Bachelor Thesis

Civilingenjör i datateknik - 300hp

Applying machine learning algorithms to multi-label text classification on GitHub issues

Computer science and engineering, 15 credits

Halmstad 2020-09-13

Daniel Artmann

HALMSTAD UNIVERSITY

Applying machine learning algorithms to multi-label text classification on GitHub issues

Bachelor's Thesis

2020

Author: Daniel Artmann

Supervisor: Taha Khan

Examiner: Pererik Andreasson

Abstract

This report compares five machine learning algorithms in their ability to categorize code repositories. The focus of expanding software projects tend to shift from developing new software to the maintenance of the projects. Maintainers can label code repositories to organize the project, but this requires manual labor and time. This report will evaluate how machine learning algorithms perform in automatically classifying code repositories. Automatic classification can aid the management process by reducing both manual labor and human errors.

GitHub provides online hosting for both private and public code repositories. In these repositories, users can open issues and assign labels to them, to keep track of bugs, enhancement, or requests. GitHub was used as a source for all data as it contains millions of open-source repositories. The focus was on the most popular labels from GitHub - both default labels and those defined by users.

This report investigated the algorithms linear regression (LR), convolutional neural network (CNN), recurrent neural network (RNN), random forest (RF), and *k*-nearest-neighbor (KNN) - in multi-label text classification. The mentioned algorithms were implemented, trained, and tested with the Keras and Scikit-learn libraries. The training sets contained around 38 thousand rows and the test set around 12 thousand rows. Cross-validation was used to measure the performance of each algorithm. The metrics used to obtain the results were precision, recall, and F1-score. The algorithms were empirically tested on a different number of output labels. In order to maximize the F1-score, different designs of the neural networks and different natural language processing (NLP) methods were evaluated. This was done to see if the algorithms could be used to efficiently organize code repositories.

CNN displayed the best scores in all experiments, but LR, RNN, and RF also showed some good results. LR, CNN, and RNN the had the highest F1-scores while RF could achieve a particularly high precision. KNN performed much worse than all other algorithms. The highest F1-score of 46.48% was achieved when using a non-sequential CNN model that used text input with stem words. The highest precision of 89.17% was achieved by RF.

It was concluded that LR, CNN, RNN, and RF were all viable in classifying labels in software-related texts, among those found in GitHub issues. KNN wasn't found to be a viable candidate for this purpose.

Sammanfattning

Den här rapporten jämför fem olika maskininlärningsalgoritmer i deras förmåga att kategorisera kodförvar. Fokuset för expanderande mjukvaruprojekt brukar skifta sig från att utveckla ny kod till att underhålla projekten. Underhållare kan sätta etiketter på kodförvar för att organisera projektet, men detta kräver manuellt arbete och tid. Denna rapport kommer utvärdera hur maskininlärningsalgoritmer presterar i att automatiskt klassifiera kodförvar. Algoritmer som automatisk klassifierar kan hjälpa administrationsprocessen genom att minska manuellt arbete och mänskliga fel.

GitHub erbjuder webhotell för både privata och publika kodförvar. I dessa förvar, kan användare öppna "issues" och sätta etiketter på dem, för att kunna hålla koll på buggar, förbättringar eller förfrågningar. GitHub användes som en källa för all data då det innehåller miljoner av kodförvar som öppen källkod. Fokus låg på de mest populära etiketterna på GitHub – både standard etikettern och de som definerats av användare.

Denna uppsats undersökte algoritmerna linear regression (LR), convolutional neural network (CNN), recurrent neural network (RNN), random forest (RF) och k-nearest-neighbor (KNN) – i multi-etikett klassifiering. De nämnda algoritmerna implementerades, tränades och testades med Keras och Scikit-learn biblioteken. Korsvalidering användes för att mäta precision, recall och F_1-poäng. Algoritmerna testades empiriskt på flera output etiketter. För att kunna maximera F_1-poäng, så provades olika designs av de neurala nätverken och provade olika metoder i naturlig språkbehandling. Detta gjordes för att se om algoritmerna kunde användas för att effiktivt kunna organisera kodförvar.

CNN visade de bästa mätvärderna i alla experiment, men LR, RNN och RF visade också bra resultat. LR, CNN och RNN hade de bästa F_1-poängen medan RF kunde uppnå en särskilt bra precision. KNN hade mycket sämre resultat än alla de andra algoritmerna. Den bästa F_1-poängen på 46,48% uppnådes när en icke-sekventiell CNN modell med text input som hade stam ord. Den högsta precisionen uppnådes på 89,17% av RF.

Slutsatsen drogs att LR, CNN, RNN och RF var alla bra kandidater för att klassifiera etiketter i mjukvaru-relaterade texter som fanns i GitHub "issues". KNN visade sig inte vara tillräckligt bra för att användas i detta syfte.

Contents

Chapter 1

Introduction

Machine learning is a field in AI that has generated a lot of interest in both research and businesses. Giant tech companies like Microsoft and Spotify use machine learning in their products to either prevent email spam or tailor music playlists to specific users. The biggest tech companies are investing heavily in AI research and acquisitions. The global AI market has experienced huge growth and the revenue from it is expected to reach 126 billion US dollars in 2025[1]. As the amount of data has increased in the internet age, so has the need to categorize and classify it.

Open-source software is a type of software where the source code has been released to the public. A big advantage of this is that anyone that is interested in a project can contribute to it. When open-source projects grows, product managers and developers find that more focus shift from developing the original idea, to maintaining the project. In a recent conference, Linus Torvalds stated that finding people to maintain the Linux kernel is really hard, even though the project doesn't lack good programmers[2]. Here, deep learning algorithms are evaluated as a classifier for an automatic tool, that would help developers to maintain code repositories.

1.1 Problem

In 2019, approximately 11.14 million new issues were opened on GitHub. Among these issues, only 17.64% had one or more labels assigned to them (see Appendix A). Applying labels to issues helps developers organize and prioritize projects. Both product managers and developers can take advantage of labels to help visualize and build a better overview of the project. They can also use the labels to filter the most important issues to better arrange their workflow. The results from this study could be used to implement tools that automate placing labels on new tickets, increasing quality, and reducing both time requirements and cognitive load. A well-performing automatic tool would hopefully increase the ratio of assigned labels when opening issues.

This study will evaluate if deep learning classifiers can effectively be used to automatically assign labels to GitHub issues. Different types of classifiers will be evaluated and compared against each other in different aspects. There is a large variety of algorithms that can be used, and many of them work very differently. Some published reports have, for instance, used tree-based, deep learning, or lazy learning, to perform multi-label classification[3].

Developers that use deep learning for text classification have to select an algorithm that is both suitable for their data set and project requirements. These developers can also decide to incorporate a few or a very large amount of unique labels. This report will examine how algorithms scale perform given different designs and a different number of output labels.

1.2 Purpose

In this study, the purpose will be to compare different classifiers and how they perform in several aspects. We will try to answer the following questions:

- Can deep learning algorithms reliably be used for multi-label classification?
- How do neural networks perform compared to other supervised learning algorithms?
- Which classifier has the best performance given a different number of output labels?
- Do non-sequential neural networks perform any better than their sequential counterparts? Do the algorithms perform better when using multiple text inputs?
- Can performance be improved by using pre-trained embedding weights?

1.3 Goals

The first goal is to collect a large amount of data that can be categorically classified. Next, the data needs to be cleaned and transformed into a suitable format. Models with text input and a categorical output will be implemented and trained. These models will be constructed with both sequential and non-sequential designs and will have to handle a different number of outputs variables. Finally, performance metrics have to be established and the models will have to be evaluated.

1.4 Delimitation

Open-source libraries were used to implement the classification algorithms in this study. No code was written to build any of the neural network layers or the ability to train/test them -and instead, they were implemented with a high-level API. The experiments only covered five different types of classifiers when there are many more that would be interesting to evaluate.

Only a small subset of the available data was used to train and test the classifiers. Using all of the data would most likely result in a better performance, but it had to be constrained in order to limit the computing power needed.

Another delimitation was that only the English language was considered in the natural language processing. This made it so only one dictionary was needed and reduced complexity for cleaning/transforming texts.

Chapter 2

Theory

This chapter will explain different areas of artificial intelligence that is important to this study. This will include areas such as text classification, machine learning, algorithms used in this study, and methods to work with languages.

2.1 Related work

Classification is a very common problem in machine learning and takes many forms. A common example of text classification is labeling topics. A deep learning classifier is used to determine what a text is about, and it can be used to structure and organize data. Common applications include organizing customer support by issue and categorizing news articles by their content[4].

Related work involves using classifiers for sentiment analysis in news and social media. Sentiment analysis is a technique to help understand peoples emotions in a text. It can be used to determine the effectiveness of customer support or detect the polarity of user reviews[5].

Another application is to use a text classifier to classify documents for retrieval, analysis, and annotation. An algorithm that has proven successful in this regard is the support vector machine (SVM). This is a supervised learning algorithm that is used for both classification and regression. SVM is robust very accurate in documents with a large number of features. A study that used text-classification for tropical diseases compared to SVM and KNN. SVM achieved an accuracy of 92.5% and KNN achieved an accuracy of 49.17%[6].

In a similar paper, linear regression were compared to five other algorithms on the multiple data sets. The study did both binary and multi-class classifications. Linear regression performed relatively well on all of these data sets. The study reported an accuracy of 93.4% in multi-class classification[7].

In a conference paper[8], experiments did measure how well a convolutional neural net (CNN), recurrent neural net (RNN), and SVM performed in short-text classification. This is simply a classification of a shorter text often found in tweets, chat messages, or search queries. The texts don’t contain a lot of features and don't provide much word co-occurrence[9]. Both CNN and RNN achieved very good results and both outperformed SVM. They also used pre-trained word vectors from both GloVe and Word2Vec and found an increased performance. CNN is commonly used in image classification and such, but these results proved that it can be successfully used in text classification as well. RNN, CNN, and SVM had accuracies of 66.2%, 65.5, and 57.0% respectively.

Another publication experimented on text classification using K-nearest-neighbor (KNN)[10]. This is a commonly used text classifier because it's simple to use. KNN assumes that the training data is evenly balanced and suffers from it when it's not. This often leads to majority classes having high accuracy, while the minority classes have low accuracy. This generally results in KNN having a bad overall performance. This publication reported KNN measuring an F_1-score of 82.18% on the Reuters data set.

A thesis used random forest (RF) for sentiment analysis in text. The thesis used the algorithm to analyze peoples twitter messages regarding an airline service. A precision of 90.8% was measured for

two-class classification. It was determined that ensemble classifiers performed better than individual classifiers in this study[11]. The random forest algorithm is flexible and adaptive when addressing fuzzy data[12].

2.2 Text classification

Text classification is the task of placing labels or categories on a text, depending on its content. It has a wide range of applications such as web search, spam detection, and document classification. There is hard to gain insight or value into unstructured data, and classifiers help to organize that data. Manually categorizing data can be hugely labor-intensive and time-consuming. Classifiers allows for a faster and more cost-effective process.

The most common and simplest type of classification is a binary classification. A classifier decides if an item belongs to a single class or not. For instance, a spam detection classifier will either classify a text as spam or not-spam[13].

Multi-class is another type of classification that is used when an item can be classified as one of the multiple classes. The problem is often solved by extending binary classification algorithms. Just as there are algorithms that can't handle multi-class classification, there also algorithms specially designed for this purpose[14]. Neural networks provide an easy extension of binary- to multi-class classification. In neural networks, each output is represented by a neuron, and more outputs can be added by simply adding more neurons.

In multi-class classification, an item can only be assigned to one class. An extension of this is multi-label classification, where an item can be assigned to multiple classes. There is also no limit on how many classes an item can be assigned to. Just as with multi-class classification, algorithms are often extended to suit this problem. As well as adapting algorithms, there also exist methods to convert multi-label classification into multiple binary classification problems[15]. A real-world example of multi-label classification could be placing a newspaper article in the categories "Politics" and "Environment".

2.3 Machine learning

Machine learning is a field in artificial intelligence that teaches a computer to do a task without being explicitly programmed to do it. In traditional programming, to solve a task, programmers have to write each step required to solve the problem. Meanwhile, machine learning algorithms train on data and learns to solve the same problem themselves.

Machine learning is mainly divided into three categories, supervised learning, unsupervised learning, and reinforcement learning. Supervised learning maps an input to an output based on provided examples. Training data is analyzed and supervised learning algorithms will try to figure out the output. This produces a function that can be used to predict outputs on unseen data. If the predictions are good it means that the algorithm has generalized well. Formally, the supervised learning task is to learn $f : x \rightarrow y$ from a training set $D = [(x_1, y_1), ..., (x_m, y_m)]$ where x is the feature space and y are a set of labels[16][17]. A feature in machine learning is a measurable property and in an application like spam detection, features can include email headers, structure, language, and specific terms[18].

In contrast, unsupervised looks for patterns in data with no human guidance. It is commonly used in applications such as data clustering or anomaly detection. An advantage of this method is it that

requires no manual annotation of the data, which makes it suitable for enormous data sets[19].

In reinforcement learning, software agents learn by interacting with their environment. These agents aren't explicitly taught but instead learn from their environments. The agents learn by trial and error, by programming rewards and punishments for their actions[20].

When learning from training data, machine learning algorithms needs a way to determine if the predictions are close to the outputs. This is accomplished with a loss function that measures how much the prediction deviates from the actual values. The are various loss functions that are well suited for different problems. One of the more common ones is mean square error. It measures the average of the square difference between actual values and predictions.

This study will use a loss function called binary cross-entropy, also known as sigmoid cross-entropy, that naturally applies to random variables that may be true or false. This is used in multi-label classification and is described in equation 2.1[21].

$$L = \frac{-1}{N} \sum_{i=1}^{N} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log(1 - \hat{y}_i) \quad (2.1)$$

In equation 2.1, $\hat{y}_i$ is the *i*-th scalar value of the model output, and y_i is the corresponding target value. N is the number of output labels in the model output[22].

2.4 Neural networks

Artificial neural networks, or called simply neural networks, are a highly flexible function approximators that are used in both research and engineering. They can map both linear and non-linear functions and are used in pattern recognition, classification, and other areas[23].

Neural networks are a computational model inspired by the biological brain. They are made up of processing elements known as neurons, and connections between these neurons. These connections make up the structure of the neural network and have coefficients called weights[24]. Figure 2.1 shows a simple neural network model with an input, hidden, and output layer. Neural networks are not limited to one hidden layer, and networks with more than one are called deep learning architectures. Deep learning architectures generally allows for more complex behaviors and are successfully used in many areas. One drawback is that more layers means that the training process will be slower and that the complexity of the neural networks is increased[25].

Neurons are usually assembled in layers and each neuron can have multiple inputs and only one output. If the weighted sum from the input signals exceeds a threshold, an output signal is produced. The training process adjusts the weights, and if tuned correctly, can recognize patterns in data[26].

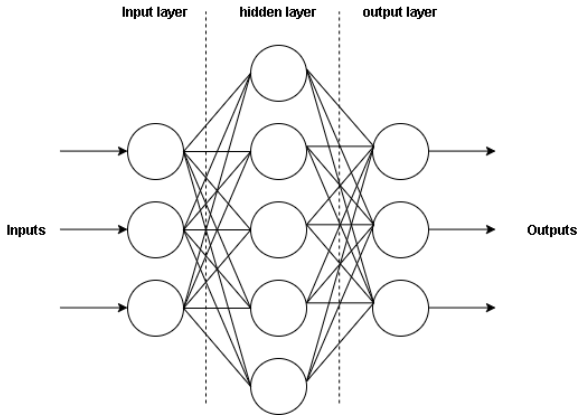


Figure 2.1 A simple neural network with an input and output with three neurons each.

2.5 Linear Regression

A linear regression model is used to find a linear relationship between input and output. A linear regression approach is common in both statistics and machine learning. It assumes that a linear combination can be found in the input variables. Equation 2.2 shows a linear combination where ŷ is the output vector. The parameters $\theta_0,\theta_1,...,\theta_n$, are established in the training process[27].

$$h_\theta(x)=\theta_0+\theta_1 x_1+\theta_2 x_2+...+\theta_n x_n=\hat{y} \quad (2.2)$$

2.6 Convolutional neural network

Convolutional neural networks (CNN) are a type of deep learning network that is found in a variety of applications but most commonly found in image analysis and classification. An advantage to CNNs is that they don't require the same amount of manual labor in feature engineering, that other algorithms do. Instead, CNNs take advantage of convolutional filters that learn features automatically[28].

The CNN architecture is built by stacking certain types of layers. One of the layers is the convolutional layer, which subdivides the input into a defined number of filters, sometimes referred to as kernels. Convolution operations are applied to these filters and produce a matrix of features. The filter offset is determined by a parameter called stride. A bias is added to these features, that are tuned together with the filter weights during training. The process used to produce the feature matrix is shown in figure 2.2.

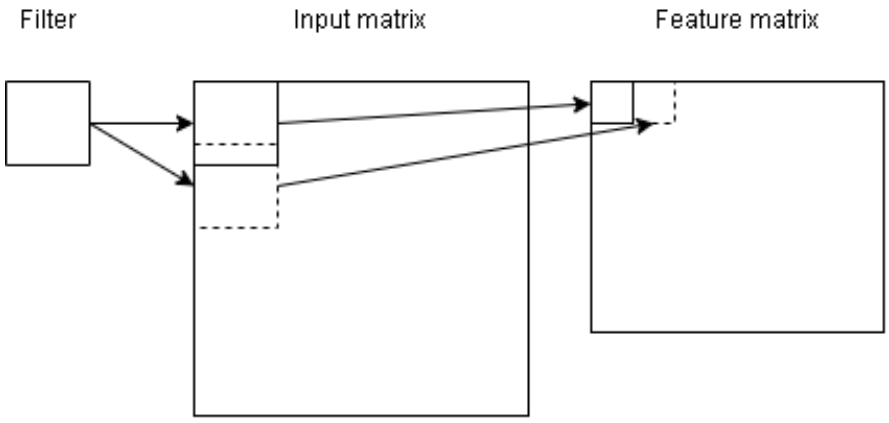


Figure 2.2 An illustration of a 2-d convolutional layer. The process involves convolution operations that are bound by filters. The filters are repeated over the whole input matrix.

Another integral layer is the pooling layer. This layer reduces the dimensionality of the input features and this is done to reduce the sensitivity of the network. The pooling layer combines neighboring matrices from the convolutional layer and produces a sub-sampling of the input. There are multiple ways to combine these matrices, but the most common one is a max-pooling function that uses the maximum value of local neighbors.

A fully-connected layer is used last in the architecture where the weights are used to define class outputs. The output dimension is equal to the number of classes in the target data[29].

2.7 Recurrent neural network

Recurrent neural networks (RNN) are a type of network that allows for temporal behavior. RNNs are built up by cyclic connections that make them a better tool for modeling sequence data, than normal forward feed neural networks. RNNs contain cycles that feed the neuron activations from a previous step in time. A RNN stores these activations in its memory referred to as their internal state. This internal state can store long-term temporal information, which allows RNNs to take advantage of changing contextual windows, as input sequences change[30]. The RNN architecture is shown in figure 2.3.

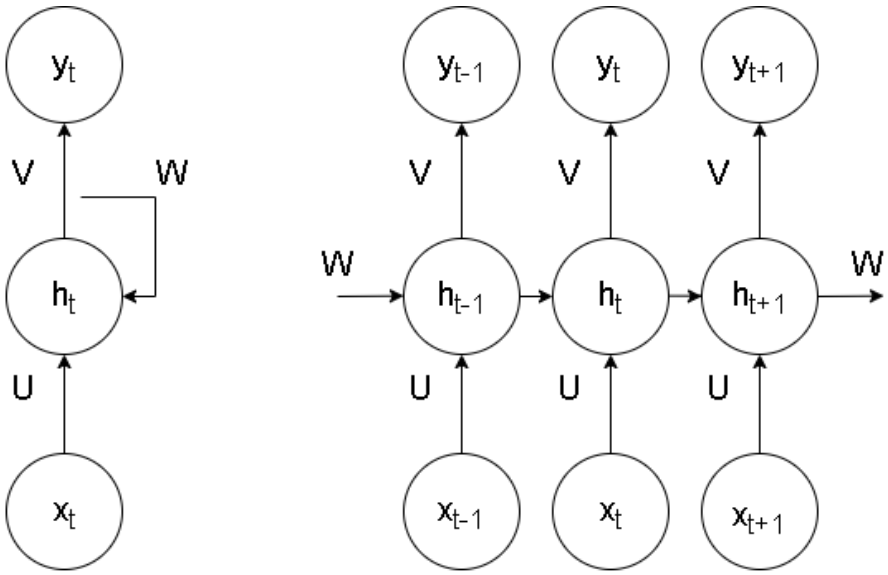


Figure 2.3 An illustration of RNN architecture. The right model shows an unfolded view of the left model. In the models, x_t and y_t are the input and output vectors. U,V represent weights of the hidden layer and output layer, while W are the weights of the hidden state.

2.7.1 Bidirectional

A limitation to RNNs are that they can only use input information from a future frame. A bidirectional RNN (BRNN) can avoid this limitation by training in both a positive and negative time direction. In a BRNN, the neurons are split into two states - one for positive time direction and one for the negative. In contrast to RNN, input information from both the past and future can be taken into consideration when training the network[31].

2.7.2 Long short-term memory

Long short-term memory (LSTM) is a type of RNN architecture. This type of network was invented because RNNs have a vanishing gradient problem. In RNNs, the gradient of the error function gets scaled by a factor. As a result of this, the gradient either vanished or increased exponentially. If it vanishes, the next weight adaptation gets lost or dominated by the gradient. A network with a very low gradient has a hard time learning new things[32].

LSTM networks are like RNNs also capable of memory storage. In addition to this, LSTM architectures have gates that regulate the flow of information inside the cell. Usually, these gates are input, output, and forget gate - but there are variations to this. The input gate controls the flow of input activations, while the output gate controls the flow of the output activations. The forget gate scale the cell internal state, and therefore forgetting and resetting the cell's memory. The LSTM architecture also contains peephole connections from the internal cell to each gate. This makes it possible to learn timings in the output[33]. Figure 2.4 shows an illustration of a LSTM memory block.

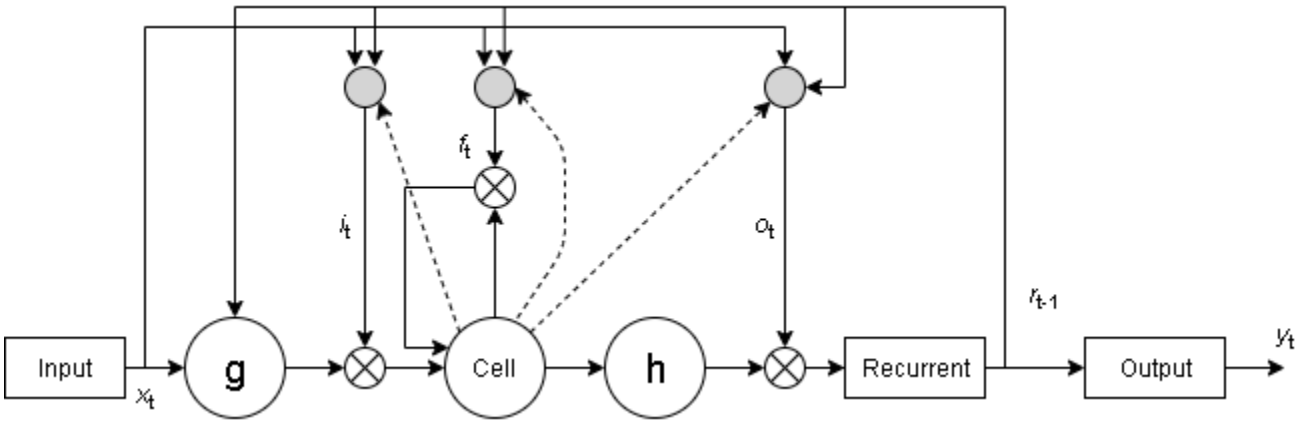


Figure 2.4 A LSTM memory block architecture. x_t and y_t are the input and output vectors. i_t,f_t and o_t are functions for the input gate, forget gate, and output gate respectively. The gray circles indicate activators and the dotted lines are peephole connections.

2.8 Random forest

Random forest is an ensemble learning algorithm that is used for both regression and classification. The algorithm has become popular since it can be applied to a wide range of applications are have few parameters to tune[34]. Ensemble learning works by generating many regressors or classifiers and aggregating their results. The algorithm works by combining multiple random tree models into one model. It also uses an improved version of bootstrap aggregation, commonly referred to as bagging, to create every tree model[35]. An illustration of the random forest algorithm can be seen in figure 2.5.

Bagging is a method to reduce both overfitting and variance by training multiple models on multiple subsets of the training data. Training subsets are randomly drawn from the training set, and each training set is used as an input in the models. The average of these models is combined into a new model that usually performs better than any individual model.

Random forest improves upon bagging by introducing more randomness in the tree splitting decision. In random trees, the tree nodes randomly decides on a certain number of attributes. Each attribute is evaluated with a defined splitting criteria, and the tree node uses the attribute that gives the best value[36].

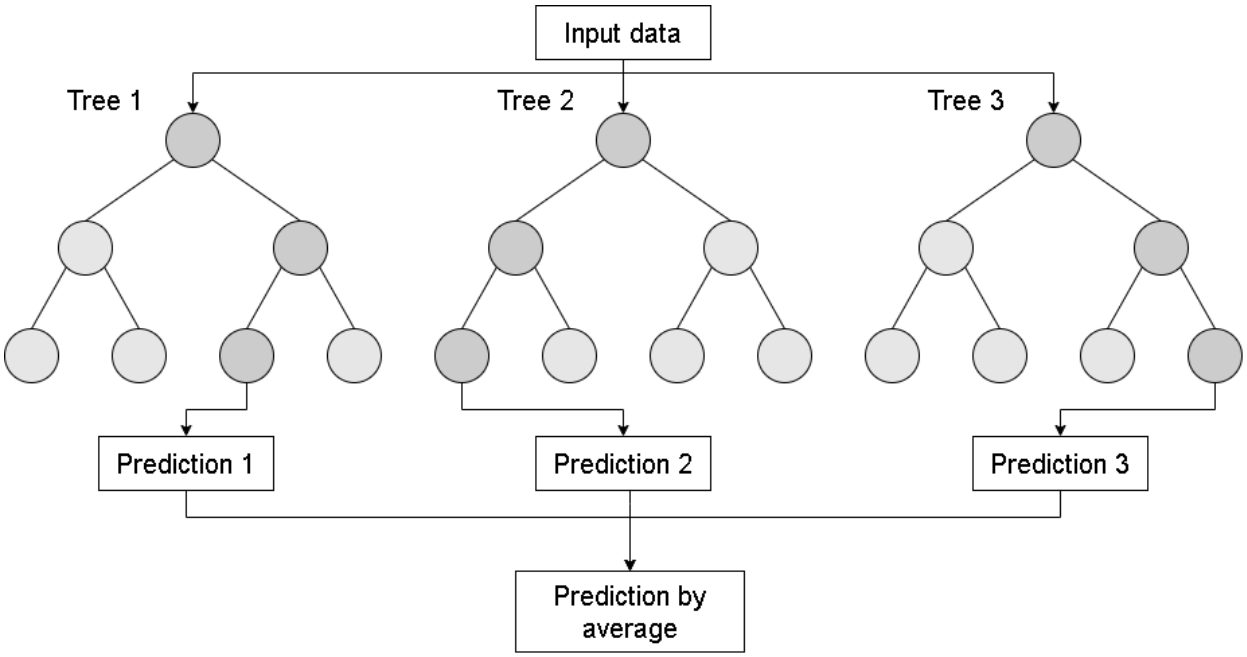


Figure 2.5 Shows a random forest illustration.

2.9 *k*-nearest-neighbor

The *k*-nearest-neighbor (KNN) algorithm is a type of lazy learning. The algorithms previously covered here are types of eager learning. Eager learning algorithms tries to generalize the training data, and construct a function that can later be used for predictions. In contrast, lazy learning algorithms defer input processing until they receive a prediction request. When they receive a request, they reply by combining their input data. As a result of this, lazy algorithms have less computational costs during training, and higher costs when predicting[37].

The training of KNN works by first placing each class in a multidimensional feature space. Classification is done by selecting a set of the *k* closest training examples. The class is set to the most frequent among the *k* closest neighbors. An example of the algorithm is shown in figure 2.6. The algorithm relies on the distance function shown in equation 2.3, where x is the testing example, y is the stored training example – and x_i, y_i are the values of the i^{th} attribute[38].

$$d(x, y) = \sqrt{(\sum (x_i - y_i)^2)} \quad (2.3)$$

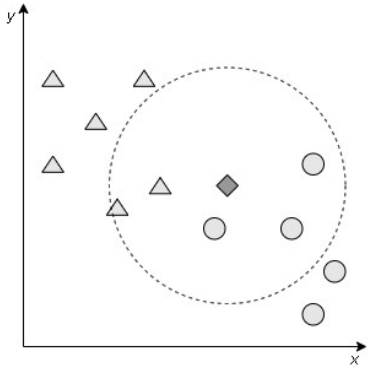


Figure 2.6 Show the KNN classification algorithm when $k=5$. The triangles represent class A and the circles represent class B. The diamond shape is a new data point that needs to be classified. Since the majority of the five nearest neighbors are circles, the data point will be classified as class B.

2.10 Natural language processing

Natural language processing (NLP) is a field in artificial intelligence that focuses on processing and analyzing languages[39]. Some methods and techniques used in NLP will be discussed here.

2.10.1 Stop words

In text processing, it's common to have vocabularies with thousands of words. Many of these words doesn't really contribute to the context of a sentence. In the English language, common stop words include 'a','the','is','you' and 'of'. Filtering a text from stop words reduces the vocabulary size and often improves the performance of learning algorithms. It also speeds up training times and reduces data sizes.

Stop words can be important in applications such as frequency analysis, but in the world of classification, bears little importance. There is also a risk in removing too many stop words by removing words that carry some of the semantic meaning, but the gains usually outweigh the cons[40].

2.10.2 Stem words

Stemming is a process used in text processing that reduces words to their base or root form. For instance, words such as 'likes','liked' and 'likely' can be reduced to 'like' with stemming. Stemming helps to reduce the vocabulary and makes sure that variations of a word are treated the same by learning algorithms[40].

2.10.3 Word embedding

When using word embeddings, it's possible to take advantage of transfer learning. Pre-trained word embeddings are trained on huge data sets and can be used as weights when training learning algorithms. Using pre-trained word vectors often helps to build a larger vocabulary and reduces training time. One of these pre-trained word embeddings is the global vectors for word representation (GloVe). This is an unsupervised learning algorithm that is used to obtain vector representations of words by training on a huge corpus[39]. Using pre-trained word embeddings have proven to increase performance in text classification[41].

2.11 TensorFlow & Keras

TensorFlow is an open-source software math library that is used for machine learning applications. It allows users to build machine learning models with both high- and low-level APIs. Lower-level APIs are useful when more control and flexibility are needed and high-level APIs allow for faster and easier model building. TensorFlow's API is mainly used in Python, but can also be used to build or run inference on models in a variety of programming languages[42][43]. TensorFlow has tools to run on GPUs which can greatly reduce training and inference times on some models. It's used in both production and research and can be found in applications such as mobile apps, trip forecasting, and self-driving cars[44].

Keras is a high-level API for TensorFlow. It has a modular and easy extendable architecture. It allows users to build models that are sequential or a graph of modules that can be easily combined. The library contains many different types of neural layers, cost- and activation functions[39].

Chapter 3

Method

Machine learning is a tool that can be used to both categorize and label texts. This report evaluated and compared five different machine learning algorithms in order to see how well they did in classifying texts. This chapter describes the process of gathering and processing data. The data is retrieved in online databases and code was written to clean and transform it to a suitable format. This format had text fields that the algorithms would learn important features from. The format also had the categorical issue labels formatted with one-hot encoding. The algorithms used these labels as targets in the training. The algorithms linear regression (LR), convolution neural network (CNN), recurrent neural network (RNN), random forest (RF) and *k*-nearest-neighbor (KNN), were implemented with two different libraries. Three different experiments were conducted in order to empirically measure their performance. The first experiment compared all the algorithms in how they performed with a different number of output labels. The second experiment compared neural networks with both sequential and non-sequential models. The last experiment compared the performance of pre-trained word embeddings and stem words.

3.1 Environment

All code for cleaning and processing data was written in python. Python makes it easy to load and manipulate large amounts of data with open-source libraries such as Pandas[45]. The natural language library NLTK provides features for removing stop words, tokenization, and word stemming[46]. This library was used to process and transform the texts in the input data.

The algorithms random forest, and k-nearest neighbor, were implemented with the Scikit-learn library[47]. The library features many types of classification and regression algorithms and provides functions to estimate metrics. The Scikit-learn library was mainly chosen because of how easy the algorithms were to implement with it. The CNN, RNN, and LR models were constructed with Keras.

3.2 Data gathering

3.2.1 GitHub Archive & BigQuery

The GitHub Archive is a project that is actively recording GitHub repositories. They have recorded activity dating back to 2011 and are updating their archives every hour. The archive offers users to browse repositories and view all their details. Each repository archive contains JSON encoded events for different types of actions. These actions events include new user commits, comments, or assigning labels. The events can be imported to a database and processed[48].

```
{
  "action": "edited",
  "issue": {
    "url": "https://api.github.com/repos/Codertocat/Hello-World/issues/1",
    "repository_url": "https://api.github.com/repos/Codertocat/Hello-World",
    "labels_url": "https://api.github.com/repos/Codertocat/Hello-World/issues/1/labels{/name}",
    "id": 444500041,
    "node_id": "MDU6SXNzdWU0NDQ1MDAwNDE=",
    "number": 1,
    "title": "Spelling error in the README file",
    "labels": [
      {
        "id": 1362934389,
        "node_id": "MDU6TGFiZWwxMzYyOTM0Mzg5",
        "url": "https://api.github.com/repos/Codertocat/Hello-World/labels/bug",
        "name": "bug",
        "color": "d73a4a",
        "default": true
      }
    ],
    "comments": 0,
    "created_at": "2019-05-15T15:20:18Z",
    "updated_at": "2019-05-15T15:20:18Z",
    "closed_at": null,
    "author_association": "OWNER",
    "body": "It looks like you accidently spelled 'commit' with two 't's."
  },
  "repository": {
    "id": 186853002,
    "node_id": "MDEwOlJlcG9zaXRvcnkxODY4NTMwMDI=",
    "name": "Hello-World",
    "full_name": "Codertocat/Hello-World",
    "private": false,
  },
}
```


Figure 3.1. The figure shows a JSON encoded payload of a GitHub event, that was retrieved from the GitHub Archive documentation[49]. The event describes the action where a user has edited an issue an already existing issue, inside a repository. It is seen here that the event contains relevant information such as the issue title, body, and a collection of labels. This particular event only has one label and the type is “bug” which is one of the default labels.

To access the data in GitHub Archive, Google BigQuery was used[50]. This is a cloud base data warehouse that lets users process large amounts of data - and for starting users, free of charge. BiqQuery allows users to run SQL queries against the GitHub Archive and retrieve the JSON encoded events mentioned above. A SQL query was used to process all archived repositories between 2011 and 2020. The query iterates over every issue and gathers some of the properties in its events, like the one shown in figure 3.1. It only selects events with the action type of “opened”. This means that we only consider labels from when an issue was opened. This prevents duplicates issues since an issue can be edited multiple times. The properties Url, Title, Body, and Labels are extracted from each issue and are grouped by each issue. To ensure a smaller data set, only issues that have at least one assigned label are considered.

BigQuery doesn't allow direct downloads of big data sets - but instead lets users export the data to Google Cloud Storage. Google Cloud Storage is a file storage web service that hosts data in repositories called buckets. The data from the SQL query was exported to a bucket and split into multiple CSV files. This data was finally downloaded into a personal computer. Since the Google Storage web page doesn't allow easy direct download of multiple files, the files were fetched with terminal commands using Google Cloud SDK. The resulting data set had a file size of 4.65 GB and contained around 7.28 million rows.

3.3 Data processing

3.3.1 Cleaning texts

Now that the data set was processed and downloaded, it was cleaned and transformed before it was used to train or evaluate any algorithms. Originally, the data set that was acquired by the SQL query, contained more columns than were needed. Some columns, like URL, were used to manually validate the data to make sure that the text in each row was describing the correct issue. Moving forward, only the bare minimum of columns were considered. The columns that were kept were Title, Body, and Labels.

In the first part of the cleaning process, the texts were converted into lower-case to make sure that the algorithms don't treat capital and lower letters differently. Then, any letter not from the Latin alphabet were filtered out. This means that special characters like exclamation marks and punctuation marks were discarded. This also means that characters from other alphabets, like the Cyrillic alphabet, were also filtered out. Next, escape characters that define tabs, newlines, or carriage returns, were removed from the texts. The cleaning process also removed all the stop words found in the texts. Finally, white spaces were trimmed from the texts.

3.3.2 Padding

The algorithms required that all the input text have the same length. Figure 3.2 shows the character length distribution of all the issue bodies, and it is shown that it varies a lot. A padding length was decided on - and any text shorter this was padded with zeroes to reach said length. The zeroes were appended at the end of each sentence. Any text longer than the decided length was discarded and removed from the data set.

This padding length of the body was arbitrarily set to 200 which means that around 50% of all the bodies are retained. A shorter padding length reduces the required computing power when training and evaluating the algorithms - while a longer length retains more of the data. The padding length of the title texts was similarly decided as 40 characters.

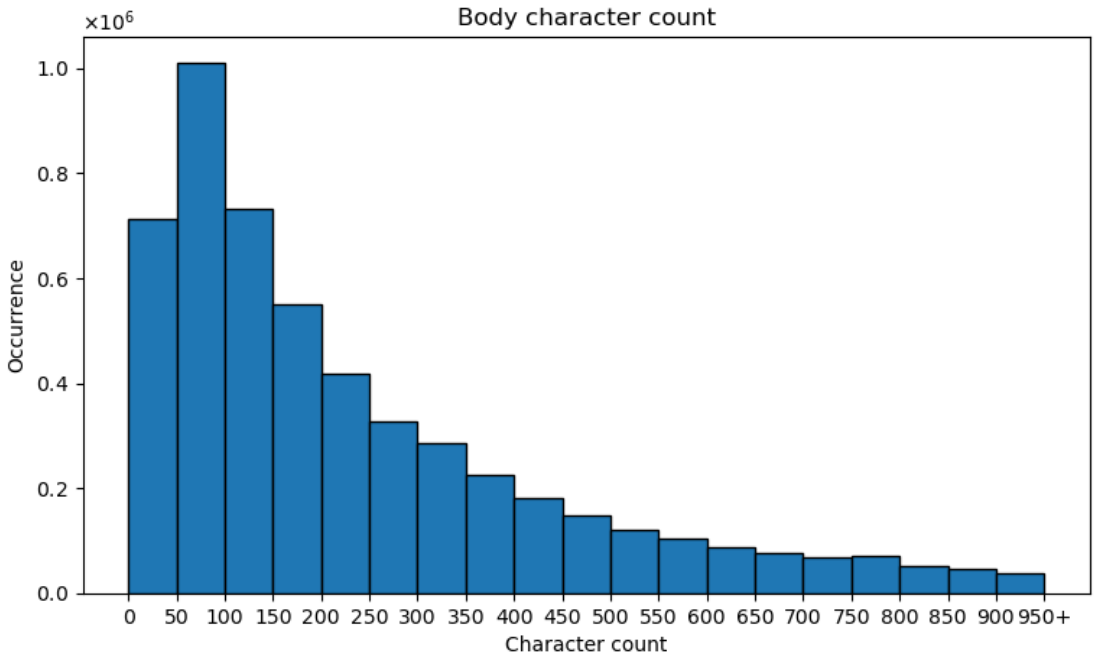


Figure 3.2 A distribution of the character count of all the issue text bodies in the data set. The most common length for a body is between fifty and one-hundred characters, and there are around 6 million bodies with these lengths. Any text body with a character length longer than 950, was merged into the last bin.

3.3.3 Converting labels

The data set contained a column containing all the labels defined as a vector of strings. The data set was transformed with one-hot encoding which meant that the Labels column was dropped. Instead, the data set was appended with one column for each of the considered labels. If the label vector contained a relevant label, the corresponding column was given the value of one, otherwise, it was given a value of zero. It was also made sure that no duplicate label columns were created.

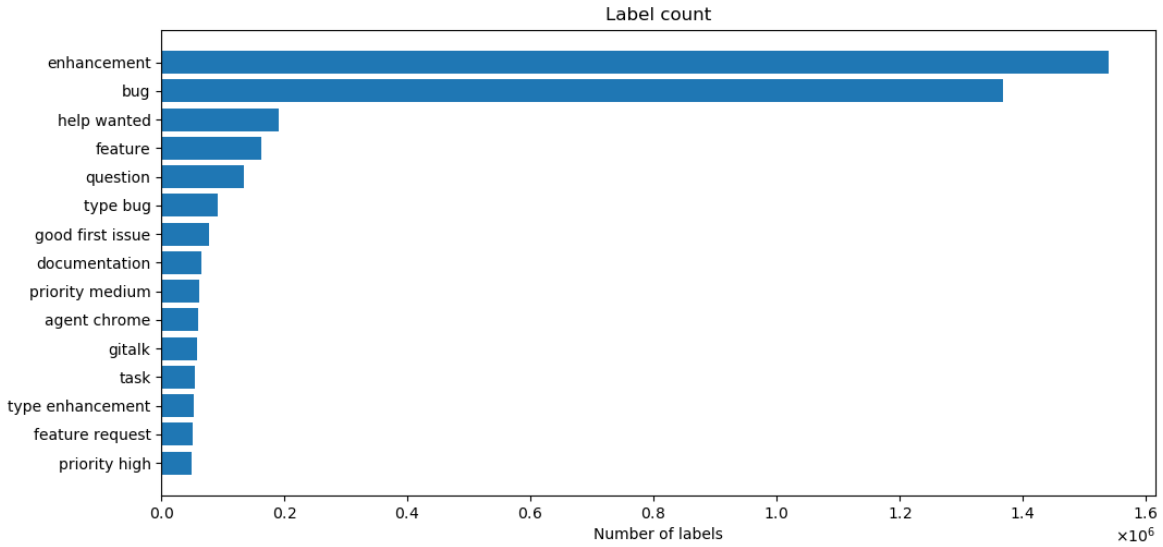


Figure 3.3 Shows the distribution of labels for all the issues. Fifteen of the most common labels are shown. Enhancement and bug are definitely the most popular labels. It is also notable that the top two labels are default labels.3.3.4 Tokenization

The texts were tokenized with the Tokenizer class from the Keras library. This means that each word in the sentences were split into its own string. This class also has the ability to vectorize a large set of texts. That means that a sentence can be converted into a vector of integers which makes it faster for the algorithms to process them. Using this class, a vocabulary was created that maps a word to an index – where words that occur more often will have a lower index. By using this vocabulary, each text in the training data was converted to a vector of integers. One downside to this is that it makes it much harder for any human to read and validate the data.

3.3.5 Embedding

The GloVe library was chosen to embed the texts in the data. Instead of training word vectors from scratch, it's possible to use pre-trained word vectors that can be downloaded from the project website. A package of word vectors that were trained on words from Wikipedia – and Gigaword. The package contained four versions with different dimensions – 50, 100, 200, and 300. The dimension refers to the length of the vectors and larger vectors can store more information. The word vectors were trained on four hundred thousand uncased words and were contained in a text file totaling 989 megabytes.

The text file was parsed and each line was converted to a numpy array and inputted into a dictionary. Next, the dictionary was mapped to the words that were acquired from the tokenization process. This resulted in a matrix with a width of 50, the same as the word vector dimension, and a height equal to that of the items in the tokenizer vocabulary. Each index represents the integer value of each tokenized word and the columns show the word vectors acquired from the embedding.

3.4 Model implementations

Five different algorithms were implemented to help answer the research questions. Three types of neural networks were used, together with two other types of supervised learning algorithms. The neural networks were realized with the Keras library and the latter two algorithms were constructed with the help of the Scikit-learn library. All algorithms were implemented with slightly modified versions of those that were found in TensorFlow and Scikit-learn documentations. The random forest classifier was used with, the default number of 100 estimators. The k-nearest neighbor classifier was implemented with 5 neighbors, which is the default parameter.

The neural networks were built with models that shared a similar overall structure since all of them started with the same input and output layers. Each neural network was implemented with two designs (see Appendix B). The first design is a sequential model that only considered one text input. The second model was non-sequential and used two text inputs. The input layer had an input dimension equal to that of the padding length (see section 3.3.2). This was connected to an embedding layer that could either use pre-trained weights or not. This layer had an input size that of the text vocabulary size. The output size was that of the word-vector dimensions (see section 3.3.5). The last output layer for every model was a dense layer with as many outputs as the desired number of labels. The output layer also used a sigmoid function for its activation.

The models for the sequential neural net was implemented as follows. The convolutional model was implemented by connecting the embedding layer to a Convolutional1D layer followed by GlobalMaxPool1D, dense - and dropout layers. The Convolutional1D had 32 filters and a kernel size of 7. The recurrent model connected a LSTM layer to the embedding layer. The LSTM layer used a bidirectional wrapper and used 32 units. This was followed by a dense and dropout layer.

For the linear regression, the embedding layer was followed by flattening, dense, and dropout layers. The dense layer had 32 units. The non-sequential models had two separate branches that were combined with a concatenate layer, as seen in figure 3.4. The reason for having two branches is to allow input for both title and body texts.

All dropout layers had a dropout rate of 20% and were used to prevent overfitting[51]. A loss function had to be defined for the model and since the output is a multi-label format, a categorical cross-entropy was chosen. The model was compiled with the “Adam” optimizer.

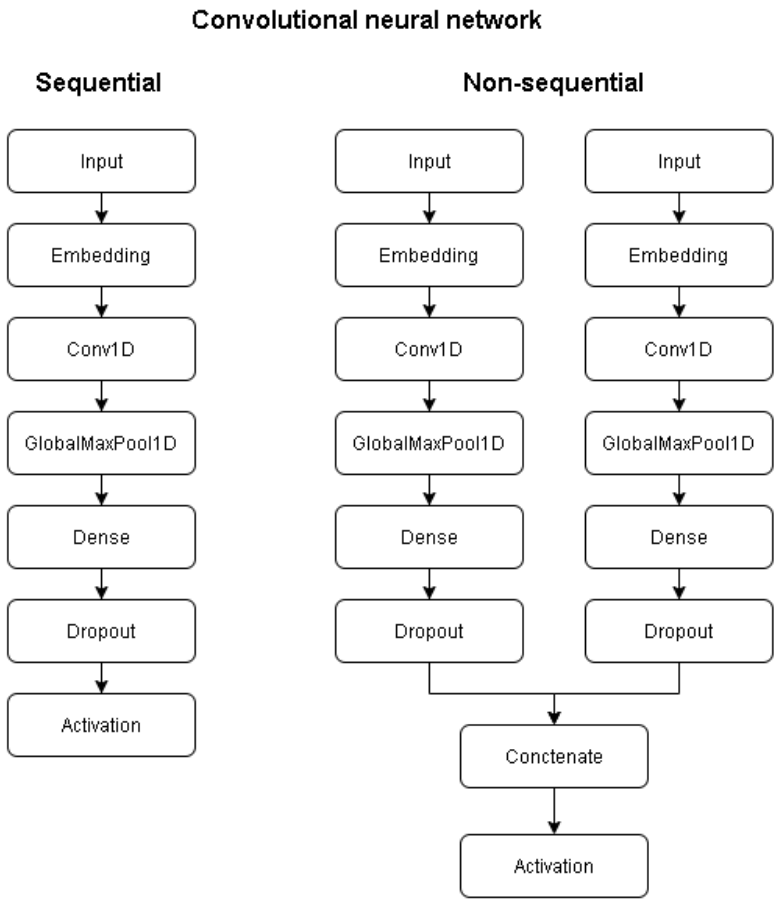


Figure 3.4 Shows the archetypes of the convolutional neural networks. The left side shows a sequential model with one input and output. The sequential model is organized as a stack of layers and each layer has only one input and output[52]. The non-sequential models use two inputs, which used to input both title and body texts.

3.5 Balancing and splitting data

To provide a more balanced data-set, not all available data was used to train and evaluate the algorithms. As seen in figure 3.3, the class distribution is heavily skewed towards the two most popular labels. This type of imbalance can become a problem since it's possible to lose important features of the minority classes[53]. Instead, smaller data-sets are selected by extracting a certain number of data points from each class. Note that this didn't result in each class having exactly this amount of data-points since certain rows can have multiple labels, which meant that some classes had more instances. These data-points were randomly sampled and all random selection used a seed for the random generator to make sure that the results were reproducible. This data-set reduction also helped to reduce the time required to run the experiments.

The data were randomly split into a training set and a test set. The algorithms are trained on the training set and then evaluated on the test set. This is to get an accurate evaluation of data that the algorithm hasn't seen. Additionally, *k*-fold cross-validation is used to discover the average metrics of the trained algorithms. The training set is further split into *k* equally sized subsets, where *k* is the number of partitions. The training process uses *k*-1 of these partitions for actually training the algorithms, and the last unused one is used to validate them. These partitions are shuffled around *k* times and an algorithm is trained each time using different parts of the training set to train it. It's popular to set *k* to 5 or 10, which is enough to give a likely estimate[54]. The *k* value was set to five which means that 5 different models will be trained for each test. Setting *k* to 10 would give better estimates, but would at the same time double the training time.

3.6 Experiments

Three different experiments were conducted to measure how the algorithms performed. The cleaned and processed data set acquired from the previous sections was divided into two new subsets, shown in table 3.1. Three smaller data sets, B, C, and D were extracted with a different number of output labels.

Data set B was constructed with six of the default GitHub labels[55]. The default labels "Invalid", "Duplicate" and "Wontfix" weren't considered since they have a much lower occurrence than the other labels. Around 10,000 data points were selected from each label and made up a data set of approximately 60,000 rows. Data set C were populated with 60 different labels, an increase by a factor of 10 compared to data set B. Here, around 1,000 data points were selected from each label, also making up approximately 60,000 rows. For data set D, 120 unique labels were selected in the same fashion.

Table 3.1 Shows the data sets used in the experiments. Data set A was acquired by downloading data from GitHub Archive and processing it as described in this chapter. Data set B, C and D are all smaller subset of data set A.

Name	Number of rows	Unique labels
Data set A	5,421,469	184,653
Data set B	59,609	6
Data set C	59,556	60
Data set D	59,554	120

All data sets were each divided into test and training sets. It was split so that 80% of the data was designated as training data and the rest 20% was used as test data. The training data was further split into train 80% and 20% validation data. The data was randomly partitioned with cross-validation and the validation data was in the calculation of the loss function. The 80-20 ratio used to split the data was arbitrarily chosen since it's a commonly used ratio - and there isn't a rule set in stone for choosing this ratio. By using K-fold cross-validation, it was possible to calculate the mean and standard deviation, for each algorithm.

For the first experiment, all algorithms were trained and evaluated on data set B, C, and D. When training, the algorithms only used the issue body as text input and the labels as target data. For all the experiments, the neural networks were trained until no further loss decrease was measured. The reason for stopping when no further loss was detected was to prevent overfitting.

A second experiment did evaluate how well the neural network algorithms performed when they were trained to consider both the text in both title and body. The experiment only considered the neural networks, and not random forest, k-nearest neighbors, since it was much easier to implement a non-sequential model with Keras. One option is to instead of using two inputs - combine title and body into one text. Instead, this study used neural network models with two inputs, because it's possible to train title and body separately. This means that the title and body can be trained with different vocabularies and embedding weights. This experiment was trained and evaluated using data set B.

The third experiment was also trained and evaluated on data set B in order to measure the effect pre-trained embedding weights and using stem words, had on the performance of the neural networks. To measure the embedding weights, the weights from GloVe were used in the embedding layer of each model. In order to measure the effect of using stem words, the title and body were processed and all words were converted into stem words. This modified data was used to train and evaluate all neural networks.

4.7 Metrics

To evaluate the classifiers, predictions were made on data that was unseen by the training process. Recall that the labels in the data are formatted as a one-hot vector and using a model for prediction will generate a guess for each label. This guess can either be true or false and predicting a label can have four different outcomes. These different outcomes are called true positive, true negative, false positive, and false negative. A true positive is when the guess is true and the actual label is true. Meanwhile, a false positive is when true is guessed but the actual value is false. A true negative is when a false value is correctly predicted and a false negative is when it's falsely predicted[56].

Precision and recall are two metrics that can measure the accuracy of the classification. Precision measures the fraction of classified labels that are correctly predicted. Recall refers to the fraction of correctly predicted labels among all positives[57].

$$precision = \frac{T_p}{T_p + F_p} \qquad (1)$$

$$recall = \frac{T_p}{T_p + F_n} \qquad (2)$$

Precision is calculated with equation (1) where T_p is the true positives and F_p are the false positives. Recall is calculated with equation (2) where F_n is the false negatives. The F_1 score is another metric used to evaluate the algorithms. This is the harmonic mean between precision and recall and can be calculated with equation (3). In a publication, it is mentioned that this metric is often used to evaluate multi-label classifiers and that one should aim to maximize this score. It's also stated that this metric does well in imbalanced data-sets[58].

$$F_1 = \frac{2 \cdot T_p}{2 \cdot T_p + F_p + F_n} \qquad (3)$$

When predicting labels with a classifier, it outputs a value between 0 and 1 for each label. A confidence threshold is used to determine if this value should be true or false. Changing this threshold has an effect on precision and recall. Increasing the threshold will increase precision while decreasing recall - and the other way around when decreasing the threshold. Choosing to prefer either recall or precision is important when designing a product, but since this study is impartial to these decisions, a normal value of 0.5 was chosen. Any value equal or higher than this would result in true, otherwise false.

The precision, recall, and F1-score was calculated by micro-averaging all the labels. This is done by calculating the sum of all true positives, false positives, and false negatives of all the labels. This results in each label having the same importance, instead of being weighted towards imbalanced labels. By doing this, it's possible to get high scores even though the classifier is performing badly on a single label[59].

Chapter 4

Results

Five supervised-learning algorithms were compared in the three experiments, which are detailed in section 3.6. The algorithms considered were Linear Regression(LR), Convolution Neural Network(CNN), Recurrent Neural Network(RNN), Random Forest(RF), and k nearest neighbor (KNN). These algorithms were trained and using k-fold cross-validation (see section 3.5) and tested on unseen data. When evaluating the algorithms, the metrics precision, recall, and F_1-score were measured. Data set C and D had 60 and 120 output labels and were only used in the first experiment. All the rest of the experiments were trained and evaluated on data set B.

4.1 Comparing the algorithms

The first experiment evaluated the performance of each algorithm when trained/tested on data sets with varying numbers of unique output labels. The results of this experiment have been compiled and are shown in figure 4.1 and table 4.1. Figure 4.1 shows the F_1-score for each algorithm and the standard deviation is shown in the error bars. The figure shows that CNN has the highest F_1-score for every data set and scored 0.4374, 0.4506, and 0.4390 respectively. The neural networks scored very evenly across all data sets and KNN had consistently a much worse performance overall. RF had the worst score for the first data set, but it greatly increased when trained on data sets with more labels.

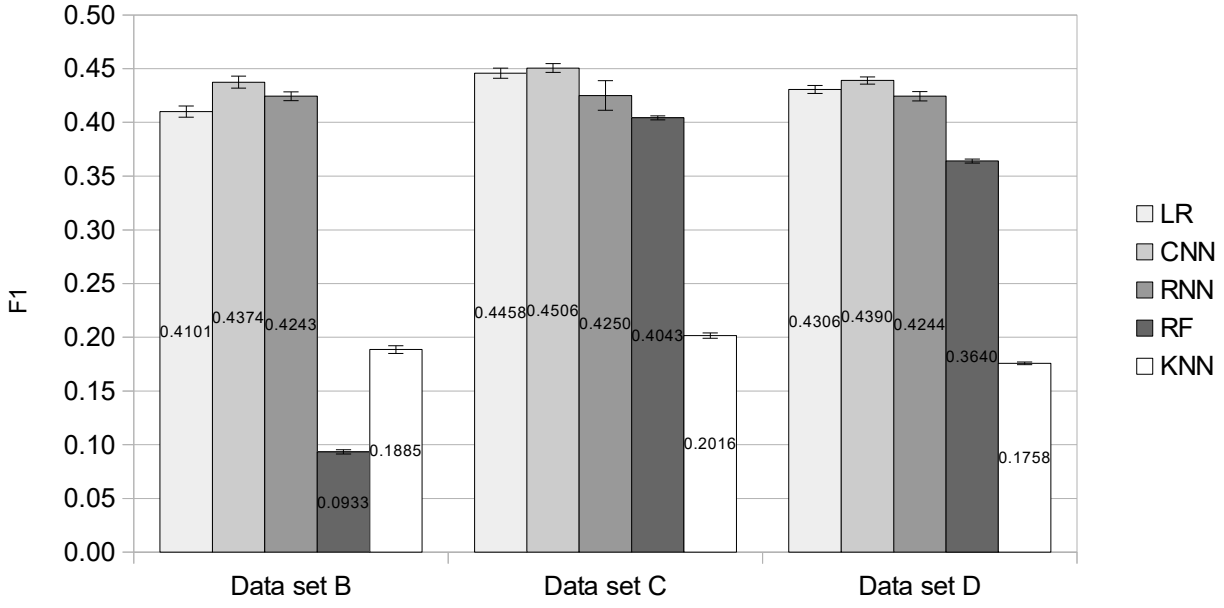


Figure 4.1 Comparison of the mean F_1-score for the five different algorithms. The figure shows how each algorithm performed on three different data sets (see section 3.6). The data sets used for evaluating the algorithms had a different number of output labels - 6, 60, and 120 respectively.

The figure shows that the neural network classifiers reaches a higher F1-score than the other supervised learning algorithms, RF and KNN. This was far more prominent for the data set containing 6 unique labels, where CNN, the best scoring neural network, score more than a 2 times higher F1-score than KNN and more than RF. It's a very even performance among the neural networks with CNN performing slightly better than LR and RNN. The neural networks also showed a somewhat higher standard deviation, compared to the other algorithms.

In data set C (60 labels), RF had the most noticeable change in relative performance. With more output labels, RF achieved an F1-score comparable to that of the neural networks. LR also reached a higher score than RNN in this data set.

The classifier performance in the data set D (120 labels) showed the same rankings as the previous one. This data set had a more even F1-score between all of the neural networks and the score RF was slightly worse. KNN had a much worse performance across all data sets.

Table 4.1 Shows the mean precision and recall for all algorithms after they were evaluated in the first experiment. The bold values indicate the highest score for each column. RF used 100 estimators and KNN used 5 neighbors.

	Data set B (6 labels)		Data set C (60 labels)		Data set D (120 labels)	
	Precision	Recall	Precision	Recall	Precision	Recall
LR	0.4418	0.3829	0.7299	0.3210	0.7640	0.2999
CNN	0.4547	0.4219	0.6028	**0.3598**	0.6496	0.3316
RNN	0.4245	**0.4243**	0.5983	0.3360	0.5882	**0.3321**
RF	**0.6179**	0.0505	**0.8917**	0.2614	**0.8916**	0.2287
KNN	0.3150	0.1345	0.4930	0.1267	0.5345	0.1052

Figure 4.2 shows the time it took to fit each algorithm to the training data. The neural networks that a noticeable longer time to train than RF and KNN. RNN had the longest average training time of 2179 seconds. The second highest was that of CNN with 1327 seconds. In comparison, LR took 712 seconds to train. LR is the simplest model and only took 32.67% of the time it took to train RNN. RF only took 36 seconds to fit and KNN took 6 seconds.

Figure 4.3 displays the time it took to run inference, for each algorithm, on data set B. LR had the fastest inference time at 1.25 seconds. CNN had the second-fastest at 1.98 seconds, followed by RF at 3.16 seconds and RNN at 8.72 seconds. Finally, the measured time of KNN was 120.45 seconds.

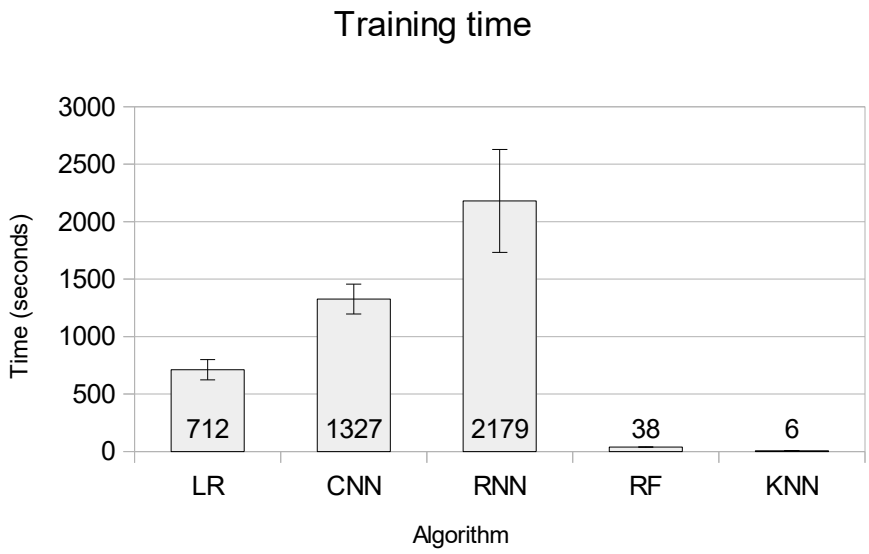


Figure 4.2 The mean training time for the five different algorithms, The y-axis shows the time it took in seconds to train an algorithm and the x-axis shows the algorithm trained.

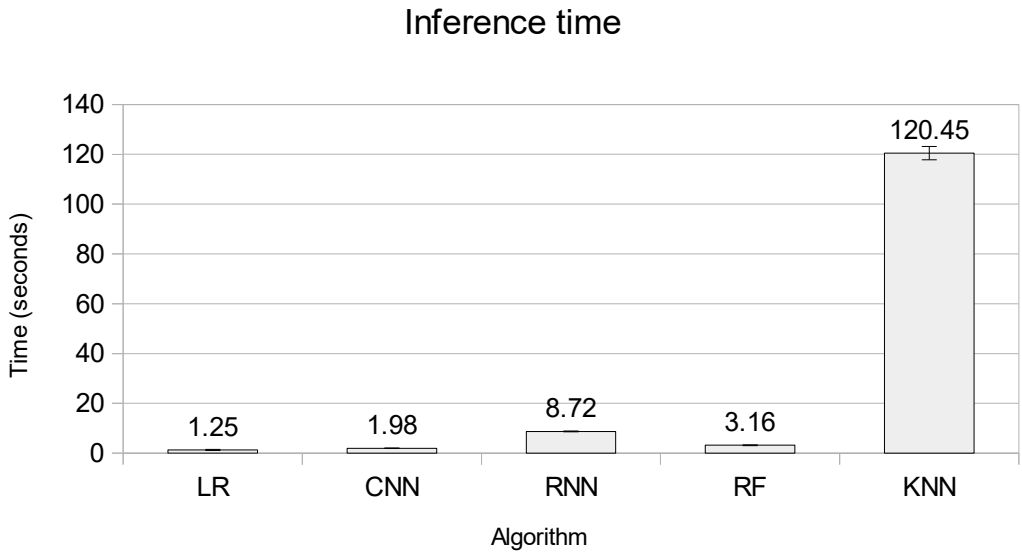


Figure 4.3 The mean time it took to run inference, i.e. predictions for the five different algorithms, The y-axis shows the time it took in seconds and the x-axis shows the algorithm used.

4.2 Improvements to the neural networks

The second experiment evaluated sequential and non-sequential models for all neural networks. The results of this experiment are shown in figure 4.4. The figure shows the F_1-score for LR, CNN, and RNN. The standard deviation is shown in the error bars. CNN scored the highest for both the sequential and non-sequential models, with F_1-scores of 0.4374 and 0.4614 respectively.

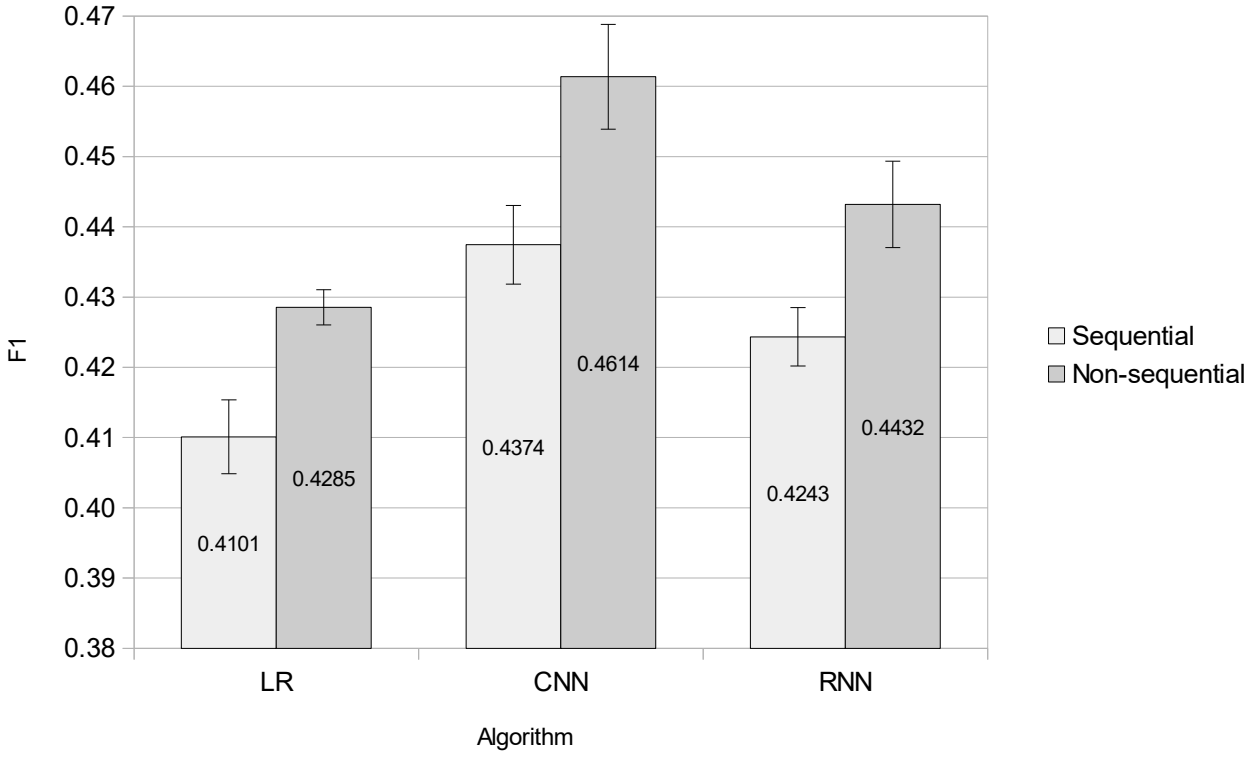


Figure 4.4 Mean F1-scores for each of the neural net algorithms when evaluated on sequential and non-sequential models.

The third experiment measured the F_1-score when slight modifications have been made to the non-sequential neural network models. Figure 4.5 shows that only CNN had a small improvement when stemming the words in title and body. The GloVe embedding, the combination of it together with stemming words, all showed a worse or equal performance for all models. The highest score was achieved by the CNN model with stemming words, and this is also the highest F1-score across all experiments.

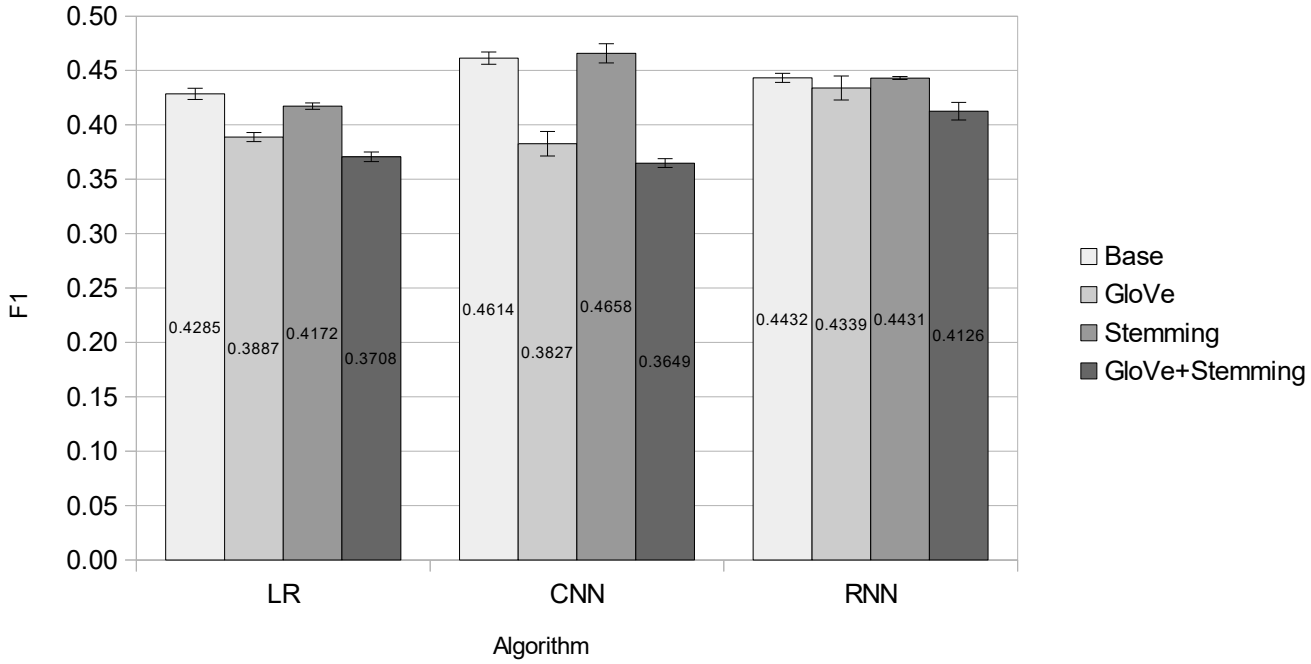


Figure 4.5 Comparison of the mean F1-scores for the neural network after they have been trained with small adjustments. The performance is measured when the models are trained on either pre-trained word vectors from GloVe or that the words in text and body have been stemmed. The performance is also measured when both options are used.

Chapter 5

Discussion

This study aims to find out which classifier algorithms that are suitable for a tool that would automatically assign labels to GitHub repositories. Even though assigning GitHub labels was the focus here, the results can be used to implement a variety of different applications. This could apply to any task that requires one or more text inputs and multiple categorical outputs.

5.1 Comparison of algorithms

When evaluating the five algorithms on the three data sets with different numbers of output labels, they achieved the same F1-score rankings on 60 and 120 labels. The rankings differed when trained an evaluated on 6 labels. When increasing the number of labels from 6 to 60, the precision increased while the recall decreased, except for RF were both improved. The performance of the neural networks stayed high for each data set which suggests that they are more reliable and more indifferent to the structure of data.

Even though the algorithms were trained on data sets that had a very similar amount of data rows, the data sets with a higher number of output labels had a much higher amount of negative samples. This could be a factor in the increased precision, which RF seems to have benefited greatly from. Generally, adding more data to the training will increase the performance of deep learning algorithms.

On the other hand, KNN had a very bad performance across all data sets. This algorithm is very dependent on the training data that is used to fit it and is very sensitive to the structure of the data. In text categorization done by another study[60], KNN was used to achieve a much higher F1-score. The authors suggested that the algorithm is sensitive to the selected hyperparameters. This study didn't explore the hyperparameters for either of the algorithms, which might be a reason for the low performance of KNN. Unless each variation of hyperparameters is empirically tested, there is no guarantee that the best parameters were selected. This also holds true for all other algorithms. In section 2.1, it was mentioned that KNN suffers when one class is performing badly, and this might be same reason for its bad performance here.

This study didn't explore the hyperparameters for either of the algorithms, which might be a reason for the low performance of KNN. Hyperparameter tuning is an important area in machine learning and is used to optimize models. Not tuning hyperparameters is a big drawback to this study but was decided against because of limited time and computing power. An alternative would be to reduce the project scope by limiting the number of examined algorithms, allowing time for hyperparameter optimization.

In a study comparing supervised learning algorithms[61] in classifying crop from aerial images. These algorithms also included Random Forest and neural networks. RF performed poorly when comparing a low amount of images, and converged to an accuracy near-equal that of neural network algorithms when adding more images. Meanwhile, neural nets achieved higher accuracy and were much less dependent on the number of images. This is a similar result to what was shown with the F1-score rankings across the different data sets.

5.2 Improving the neural networks

The results from this study show that the neural network algorithm achieved a high F1-score on data sets with varying output labels. This suggests that the sequential models had an ability to learn features from the body text. The non-sequential models performed even better than their sequential counterparts. These models also considered the text from the titles, and since they reached a higher F1-score, we can draw the conclusion that there are important features that can be learned from the title texts as well. There isn't a guarantee that more input variables will increase neural network performance.

Non-sequential models are used to train on the title, but another option would be to concatenate the title and body into one larger text. The main benefit of using non-sequential models is that it's possible to use two embedding layers simultaneously. This allows the models to use two distinct vocabularies, and assign different weights to different words. A word that exists in both texts could have different meanings in each of them.

The relative difference was 4.49% higher for the non-sequential LR compared to its sequential counterpart. Meanwhile, the relative difference was 5.49% higher for CNN and 4.45% for RNN. CNN benefited the most from also training on the titles, and reached the highest F1-score for all models.

The same level of success wasn't reached when trying out pre-trained word embeddings from GloVe and stemming words in the input data. Both methods were used in an effort to increase the F1-score of the non-sequential models, but in most cases had a lower score. CNN was the only model that showed a slight increase in score when using stemmed words. This granted a negligible increase and using GloVe embeddings and the combination of them, both decreased the score significantly. This means that the models perform better when they can train their own embedding weights. The data set has a vocabulary that uses words and abbreviations that are mostly found in software development, which might be the reason for the pre-trained vectors lackluster performance. The texts contain words like SSL, Xamarin, and Dotnet - which are not found in the GloVe word embeddings.

Stemming words reduces words to their root form and is a common method in NLP. Even though the process only improved the performance of CNN very slightly, it might still be useful to use with this model. Stemming words reduce the vocabulary size and therefore the number of parameters used in training of the model, which will, in turn, reduce the training time. If facing time constraints, this will in turn allow for a larger data set to be used for training.

5.3 Evaluating the performance

This study has measured and discussed the relative performance of each algorithm. Metrics such as precision, recall, and F1-score have been used in an effort to gain insight into their effectiveness. But what is a good F1-score and do the algorithms reach a high enough score in order to be used in practical applications?

In an effort to answer this question, the F1-score from the results can be compared to a classifier that did predictions on the test data by flipping a coin. For every label prediction, either a true or false was assigned based on a 50% chance. The results from this gave an F1-score of 0.3117 and a precision of 0.2264. The best algorithm result from the experiments gave an F1-score of 0.4658 which is 49.44% higher than the coin-toss classifier. This means that the tested algorithms are much more likely to give an accurate prediction. The compiled metrics from the coin-toss classifier can be found in Appendix A.

The algorithm performance greatly depends on the data set provided. In a research publication examining multi-label classifiers[58], the authors experimented with four different algorithms across three distinct data sets. The data sets used were Reuters, WIPO, and JPAT. Reuters contains English news articles, WIPO contains international patent documents and JPAT consists of Japanese patent documents. In their experiments, their classifiers achieved an F1-score between 82.4 and 87.8 on the Reuters data set, between 40.5 and 51.4 on the WIPO data set, and between 32.2 and 42.1 on JPAT. The authors concluded that their methods work well on complex data sets. The data set in the study is more similar to the patent data sets of WIPO and JPAT. Both data sets carry words with less semantic meaning than that of Reuters since they are very technical and esoteric in nature. The texts also tend to be shorter than that of the texts from Reuters. Longer texts might contain useful features that help the performance.

Training and inference times are an important aspect of real-world applications. The results show that both differ for each algorithm that was tested. The training time for the neural network was far higher compared to that of RF and KNN. RNN had the highest training time and KNN had the lowest. The training time of RF and KNN is negligible compared to the neural networks, and this is a big drawback to neural networks as training time increases with model complexity. Also, if one is interested in finding the optimal hyperparameters for a neural network, each variation has to be trained.

When using machine learning in applications, we are not limited to only using one model. From the results here it shows that the algorithms have different strength, and one idea is to combine CNN with RF. Since RF can achieve a very high precision, we could run inference on RF first, and if it doesn't find a class above the desired prediction threshold, CNN could be used as a backup since it has a higher recall. A solution like this would take a longer time to run, but would probably achieve a better result.

5.4 Sustainable development

5.4.1 Social Development

The users of any deep learning algorithm bear the responsibility that the training data is intelligently cleaned and processed. In March 2016, Microsoft released a chatbot on twitter, that learned by interacting with users. The bot started out by emulating a teenage girl that had her own interests and used slang, but within hours it had devolved to tweet sexist and racist remarks. After only 16 hours, Microsoft had to suspend to the bot and apologize for its remarks[62]. This is a good example that applications using machine learning can work outside their intended use if the data becomes corrupted, or is poorly chosen to begin with. For automatically labeling GitHub issues, it's easy to imagine ways for nefarious users to corrupt the application. It would for instance be possible to flood issues that label a minority group as a "Bug", and a majority group as "Enhancement". If not taking the proper precautions, a useful program could quickly degenerate into an anti-semantic bot.

5.4.2 Ecological Development

Machine learning is a powerful technology that can be used in a wide range of directions. News articles continue to highlight machine learning powerful uses, but very rarely mention one of its issues - its carbon footprint. In a study, it was estimated that the training of a large and complex machine learning model could produce 284 metric tons of carbon dioxide[63]. This is when the electricity used is produced from non-renewable sources. It's therefore important to make sure that the electricity used to train and run inference, is generated by carbon-friendly means.

5.4.3 Economical Development

From an economic perspective, automation in software has many benefits. The main benefit is that it allows developers to be more effective with their time. Time that was used for manual labor can be diverted to other tasks. Automating tasks allows for fewer human errors, and time dedicated to resolving these errors. It also makes tasks less repetitive and boring and therefor increasing job enjoyment.

Chapter 6

Conclusions

The aim of this study was to investigate how various algorithms performed in multi-label classification. It was concluded that some of the algorithms do in fact show good metrics. Linear regression (LR), convolutional neural network (CNN), recurrent neural network (RNN), and random forest(RF) did very well and all would be suitable candidates for implementation in a classifier. *K*-nearest-neighbor (KNN) didn't provide any good performance for this particular problem. The neural networks, LR, CNN, and RNN, showed different strengths and weaknesses compared to RF and KNN. They demonstrated an overall well-rounded performance and had mostly faster inference times.

The relative performance differed when exploring categorical output with a different number of output labels. RF showed a very drastic improvement when adding more labels. CNN consistently displayed the overall best performance for all label variations.

The non-sequential models of the neural networks proved better than their sequential counterparts. The sequential models showed that each neural network could learn important features in the body text, inside of a GitHub issue. In the same fashion, the non-sequential determined that the networks could learn important features from the title text as well and that a multi-branch design gave a better result.

Pre-trained word embeddings from GloVe were also tested. Using them resulted in a worse performance for LR, CNN, and RNN. Instead, the neural networks had better results in training their own word embeddings. Stemming the words in the input data decreased the performance of LR and RNN, and providing a negligible increase in CNN. Combining a non-sequential CNN model with stemming of the input words, gave the highest performance across all experiments.

6.1 Future improvements

The experiments could be improved by increasing the amount of training and testing data. This could lead to higher metric measurements for all algorithms and would be an easy thing to do. Provided better hardware, one could also increase the *k* value in the cross-validation, for less statistical variance.

Among all algorithms capable of multi-label classification, only a few were of them were evaluated. More experiments could cover a wider range of promising algorithms. The algorithms were mostly trained with the default parameters, and it would be interesting to explore more of these variations. Using a more complex model, such as the non-sequential designs, proved impactful. This avenue could be further explored by evaluating less naive models. For instance, some advanced image recognition model stack multiple convolutional layers, which might be an interesting option for text classification. Another option is to combine the model in a hybrid approach.

Further work could involve implementing the classifiers into a marketable application. This would include deciding on the most desirable metrics and choosing suitable algorithms that do well in desired aspects. The application would need to implement the same data processing pipeline, that is used for training and testing here. It would also have to be capable of running inference and converting raw prediction values into categorical classifications.

Bibliography

1: Shanhong L. Artificial intelligence software market revenue worldwide. Statista. 2020 Mar 13 [cited 2020 Aug 10]; Available from: https://www.statista.com/statistics/607716/worldwide-artificial-intelligence-market-revenues/

2: Anderson T. 'It's really hard to find maintainers...' Linus Torvalds ponders the future of Linux. The Register. 2020 Jun 30 [cited 2020 Aug 10]; Available from: https://www.theregister.com/2020/06/30/hard_to_find_linux_maintainers_says_torvalds/

3: Liu J, Chang W, Wu Y, Yang Y. Deep Learning for Extreme Multi-label Text Classification. SIGIR 17. 2017.

4: MonkeyLearn. Text Classification. [cited 2020 Aug 15]; Available from: https://monkeylearn.com/text-classification

5: Li X, Xie H, Rao Y, Chen Y, Liu X, Huang H, et al. Weighted Multi-label Classification Model for Sentiment Analysis of Online News. IEEE. 2016. 215-222

6: Wulandini F, Nugroho A. Text Classification Using Support Vector Machine for Web mining Based Spatio Temporal Analysis of the Spread of Tropical Diseases. ICTAS. 2009. (11)

7: Zhang T, Oles F. Text Categorization Based on Regularized Linear Classification Methods. Information Retrieval. 2001. 5–31

8: Lee J, Dernoncourt F. Sequential Short-Text Classification with Recurrent and Convolutional Neural Networks. NAACL. 2016. 515–520

9: MonkeyLearn. Short Text Classification. [cited 2020 Aug 15]; Available from: https://monkeylearn.com/short-text-classification

10: Tan S. An effective refinement strategy for KNN text classifier. Expert Syst. Appl. 2005. 30(2): 290-298

11: Wan Y. An Ensemble Sentiment Classification System of Twitter Data for Airline Services Analysis. Nova Scotia: Dalhousie University. 2015 [cited 2020 Aug 6]. Available from: https://dalspace.library.dal.ca/bitstream/handle/10222/56328/Wan-Yun-MEC-ECMM-March-2015.pdf?sequence=3

12: Li K, Yu N, Li P, Song S, Wu Y, Li Y, et al. Multi-label spacecraft electrical signal classification method based on DBN and random forest. PLoS ONE. 2017. 12(5)

13: Lewis D. Evaluating and Optimizing Autonomous Text Classification Systems. SIGIR 95. 1995. 246–254

14: Aly M. Survey on Multiclass Classification Methods. Neural networks. 2005. 1–9

15: Read J, Pfahringer B, Holmes G, Frank E. Classifier chains for multi-label classification. ECML PKDD. 2011.

16: Zhou Z. A brief introduction to weakly supervised learning. Natl. Sci. Rev. 2017. 5(1); 44–53

17: Trohidis K, Tsoumakas G, Kalliris G, Vlahavas I. Multi-label classification of music into emotions. EURASIP. 2008. 2011(1):325-330

18: Bishop C. Pattern recognition and machine learning. 2 ed. Oxford. Oxford University Press. 2006

19: Caron M, Bojanowski P, Joulin A, Douze M. Deep Clustering for Unsupervised Learning of Visual Feature. ECCV. 2018. 11218: 139-156

20: Kaelbling L, Littman L, Moore A. Reinforcement Learning: A Survey. JAIR. 1996. 4: 237-285

21: Creswella A, Arulkumarana K, Bharatha A. On denoising autoencoders trained to minimise binary cross-entropy. Pattern Recognit. Lett. 2017.

22: Peltarion. Binary crossentropy. [cited 2020 Aug 17]; Available from: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/binary-crossentropy

23: Kaastra L, Boyd M. Designing a neural network for forecasting financial and economic time series. Neurocomputing. 1994. 10(3): 215-236

24: Shanmuganathan S, Samarasinghe S. Artificial Neural Network Modelling. Switzerland. Springer. 2016

25: Ravi D, Wong C, Deligianni F, Berthelot M, Andreu-Perez J, Lo B, et al. Deep Learning for Health Informatics. IEEE. 2017. 21(1): 4-21

26: Lee K, Cha Y, Park J. Short-term load forecasting using an artificial neural network. IEEE. 1992. 7(1): 124-132

27: Huang X, Gao L, Crosbie R, Zhang N, Fu G, Doble R. Groundwater Recharge Prediction Using Linear Regression, Multi-Layer Perception Network, and Deep Learning. Water. 2019. 11(9): 1879

28: Rios A, Kavuluru R. Convolutional Neural Networks for Biomedical Text Classification: Application in Indexing Biomedical Articles. PMC. 2017. 258–267

29: Georgakopoulos S, Tasoulis S, Vrahatis A, Plagianakos V. Convolutional Neural Networks for Toxic Comment Classification. arXiv. 2018. 1802.09957

30: Sak H, Senior A, Beaufays F. Long Short-Term Memory Recurrent Neural Network Architecturesfor Large Scale Acoustic Modeling. arXiv. 2014. 1402.1128

31: Schuster M, Paliwal K. Bidirectional Recurrent Neural Networks. IEEE. 1997. 45(11): 2673-2681

32: Sundermeyer M, Schluter R, Ney H. LSTM Neural Networks for Language Modeling. INTERSPEECH. 2010.

33: Gers F, Schraudolph N, Schmidhuber J. Learning precise timing with LSTM recurrent networks," Journal of Machine Learning Research. J MACH LEARN RES. 2003. 3(2002): 115–143

34: Biau G, Scornet E. A Random Forest Guided Tour. Springer. 2016. 25: 197–227

35: Liaw A, Wiener M. Classification and Regression by Random Forest. R News. 2002. 2(3): 18

36: Breiman L. Random Forests. Machine Learning. 2001. 45: 5-32

37: Aha D. A Lazy Approach for Machine Learning Algorithms. AIAI . 2009. 296: 517-522

38: Ling C, Wang H. Computing Optimal Attribute Weight Settings for Nearest Neighbor Algorithms. Artif Intell Rev. 1997. 11: 255–272

39: Gulli A, Pal S. Deep Learning with Keras. 1 ed.. Birmingham. Packt Publishing Limited. 2017

40: Bramer M. Principles of Data Mining. 2 ed. Berlin. Springer Science & Business Media. 2007

41: ALRashdi R, O'Keefe S. Deep Learning and Word Embeddings for Tweet Classification for Crisis Response . arXiv. 2018. 1903.11024

42: TensorFlow. Main Page. [cited 2020 July 25]; Available from: https://www.tensorflow.org/

43: TensorFlow. About. [cited 2020 Aug 20]; Available from: https://www.tensorflow.org/about

44: Sergeev A, Balso M. Horovod: fast and easy distributed deep learning in TensorFlow. arXiv. 2018. 1802.05799

45: Pandas. Main Page. [cited 2020 July 25]; Available from: https://pandas.pydata.org/

46: Natural Language Toolkit. Main Page. [cited 2020 July 25]; Available from: https://www.nltk.org/

47: Scikit-learn. Main Page. [cited 2020 July 25]; Available from: https://scikit-learn.org/

48: GitHub Archive. Main Page. [cited 2020 July 9]; Available from: https://www.gharchive.org/

49: GitHub Archive. Events. [cited 2020 July 9]; Available from: https://developer.github.com/webhooks/event-payloads/#issues

50: BigQuery. Main Page. [cited 2020 July 11]; Available from: https://cloud.google.com/bigquery

51: TensorFlow. Dropout layer. [cited 2020 July 20]; Available from: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout

52: Keras. Sequential model. [cited 2020 July 20]; Available from: https://keras.io/guides/sequential_model/

53: Provost F. Machine Learning from Imbalanced Data Sets 101. AAAI. 2000. 1-3

54: Russell S, Norvig P. Artificial Intelligence: A Modern Approach. 3 ed. London. Pearson. 2009

55: GitHub. Labels. [cited 2020 July 9]; Available from: https://docs.github.com/en/github/managing-your-work-on-github/about-labels

56: Explorable. Type Errors. [cited 2020 July 20]; Available from: https://explorable.com/type-i-error

57: DeepAI. Precision and Recall. [cited 2020 July 21]; Available from: https://deepai.org/machine-learning-glossary-and-terms/

58: Fujino A, Isozaki H, Suzuki J. Multi-label Text Categorization with Model Combination based on F1-score Maximization. IJCNLP. 2008. 2

59: Peltarion. Micro-average. [cited 2020 Aug 13]; Available from: https://peltarion.com/knowledge-center/documentation/evaluation-view/classification-loss-metrics/micro-f1-score

60: Baoli L, Shiwen Y, Qin L. An Improved k-Nearest Neighbor Algorithm for Text Categorization. Expert Syst. Appl. 2003. 39(1): 1503-1509

61: Nitze I, Schulthess U, Asche H, Comparison of Machine Learning Algorithms Random Forest, Artificial Neural Network and Support Vector Machine To Maximum Likelihood for Supervised Crop Type Classification, 2012

62: Schwartz O. In 2016, Microsoft's Racist Chatbot Revealed the Dangers of Online Conversation. IEEE Spectrum. 2019 Nov 25 [cited 2020 Aug 11]; Available from: https://spectrum.ieee.org/tech-talk/artificial-intelligence/machine-learning/in-2016-microsofts-racist-chatbot-revealed-the-dangers-of-online-conversation

63: Strubell E, Ganesh A, McCallum A. Energy and Policy Considerations for Deep Learning in NLP. ACL. 2019.

Appendices

Appendix A

Table A The table shows how many issues on GitHub had either zero or any labels assigned to them. Only issues from 2019 are considered – and the process of retrieving the data is detailed in section 3.2.

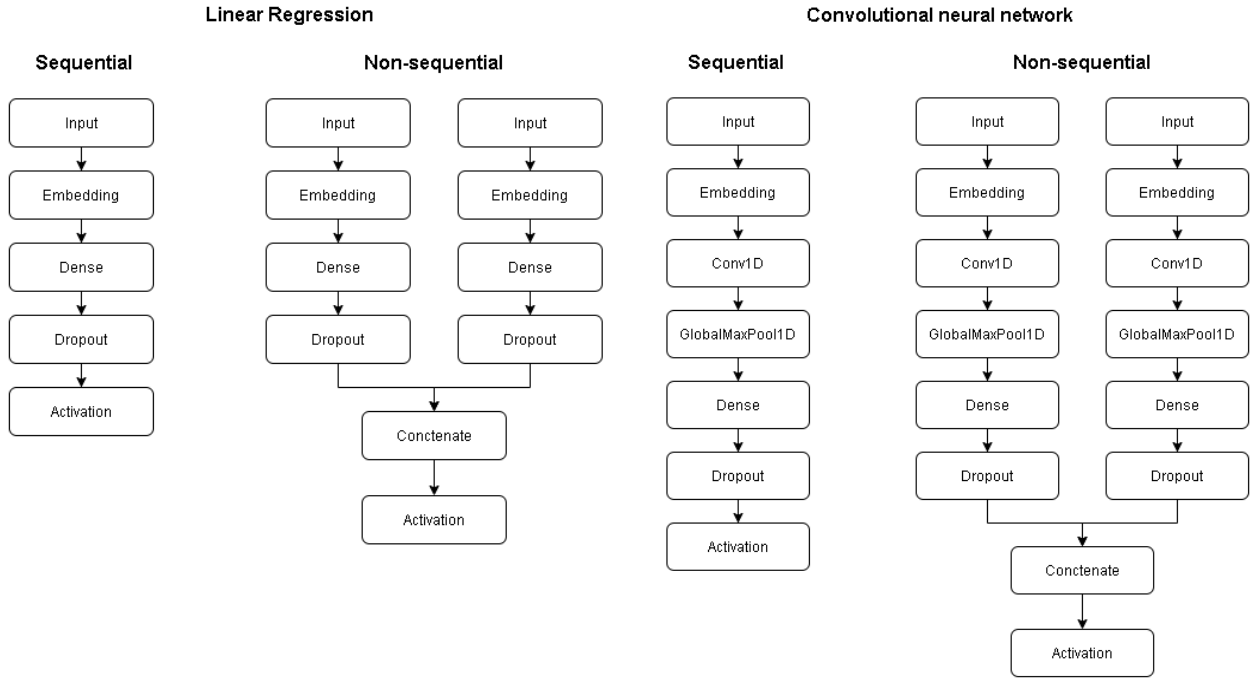
Assigned labels	**Number of issues**
None	9174543
Any	1965395
Sum	11139938

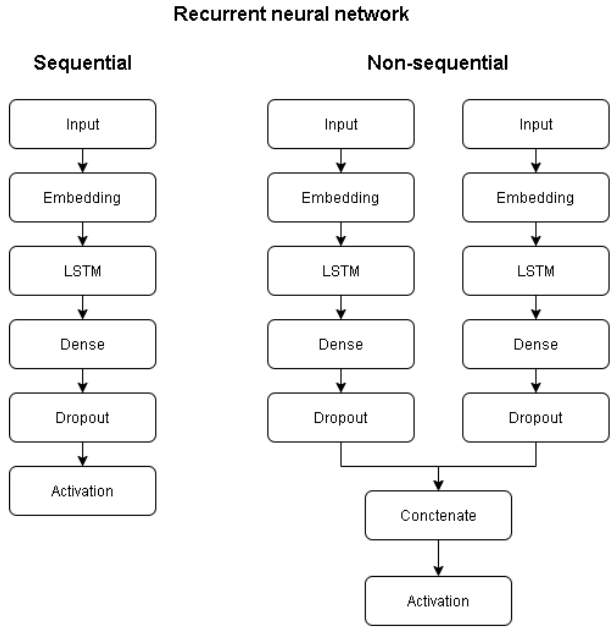
Table B The table shows the measured metrics of a coin-flip classifier. The classifier was evaluated on data set B (see Section 3.6). The classifier made predictions by flipping a coin for each label, randomly assigning true or false.

	Precision	Recall	F_1
Mean	0.2264	0.5002	0.3117
SD	0.0026	0.0071	0.0038

Appendix B

This appendix shows the architectures for all the neural networks that were used in this report.





–

PO Box 823, SE-301 18 Halmstad
Phone: +35 46 16 71 00
E-mail: registrator@hh.se
www.hh.se