word embeddings. Essentially, the preloaded word embeddings give the model a head start of words relation to each other which effect is noticeable. We planned to use two different models to create the preloaded word embeddings but decided to only use Word2vec. Instead, we decided to compare two different ways to represent the word embeddings (Skip-gram and CBOW) after having read Mikolov et al. [34] work which discusses that Skip-gram and CBOW can behave differently for different volumes of text corpora. We also experimented with a different number of dimensions for the word embeddings, but any difference from the dimension range 200 and 300 did not produce any major changes to the results. The recommendation from the Word2vec project [1] when using Word2vec suggests that 300 is a good all-around choice, which is the number we chose to keep.

[1] https://code.google.com/archive/p/word2vec/

The use of word embeddings is concluded to be beneficial from our experiments, but when working and developing models with word embeddings we encountered one issue we believe is a diffuse area. It is recommended to train word embeddings on large corpora of natural languages, like the document database from Wikipedia or large news sets [2]. Then the embeddings are used for classification of other documents, e.g. tweets. When providing a classifying model with these vast amounts of text, the model is likely to have knowledge of every word that can possibly occur in unseen data. However, this has not been the case for us. If we train and preload word embeddings based on the TRs in training data and the data we don't use for unseen data, this means that we can encounter words the model has not seen before, for example, a specific product ID. The CNN handles unseen words accordingly by translating them into word embeddings, but the issue is whether or not it is considered biased by training a model with words that can occur in TRs, even those used for evaluation. Note that in our work, we did not use word embeddings from unseen TR data, but there is still a debatable issue. We have not found sources whether it is considered biased or not.

5.1.2 Approach

In the beginning of the project, the intentions were only to have a frugal hyperparameter tuning when working with the CNN models since we were under the impression that this would have a small impact on the end result metrics which has been investigated, as mentioned by Yoon Kim [1]. However, small efforts in the configurations turned out to produce better results which motivated the pursuit of further experimentation with tuning.

The CNN architecture is based on the paper by Yoon Kim [1]. An implementation by Denny Britz (Google Brain team) for NLP became our starting point. It utilizes TensorFlow and is also based on Yoon Kim's work, which made it a natural starting point. By using an existing implementation which utilizes established libraries and functions, we increased our chances of achieving better results, rather than being stuck on implementation issues due to little prior experience with TensorFlow throughout the work. Today there exists many frameworks when working with CNNs and there are more high-level frameworks accessible, like Keras [3], which could be more suitable for people with little prior experience when working with neural networks. TensorFlow has the potential to create complex implementations and TensorBoard is a great visualizing tool when working with neural networks, but if a simpler architecture is to be developed, a more abstract framework is recommended by the authors.

When examining the graphs from CNN models with the help of TensorBoard, occasional signs of overfitting were present, similar to the graph in Figure 4.8. Taking measures like regularization helped to combat the overfitting and it made a noticeable difference, i.e. the investigated measures increased. This was also the reason to why we always kept dropout throughout the experiments, rather than not testing it as we probably would overfit even more. These kinds of consequences were hard to foresee and had to be adjusted accordingly as the development went along. This created a kind of timeline throughout the experiments, where the models were continuously improved and configured, which is why in the most cases the latter experiments yield better results.

Initially only the Shadiq and Artchounin experiments were planned to be conducted in terms of the CNN. Together with feedback from relevant parties, it was deemed interesting to extend the experiments as well as varying them. This is when the experiments called Stratified and Overlap were performed. The Stratified experiment aimed to give a basis to see how a CNN model behaved on random samples keeping the original skewed distribution of TRs. These two models were performed with the Artchounin's preprocessing technique, as it had shown most promising results for the TRs. It was run on 16 classes, meaning two classes were excluded. The reasons for this was that one class acted as a collector for TRs

[2] https://code.google.com/archive/p/word2vec/
[3] https://keras.io/

which did not belong to any of the other 17 classes, meaning that the samples are inconsistent i.e. the TRs for this class are most likely problematic to classify. The other class had very few samples compared to the other classes and thus adding another class would probably do more harm to the model, rather than giving it a solid basis with training samples from that class. The Overlap experiments were conducted in order to see how our developed models would perform on the same identical dataset, since in order to be able to draw any final and reliable conclusions whether one preprocessing technique is better than the other, it must be investigated how they behave on the exact same set of TR training samples. Comparing the sets on the Shadiq and Artchounin could potentially be limited by having a biased set of sample TRs when performing the experiments, as to why the Overlap experiment was conducted. These three types of experiment are related to each other but can also be seen as stand alone experiments, which aims to open up interesting areas for future research, whether it is related to configurations of the models, the preprocessing techniques or the distribution of samples.

The use of Multinomial Naive Bayes and linear kernel SVM as choices for traditional classifiers are justified by their frequent occurrence in similar work addressing the task of text categorization [3] [14]. The goal was to establish a simple baseline that reflected these prior works. The results of the methods presented in this thesis could then be compared to this baseline. The same set of traditional classifiers were included in both the flat classification experiments on 18 classes, as well as the hierarchical experiments with 18 classes in the first level and 150+ classes in the second level.

5.1.3 Evaluation

Evaluation during training of the CNN is done with a 90/10 split train/development set. After training, the network is evaluated by showing it an equal amount of unseen samples as it was trained on. Ten-fold cross-validation is performed when evaluating the traditional classifiers since this is an established method when evaluating these types of classifiers as well as making sure the result is not biased by for example overfitting [63, 64]. These two techniques has been evaluated differently, as to why it is important that the chosen method is limited to do an identical comparison of the metrics. The investigated metrics Precision, Recall and F_1-score have been looked into detail as shown in Table 4.8, but they have also been listed weighted, meaning that the number of evaluating samples for a class have been taken into consideration i.e. they are averaged. However, there exist other techniques for assessing metrics when the there is an uneven distribution of class samples such as micro and macro-averaging [8].

There was no attempt at trying to apply a hierarchical classification approach together with the CNN. We are yet to find any relevant sources on the subject. CNNs for Natural Language Processing text classification is also a relatively new technique [54, 1, 53, 2]. A more intricate classification model could be achieved if the relationships between the classes were further modeled. For example, the classes could be represented in such a way that some classes are closer than others i.e. the classes share more similarities. This means that a more detailed and complex tree hierarchy could potentially benefit more from this method. This would also open up to adjust penalization for classifying to certain classes, i.e. it should take more certainty for a model to classify to a rare class.

The Multinomial Naive Bayes classifier was used in both the flat and hierarchical classification contexts. By compiling results for the two contexts with both standard and hierarchical F_1-score, we can draw conclusions regarding the viability of the hierarchical classifier. As Table 4.4 reveals, it is outperformed by flat classification when using either evaluation metric. When the hierarchical metric is used and the hierarchical classifier is allowed to be partially correct, both contexts score fairly similar. When the standard metric is used, the hierarchical classifier fares much worse than its flat classification counterpart. Since the classifier was implemented by the authors of this thesis themselves, using components in Python and Scikit-

Learn rather than a preexisting implementation, it could be due to inexperience working with the framework. Another reason might be that it is simply not fit for the problem domain and source data. The class hierarchy turned out not to be very complex, and adding the additional complexity of a hierarchical classification approach might lead to worse results in such a situation. This leaves us to believe it is more beneficial to use flat classification for classifying to the 150+ classes. This is further supported by the fact that the linear SVM performed better than expected in classifying to a large number of classes. In addition to better performance, it would also allow the classifier to be trained with many more samples since the hierarchical classifier consumed large amounts of memory.

5.2 Results

This section will discuss the various parts of the Results. It will discuss the achieved results and what this encompasses.

5.2.1 Experiments

Investigating the Table 4.5,4.6 and 4.7 the different results of the CNN experiments can be found. As we are yet to find similar references of this type of work with CNN, we are relying on discussions with supervisors and experts (persons working with TRs). The verdict is that achieving up over 70% in some experiments are helpful for the organization and even though the model should not be fully reliable, it would most likely be a helpful tool in production.

The results point to that the preprocessing technique performing the better of the two was the one by Artchounin. It indicates that a preprocessing technique which takes machine generated code into consideration is a beneficial characteristic. These indications are confirmed by the authors of this work, as it was empirically tough for us and experts to classify only the text after the preprocessing of Shadiq. This means not only that Artchounin's preprocessing technique performs better when automating it, but a suggestion would be that better standards for writing detailed TRs are enforced if the current TR system should remain. Otherwise, machine generated code is desired. One reason for this is that machine generated code is consistent and gives us detailed and information about issues when troubleshooting. Issues which are recurring will likely be formatted similarly to historical issues of the same type, which is beneficial for an automated classifier.

As the work has been conducted some classes has had a sparse amount of TRs, meaning that the class distribution became skewed. In order to make stronger predictions for an automated system a more even sample distribution would be required. In the case of classifying 18 classes with a total of over 66 000 TRs, it will be more harmful of having a class with only around 100 TRs (less than 0.001%) to train the model on. Comparing the Stratified experiments with the Artchounin experiments confirms this, as the experiments are performed similarly with the exception of a number of training examples and the removal of two deemed problematic classes.

Either penalization for weights should be implemented, or another question to investigate is that if the current representation of TRs used gives the whole image of the real-world representation. This is related to the issue which is likely to occur is when products related to TRs change over time, e.g. new releases generating a surge in a specific kind of TRs, over representing a certain class. If these new kinds of TRs have been the one which had few training samples the end result could be harmful to the automated system, even though the system could always be retrained at any point in time. Another solution, other than retraining the model, is to continuously improve the model, where it is regularly fed with incoming TRs, meaning that the model would successively adapt to TRs adapt through time making CNN a strong candidate for this classifying task.

The Multinomial Naive Bayes classifier seems particularly inclined to classify reports to classes that dominate the training set. This can turn out to be an issue if the model were to be

used in a production environment, since incoming trouble reports may need to be classified to other organizations for longer periods depending on issues from new product releases. The model should ideally classify reports based on their content, not the class distribution in the training data.

The best performing model in terms of classification accuracy was the linear kernel SVM. SVMs are known in the field of machine learning as state-of-the-art classifiers and are well suited for text classification [13]. Combined with the fact that the base classifiers in a flat classification context were the only ones that could be trained using the full data set (due to hardware limitations and implementation issues in other experiments), this is perhaps not surprising. More data should reasonably allow us to train a stronger predictive model, as long as side effects such as overfitting are handled accordingly.

The hierarchical classifier that was implemented produced fairly poor results. The trained model was not able to consistently predict the correct path for given test samples which led to a low hierarchical F-score. The implementation suffered from memory issues, which limited the number of samples that could have been used to train the classifiers.

5.2.2 Limitations

Limited computational resources and unexpected amount of required memory made training the CNN model arduous. Only a fraction of the total TR dataset could be used during training to prevent the implementation from running out of memory or reach unreasonably long training durations. Despite this, the model performs surprisingly well and achieves better accuracy than the Multinomial Naive Bayes classifier that has been trained with considerably more data. The CNN model would likely achieve better results if it was possible to use more of the data samples for training as well as further optimizing the hyperparameters. Due to the settings of the hyperparameters (e.g. dimensionality of the vectors used) and the size of the TR dataset, only a CPU was feasible due to the memory constraints, even though a GPU would've achieved a speed-up of the experiments. Another reason for the hardware limitations is the problem with using CNNs and word embeddings together. This leads to the production of computational heavy matrices such as the embedding layer where the word embeddings act as a lookup table with weights corresponding to each word stored in the RAM memory. The matrix is depicted as *vocabulary size* times *embedding dimension*, meaning we have a 139 835 $\times$ 300 matrix to store in memory with our largest vocabulary and doing calculations will thus be very demanding. A solution would be increasing the computational power or implementing another way of how the largest matrices are stored, but this solving this problem was expected to exceed the time frame for the work.

5.3 Source Criticism

This work has been greatly influenced by the work of Yoon Kim [1]. One of the reasons is that a relatively simple architecture with CNNs for making a binary classification, like the work performed by Kim, performs well compared to other work performing similar tasks [65, 66, 67], was a good starting point for extending to a multi-class classification as well. Another reason is that this is the work by Kim [1] is currently one of the most cited papers for CNN with text classification, together with the consideration that CNN for text classification is a relatively new field as well. However, it is important to keep in mind that Kim's architecture and the solution might not be the best possible for this task, even though it shows that it seems applicable.

5.4 Ethical Aspects

When putting the work in a wider context, the realization of automatic bug- and issue assignment has been sought after for many years, which is indicated by its extensive research [3]

[47] [68]. The process of manually screening vast amounts of reports is costly for an organization, thus an automated one would yield a lot of benefits.

Solely relying on a completely automated system is not feasible. This report shows that assignment of bug reports is a complex problem covering many aspects to keep in consideration. While a classification model's accuracy can be high, it never reaches the gold standard in practice. Rather, it is instead suggested to be used as a tool when the regular screening process takes place. Even this would not be fully automated, it still has the potential to significantly reduce the time required to assign the report to the responsible product organization. In the case of Ericsson, it has been estimated that an hour of work would be saved every day for personnel involved in the trouble report screening process, should such a tool be utilized. The reduced workload could lead to increased well-being for these people, as well as saving company funds required to have experts manually analyze contents of trouble reports.

On the other hand, there is the possibility that a responsibility assignment system could be biased, in the sense that the system would recommend routing reports to certain product organizations more frequently than others. Such a situation could be caused by an imbalanced class representation when training the model, or choice of the learning algorithm chosen. This would perhaps give rise to skewed workload for developers, leaving some over-encumbered and stressed.

6 Conclusion

This thesis presents a study of how the task of text classification in a TR/bug report context can be performed using a CNN, as well as an approach using hierarchical classification. These methods are compared to a baseline method using standard classifiers in a flat classification context.

6.1 Project and Research Questions

Regarding Research Question 1, using the two preprocessing techniques has shown that keeping machine generated text when extracting features is beneficial through several types of experiments. The conservation of machine generated code such as error codes, product numbers etc. leads to an improvement of information for a classification model, which in turn leads to an increased value in the classification metrics investigated.

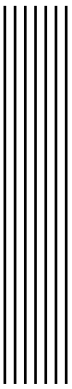
When it comes to Research Question 2, the results did not show superior results when comparing the CNN approach or the hierarchical approach to flat classification with traditional classifiers. The CNN model has comparable performance to the Multinomial Naive Bayes classifier, with only a fraction of the training data at its disposal. On the other hand, the CNN model has another aspect of importance and that is the behavior over time. A CNN model can be continuously trained, meaning it will be flexible towards facing new types of TR if the textual content change over time. It does not require re-training of all previous historical TRs, but it can yet add new training examples as they occur. This trait gives the CNN a desired behavior which the more traditional classifiers investigated lack. However, The SVM classifier performs the best out of all, which is much in line with results from similar work in the field [3] [13].

6.2 Future Work

The trouble reports sometimes include attachments in the form of log data. This data is generated output from company equipment and differs depending on which errors are encountered. Including such data when constructing features, in addition to the heading and observation text of a trouble report, could potentially lead to better classification accuracy. This follows from the intuition that since the output is machine generated, it would be more consistent in its behavior in regards to errors than a human would.

In addition to this, one could consider incorporating penalization into the model to counter the unbalanced dataset. This would make it more costly for the model to predict the more common classes. One could also develop a more detailed class hierarchy, in order to depict the real world even further.

Finally, the CNN classification model could be improved if more of the training data was able to be used. Due to the unforeseen memory requirements needed for training the model, only a small fraction of the complete dataset was used. Securing more resources would be key for making this possible using the same tools as in the experiments. Alternatively, other tools and frameworks that are known to consume less memory could be used in experiments instead. The CNN model also has room for improvement in regards to the tuning of the hyperparameters and the architecture. There are a wide range of different architectures, convolutional layers, hyperparameters and word embedding techniques which could be tried and optimized for the task at hand, surely boosting the performance of the model.



Bibliography

[1] Yoon Kim. "Convolutional neural networks for sentence classification". In: *arXiv preprint arXiv:1408.5882* (2014).

[2] Xiang Zhang, Junbo Zhao, and Yann LeCun. "Character-level convolutional networks for text classification". In: *Advances in neural information processing systems*. 2015, pp. 649–657.

[3] Leif Jonsson, Markus Borg, David Broman, Kristian Sandahl, Sigrid Eldh, and Per Runeson. "Automated Bug Assignment: Ensemble-based Machine Learning in Large Scale Industrial Contexts". In: *Empirical Softw. Engg.* 21.4 (Aug. 2016), pp. 1533–1578. ISSN: 1382-3256. DOI: 10.1007/s10664-015-9401-9.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN: 0387310738.

[5] Sotiris Kotsiantis. "Supervised machine learning: A review of classification techniques". In: *Informatica* 31 (2007), pp. 249–268.

[6] Melvin Earl Maron. "Automatic indexing: an experimental inquiry". In: *Journal of the ACM (JACM)* 8.3 (1961), pp. 404–417.

[7] Andrew McCallum, Kamal Nigam, et al. "A comparison of event models for naive bayes text classification". In: *AAAI-98 workshop on learning for text categorization*. Vol. 752. Citeseer. 1998, pp. 41–48.

[8] Fabrizio Sebastiani. "Machine learning in automated text categorization". In: *ACM computing surveys (CSUR)* 34.1 (2002), pp. 1–47.

[9] Chi Jin and Liwei Wang. "Dimensionality dependent PAC-Bayes margin bound". In: *Advances in Neural Information Processing Systems*. 2012, pp. 1034–1042.

[10] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. "Top 10 algorithms in data mining". In: *Knowledge and information systems* 14.1 (2008), pp. 1–37.

[11] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. "Thumbs up?: sentiment classification using machine learning techniques". In: *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*. Association for Computational Linguistics. 2002, pp. 79–86.

[12] Thorsten Joachims. "Text categorization with support vector machines: Learning with many relevant features". In: *Machine learning: ECML-98* (1998), pp. 137–142.

[13] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*. Vol. 1. 1. Cambridge university press Cambridge, 2008.

[14] Susan Dumais and Hao Chen. "Hierarchical classification of Web content". In: *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2000, pp. 256–263.

[15] Julio Hernández, L Enrique Sucar, and Eduardo F Morales. "Multidimensional hierarchical classification". In: *Expert Systems with Applications* 41.17 (2014), pp. 7671–7677.

[16] Aixin Sun and Ee-Peng Lim. "Hierarchical text classification and evaluation". In: *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*. IEEE. 2001, pp. 521–528.

[17] Eduardo P Costa, Ana C Lorena, André CPLF Carvalho, Alex A Freitas, and Nicholas Holden. "Comparing several approaches for hierarchical classification of proteins with decision trees". In: *Brazilian Symposium on Bioinformatics*. Springer. 2007, pp. 126–137.

[18] Simon Haykin and Neural Network. "A comprehensive foundation". In: *Neural Networks* 2.2004 (2004), p. 41.

[19] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[20] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 437–478.

[21] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[22] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks." In: *Aistats*. Vol. 15. 106. 2011, p. 275.

[23] Jianchao Yang, Kai Yu, Yihong Gong, and Thomas Huang. "Linear spatial pyramid matching using sparse coding for image classification". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 1794–1801.

[24] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[25] Ye Zhang and Byron Wallace. "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification". In: *arXiv preprint arXiv:1510.03820* (2015).

[26] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C Courville, and Yoshua Bengio. "Maxout Networks." In: *ICML (3)* 28 (2013), pp. 1319–1327.

[27] Matthew D. Zeiler and Rob Fergus. *Stochastic Pooling for Regularization of Deep Convolutional Neural Networks. ArXiv e-prints, January 2013. Ping Zhong and Runsheng Wang. Using combination of statistical models and multilevel structural information for detecting urban areas from a single*.

[28] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. DTIC Document, 1985.

[29] M Ikonomakis, S Kotsiantis, and V Tampakas. "Text classification using machine learning techniques." In: *WSEAS transactions on computers* 4.8 (2005), pp. 966–974.

[30] Bernard J Jansen and Soo Young Rieh. "The seventeen theoretical constructs of information searching and information retrieval". In: *Journal of the American Society for Information Science and Technology* 61.8 (2010), pp. 1517–1534.

[31] Juan Ramos et al. "Using tf-idf to determine word relevance in document queries". In: *Proceedings of the first instructional conference on machine learning*. 2003.

[32] Anisha Mariam Thomas and M.G. Resmipriya. "An Efficient Text Classification Scheme Using Clustering". In: *Procedia Technology* 24 (2016). International Conference on Emerging Trends in Engineering, Science and Technology (ICETEST - 2015), pp. 1220–1225. ISSN: 2212-0173. DOI: http://dx.doi.org/10.1016/j.protcy.2016.05.095.

[33] Gerard Salton and Christopher Buckley. "Term-weighting approaches in automatic text retrieval". In: *Information processing & management* 24.5 (1988), pp. 513–523.

[34] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[35] Andriy Mnih and Geoffrey E Hinton. "A scalable hierarchical distributed language model". In: *Advances in neural information processing systems*. 2009, pp. 1081–1088.

[36] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[37] Frederic Morin and Yoshua Bengio. "Hierarchical Probabilistic Neural Network Language Model." In: *Aistats*. Vol. 5. Citeseer. 2005, pp. 246–252.

[38] Ian H. Witten, Eibe Frank, and Mark A. Hall. "Chapter 5 - Credibility: Evaluating What's Been Learned". In: *Data Mining: Practical Machine Learning Tools and Techniques (Third Edition)*. Ed. by Ian H. Witten, Eibe Frank, and Mark A. Hall. Third Edition. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, 2011, pp. 147–187.

[39] David Martin Powers. "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation". In: (2011).

[40] Carlos N Silla Jr and Alex A Freitas. "A survey of hierarchical classification across different application domains". In: *Data Mining and Knowledge Discovery* 22.1-2 (2011), pp. 31–72.

[41] Svetlana Kiritchenko, Stan Matwin, A Fazel Famili, et al. "Functional annotation of genes using hierarchical text categorization". In: *Proc. of the ACL Workshop on Linking Biological Literature, Ontologies and Databases: Mining Biological Semantics*. 2005.

[42] Aris Kosmopoulos, Ioannis Partalas, Eric Gaussier, Georgios Paliouras, and Ion Androutsopoulos. "Evaluation measures for hierarchical classification: a unified view and novel approaches". In: *Data Mining and Knowledge Discovery* 29.3 (2015), pp. 820–865.

[43] T. Joachims. "Text categorization with support vector machines: learning with many relevant features". In: *Proceedings of ECML-98, 10th European Conference on Machine Learning*. 1998, pp. 137–142.

[44] Shaowei Wang and David Lo. "Version history, similar report, and structure: Putting them together for improved bug localization". In: *Proceedings of the 22nd International Conference on Program Comprehension*. ACM. 2014, pp. 53–63.

[45] Shaowei Wang, David Lo, and Julia Lawall. "Compositional vector space models for improved bug localization". In: *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. IEEE. 2014, pp. 171–180.

[46] Shaowei Wang, David Lo, Zhenchang Xing, and Lingxiao Jiang. "Concern localization using information retrieval: An empirical study on linux kernel". In: *Reverse Engineering (WCRE), 2011 18th Working Conference on*. IEEE. 2011, pp. 92–96.

[47] Leif Jonsson, David Broman, Måns Magnusson, Kristian Sandahl, Mattias Villani, and Sigrid Eldh. "Automatic Localization of Bugs to Faulty Components in Large Scale Software Systems using Bayesian Classification". In: *Software Quality, Reliability and Security (QRS), 2016 IEEE International Conference on*. IEEE. 2016, pp. 423–430.

[48] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[49] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks". In: *CoRR* abs/1312.6229 (2013).

[50] Aravindh Mahendran and Andrea Vedaldi. "Visualizing Deep Convolutional Neural Networks Using Natural Pre-images." In: *International Journal of Computer Vision* 120.3 (2016), pp. 233–255. ISSN: 09205691.

[51] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going Deeper with Convolutions". In: *Computer Vision and Pattern Recognition (CVPR)*. 2015. URL: http://arxiv.org/abs/1409.4842.

[52] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. "Natural language processing (almost) from scratch". In: *Journal of Machine Learning Research* 12.Aug (2011), pp. 2493–2537.

[53] Cicero Nogueira Dos Santos and Maira Gatti. "Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts." In: *COLING*. 2014, pp. 69–78.

[54] Phil Blunsom, Edward Grefenstette, and Nal Kalchbrenner. "A convolutional neural network for modelling sentences". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics. 2014.

[55] Alec Go, Richa Bhayani, and Lei Huang. "Twitter sentiment classification using distant supervision". In: *CS224N Project Report, Stanford* 1.12 (2009).

[56] Rie Johnson and Tong Zhang. "Effective Use of Word Order for Text Categorization with Convolutional Neural Networks". In: *CoRR* abs/1412.1058 (2014). URL: http://arxiv.org/abs/1412.1058.

[57] Rie Johnson and Tong Zhang. "Semi-supervised convolutional neural networks for text categorization via region embedding". In: *Advances in neural information processing systems*. 2015, pp. 919–927.

[58] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. "Scikit-learn: Machine learning in Python". In: *Journal of Machine Learning Research* 12.Oct (2011), pp. 2825–2830.

[59] Google Brain Team. *TensorFlow - An open-source software library for Machine Intelligence*. URL: https://www.tensorflow.org/ (visited on 02/21/2017).

[60] Christian Kohlschutter, Peter Fankhauser, and Wolfgang Nejdl. "Boilerplate Detection using Shallow Text Features". In: *Proceeding of 3rd ACM International Conference on Web Search and Data Mining*. Association for Computational Linguistics. New York, USA, 2010, pp. 441–450.

[61] Ammar Shadiq. "Automatic Bug Report Assignment using Multilevel Recurrent Neural Network." 2017.

[62] Daniel Artchounin. "Tuning of machine learning algorithms for automatic bug assignment". 2017.

[63] Ron Kohavi et al. "A study of cross-validation and bootstrap for accuracy estimation and model selection". In: *Ijcai*. Vol. 14. 2. Stanford, CA. 1995, pp. 1137–1145.

[64] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. "A practical guide to support vector classification". In: (2003).

[65] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, Christopher Potts, et al. "Recursive deep models for semantic compositionality over a sentiment treebank". In: *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*. Vol. 1631. 2013, p. 1642.

[66] Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. "Semi-supervised recursive autoencoders for predicting sentiment distributions". In: *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics. 2011, pp. 151–161.

[67] Li Dong, Furu Wei, Shujie Liu, Ming Zhou, and Ke Xu. "A statistical parsing framework for sentiment classification". In: *Computational Linguistics* (2015).

[68] John Anvik, Lyndon Hiew, and Gail C Murphy. "Who should fix this bug?" In: *Proceedings of the 28th international conference on Software engineering*. ACM. 2006, pp. 361–370.