Conventional Machine Learning models versus Convolutional Neural Networks in a multi-class text classification problem

Thijs Quast (LiU ID: thiqu264)*

January 9, 2020

Examiner: Marco Kuhlmann**

Course code: 732A92

Abstract

The importance of accurate text classification algorithms is straightforward, however a consensus on which Machine Learning model is superior in text classification settings remains absent. This paper aims to determine whether state-of-the art Convolutional Neural Networks (CNN) outperform conventional Machine Learning models. A wide range of models is proposed: Naive Bayes, Logistic Regression, Decision Tree, Random Forest, XGBoost and Convolutional Neural Networks. In order to identify a superior classification model, a comprehensive model evaluation is performed. Effects of different vectorizers, CountVectorizer and tf-idf vectorizer, together with balancing the training dataset is analyzed. A simple CNN architecture consisting of 9 layers is assessed, as well as a more complex design consisting of 24 layers. Results show that a more complex CNN does not improve results. Considering all models, Logistic Regression shows an outperformance when using a CountVectorizer on an unbalanced training dataset. Balancing the training data improves recall rates throughout multiple models, however, implementing the tf-idf vectorizer fails to raise performance rates. All models are trained and evaluated on a Coursera review dataset consisting of 107,018 reviews together with ratings ranging on a 1-5 scale.

Keywords: *Text mining, Text classification, Convolutional Neural Networks, Logistic Regression, Decision Tree, Random Forest, XGBoost*

* Corresponding author: Thijs Quast, Atterboms Gata 20, 58437, Linköping (SWE) - E-mail: **thijsquast@gmail.com**

** A special thank you to Marco Kuhlmann for his contributions, suggestions and guidance throughout this project. For his credentials I refer to his **website.**

Contents

1 Introduction

Modern applications of text classification algorithms vary widely across many domains, ranging from sentiment analysis to monitoring emergency situations [5]. Despite the great number of applications of text mining and the presence of extensive research on this topic, a consensus on which Machine Learning model is superior in text classification problems remains absent.

In this paper performance of a wide range of Machine Learning models in text classification is presented: Naive Bayes, Logistic Regression, Decision Tree, Random Forest, XGBoost and Convolutional Neural Networks (CNN). Through a comprehensive model evaluation, this paper aims to determine whether Convolutional Neural Networks outperform conventional Machine Learning models in a text classification problem. In order to compare model performance, a dataset consisting of 107,018 Coursera reviews together with a given course rating, ranging from 1 to 5, is used. This dataset is publicly available on *www.kaggle.com* [2].

1.1 Related work

Ting, Ip et al. [24] report an outperformance of Naive Bayes compared to Support Vector Machines, Decision Trees and Neural Networks on a dataset of 4000 documents consisting of 4 classification categories . Lee and Dernoncourt [17], however, find promising results on multiple datasets for Convolutional Neural Networks in a text classification setting. Additionally, Yoon [14] finds that a simple Convolutional Neural Network with only one convolutional layer, built on top of the pre-trained word2vec vector, shows outstanding performance.

2 Theory

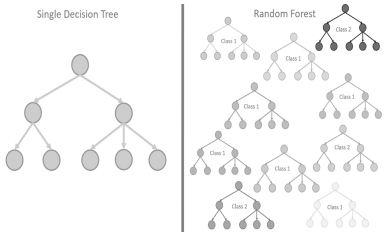
This chapter presents the theory underlying to Decision Tree, RandomForest and XGBoost models, additionally Convolutional Neural Networks are explained in detail. In order to keep the content of this chapter concise, for detailed descriptions of Logistic Regression and Naive Bayes, please see Bishop [9].

2.1 Decision Tree and Random Forest

Starting with a single tree, this section will further explain expansions of decision tree models such as Random Forest and XGBoost.

A Decision Tree model divides the training data into different leaf nodes at the bottom layer of the tree through hierarchal division. Each layer of the tree represents a binary condition, see figure 1, whereby the number of conditions present in the tree determines the complexity of the model. Although decision criteria within trees vary widely among domains, in text mining these conditions are usually based on specific words being present or absent in the text document. The training data is divided recursively until the leaf nodes have reached a minimum number of observations.

Figure 1: Decision Tree and Random Forest architecture [1]



Any previously unseen data starts at the top of the tree with the first binary split and works its way down towards the final nodes of the tree by assessing each criterion in the tree. Subsequently the tree can classify according to majority voting of observations in the leafs [7].

Because decision trees are based on a set of binary criteria and are relatively easy to visualize, the trees are fairly simple for humans to interpret. This makes decision trees a popular alternative in several Machine Learning challenges across multiple domains [9].

Building upon Decision Trees, a Random Forest model grows an ensemble of different trees by bootstrapping[1] the training data. Every document that has to be classified is run through every decision tree, and the final classification is made according through majority voting of the trees [10]. Ali, Khan et al. [8] report that Random Forest generally significantly outperform a single tree classifier.

2.2 XGBoost

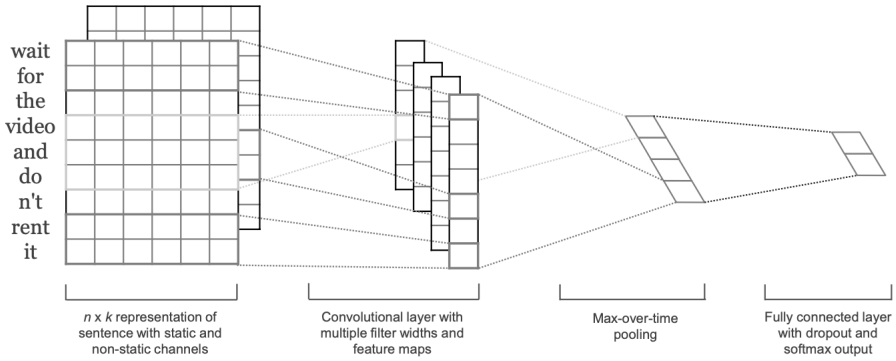
Where a Random Forest model expands on regular decision trees by creating multiple trees in parallel, an alternative possibility to improve model performance is to create decision trees in a sequential manner, through so called tree boosting. Boosting of trees aims to build trees sequentially whereby each tree learns from the misclassifications made in the previous tree.

One of such boosting algorithms is the XGBoost algorithm, which works through gradient descent boosting. Gradient descent boosting aims to reduce the loss function of the model by iteratively building decision trees that slowly move towards the minimal value of the loss function [6]. The specified learning rate determines how strongly misclassifications in previous trees are corrected in subsequent trees [18].The XGBoost algorithm has proven to be highly effective and is implemented across many Machine Learning domains [11].

2.3 Convolutional Neural Network

Contrary to previously described models, Convolutional Neural Networks, and deep learning models in general, classify unseen data in a substantially different manner. Figure 2 shows a simple Convolutional Neural Network architecture with an example sentence of length n words as input. In the first layer each word is represented as a k dimensional vector. Resulting from this, x_i represents the word vector of dimension k for word i in the sentence fed to the network.

Figure 2: Convolutional Neural Network architecture [14]



Subsequently, a convolutional layer of multiple filters applies a filter over a window of output data from the previous layer [14]. An example of a mathematical representation of a newly generated feature by one of the filters in the convolutional layer is shown below:

$$c_i = f(w \cdot x_{i:i+h-1} + b) \tag{1}$$

In which b is a bias term, w represents the filter, h is the window of word vectors x, and f is a non-linear activation function [14]. The convolutional layer convolves over each possible window of word vectors and computes a new feature c, where c is represented as:

[1]Through sampling with replacement from the training data, multiple training datasets are created. This methodology is commonly referred to as bootstrapping.

$$c = [c_1, c_2, ..., c_{n-h+1}] \tag{2}$$

Subsequently, a pooling layer captures the maximum value of c for every particular filter [14]. Adding pooling layers to the network is important as these layers retain the most important feature for every vector c [14].

The context of this paper requires the model to predict a class for each document. Through the Softmax activation function in the final layer of the model, the neural network predicts the probability of the document belonging to each of the possible classes. Naturally, the class with the highest predicted probability is the predicted class.

2.3.1 Optimization

In order to properly learn the hyper parameters in the network, through back propagation, during each epoch the network modifies the weights of the connections in the layers of the network. This minimizes the loss function in the network. The loss functions represents the distance between the output of the network and the desired output [22]. In the multi-class classification problem of this paper, the following Cross-Entropy loss function is implemented:

$$L = -\sum_{c=1}^{M} y_{o,c} ln(p_{o,c}) \tag{3}$$

Where M represents the number of classes, y obtains a value of 1 if the prediction is correct and 0 otherwise and p is the predicted probability for observation o with respect to class c [3].

In order to prevent overfitting, developers of the network may decide to add a single or multiple dropout layers to the network. A dropout layer randomly drops out a proportion of hidden units in each layer during the forward propagation phase [14], and thereby reduces the number of hidden units in the entire network. Dropping out several units in the network prevents the hidden units from adapting to each other too much [23] and therefore reduces overfitting. Srivastava [23] states that using dropout is a useful technique in reducing overfitting that has proven to be effective across many domains.

3 Data

This chapter discusses the dataset used in this paper and provides a brief summary and visualization of the data.

3.1 Coursera course review data

This project uses a dataset consisting of 107,018 course reviews from the online learning platform Coursera. Specifically, the dataset consists of an *id*, which represents the ith observation in the dataset, the *review* and a *label*. In which the *label* represents the grade given for the course by the respective student. The dataset is publicly available on Kaggle.com [2]. The first five rows of the dataset are shown below:

Id	Review	Label
0	good and interesting	5
1	This class is very helpful to me. Currently, I'm still learning this class which makes up a lot of basic music knowledge.	5
2	like!Prof and TAs are helpful and the discussion among students are quite active. Very rewarding learning experience!	5
3	Easy to follow and includes a lot basic and important techniques to use sketchup.	5
4	Really nice teacher!I could got the point eazliy but the v	4

3.2 Pre-processing

Prior to implementing a variety of Machine Learning models, the raw data is fed to a series of pre-processing steps. Firstly, the sentence is tokenized, which means that it is split into smaller chunks of text, in this case each chunk of text represents a single word. Secondly, for every chunk of text the lemma is identified.

As is commonly known, in a language, multiple grammatical forms of words share the same meaning. In text mining this is dealt with accordingly through lexemes and lemmas. Several words are part of a lexeme, which represents a set of words in different forms that share the same meaning. E.g., for *working, works, worked,* the lemma refers to a the normalized form of the lexemes, for example *work* [20]. Thirdly, chunks that contain non-alphabetical characters are removed. Fourthly, all words that are considered to be stop words are removed. Stop words are words that make the text contain more words without contributing to the essence of the text. Examples of such words are: *a, the,* and *with* [25]. An example of pre-processing on a raw review is shown below.

Raw review:
'Great course to begin coding in Python. Fully recommended..-:)'

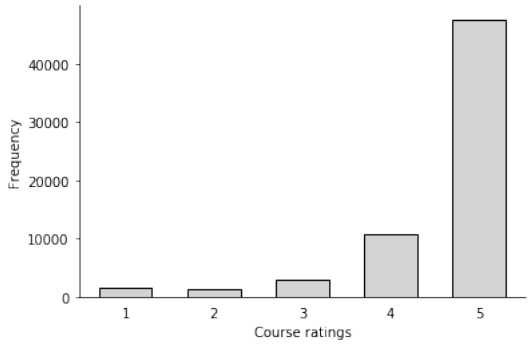
Pre-processed review:
'great course begin code Python fully recommend'

From the pre-processed sentence shown above, one can see that stop words such as *to* and *in* are removed, and *recommended* is transformed to its lemma *recommend*. Additionally, every word except the name Python is stored in lowercase letters. Possible improvements to pre-processing these reviews would be to keep characters that form emoticons, such as :) in the pre-processed review. Symbols as such are likely to indicate sentiment.

3.3 Exploratory analysis

Subsequent to pre-processing the data, the dataset is split into 60% training data, 20% validation data and 20% test data. Since the validation data and test data have to be unseen data for the models, the exploratory analysis of the data is solely performed on the training data. The distribution of the course ratings accompanied by submitted reviews is shown below:

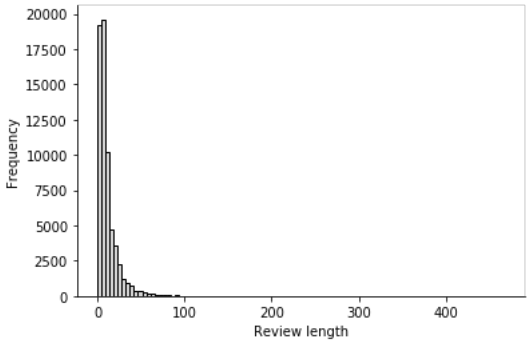
Figure 3: Histogram of pre-processed review lengths



Clearly, most users give a rating of 5, followed by ratings of 4 and 3. Above histograms indicates that writers of reviews on Coursera are generally satisfied about the course they have taken. Whether the dataset is an accurate representation of the sentiment of Coursera students is difficult to comprehend. People that are very satisfied with their experience may be more likely to write a review, whereas people that are unsatisfied might shy away from writing reviews.

Highly skewed training data can be accompanied by several challenges for classification purposes, these potential challenges will further be discussed in section 4. Following the distribution of ratings given, the distribution of pre-processed review lengths is shown in figure 4 below.

Figure 4: Histogram of pre-processed review lengths



The vast majority of pre-processed reviews is shorter than 50 words, whereby the maximum pre-processed review length is 471. Statistics on the review length either indicate that people generally write reviews that are not long or that much of the content in their reviews is of minor importance to the essence of the review content, and is therefore omitted in modeling the data.

4 Method

This chapter discusses the application of the following models on the pre-processed data: Naive Bayes, Logistic Regression, Decision Tree, Random Forest, XGBoost and Convolutional Neural Network. The mentioned steps are computed in Python version 3 using Google Colaboratory, which allows the use of GPUs for computations.

For all models discussed below, except for the Convolutional Neural Network, the pre-processed data is once transformed by the *CountVectorizer* function and once by the *TfidfVectorizer* function. Both functions come from the *scikit-learn* library [4].

The *CountVectorizer* function determines a dictionary of words, which is dependent on the entire training dataset. Each review is transformed into a vector in which each element represents the number of times a particular word occurs in the review [13].

The *TfidfVectorizer* function applies the Term Frequency Inverse Document Frequency (tf-idf) vectorization to the pre-processed data according to the following formula:

$$tf - idf(t,d) = tf(t,d) \cdot \left(log \frac{1+N}{1+df(t)} + 1 \right) \tag{4}$$

Where *tf(t,d)* represents the number of times a term t occurs in a document *d, df(t)* is equal to the number of documents that contain a term *t,* and N is the total number of documents in the dataset [4]. Intuitively, the tf-idf vectorizer measures the importance of a word in a review by comparing the frequency of word in a review compared to the frequency of that word in the whole set of reviews [13]. The training data is ran through both vectorizers in order to compare possible different effects of the type of vectorization used.

4.1 Balancing training data

As shown in section 3, the distribution of the training data is significantly skewed, the vast majority of the training observations, 74%, are labelled with rating 5. Training models on unbalanced data can produce unsatisfactory results [21]. If the validation data and the test data follow the same distribution as the training data, with a straightforward classifier that labels every instance with rating of 5, an accuracy of 74% is achieved. This can be seen in table 4 in Appendix. A common way to deal with this problem is to artificially balance the dataset by either under-sampling, in which the dataset contains observations from every class equal to the number of observations from the class with the least observations in the initial training data. Alternatively, one can oversample, in which it replicates observations from the minority classes [21]. Due to simplicity, and given that Drummond and Holte [12] find that under-sampling is more effective than oversampling, this paper follows the under-sampling approach.

All models are evaluated on the original unbalanced training data, however Logistic Regression, Decision Tree, RandomForest and XGBoost are additionally evaluated on the balanced training data. Model evaluations are based on a classification report which mentions accuracy, recall, precision and f1-score for every class as well as an average and and a weighted average for each score. This classification report is derived with the *classification_ report* function from the *scikit-learn* library.

4.2 Naive Bayes

For the Naive Bayes model, the pre-processed training data is vectorized according to the CountVectorizer. Subsequently the model is trained using the *MultinomialNB()* function, provided by the *scikit-learn* library. All function parameters are kept at their default values. The computed Naive Bayes model is the baseline model for this paper, the classification report is shown in Appendix A.

4.3 Logistic Regression

Logistic Regression is performed with the *LogisticRegression* function from the *scikit-learn* library. The model is ran with the CountVectorizer and tf-idf vectorizer, both on unbalanced as well as balanced training data. For each of the four previously mentioned setups, an exhaustive grid search over the regularization parameter c and the optimization parameter *solver* is performed [4].

4.4 Decision Tree

The Decision Tree model is fitted using the tree function from *scikit-learn*. The CountVectorizer and tf-idf vectorizer are used, on the unbalanced dataset as well as on the balanced dataset. For each setup and exhaustive grid search over the *min_ samples_ split*, *criterion* and the *max_ depth* parameter is performed. Where the previously mentioned parameters specify the minimum number of observations required for the split of an internal node, the quality of a split and the maximum depth of the tree respectively [4].

4.5 Random Forest

As an extension to the Decision Tree model, the Random Forest model is fitted using the *RandomForestClassifier* from *scikit-learn*. The Random Forest is fitted according to the CountVectorizer and tf-idf vectorizer, both on unbalanced as well as balanced training data. In each previously mentioned setup an exhaustive grid search over the *min_ samples_ split* parameter and the *criterion* parameter is performed. Both parameters represent the same features as described for the Decision Tree model.

4.6 XGBoost

In order to fit the XGBoost model in Python, the *XGBClassifier* function from the *xgboost* library is implemented. The setup covers the CountVectorizer and tf-ifd vectorizer, again for balanced and unbalanced training data. An exhaustive grid search over the *learning_ rate* parameter and the *max_ depth* parameter is implemented.

4.7 Convolutional Neural Network

Contrary to previously discussed models, for the Convolutional Neural Network, the data will not be ran through either the CountVectorizer or the tf-idf vectorizer. This is due to the required input dimensions of the first layer in a neural network.

The first 10000 words in the training data are assigned to an integer value using the Tokenizer function from the *Keras* library. Subsequently, the first layer of the neural network needs to match the dimensions of the input data. Therefore, all reviews vectors are appended with zeros until they meet the maximum review length in the training data. Thereafter, the label variable is one hot encoded and finally, using the *Sequential*, *Embedding*, *Conv1D*, *MaxPooling1D* and *Dropout* functions from *Keras*, the Convolutional Neural Network is created [19]. A simple and a more complex Convolutional Neural Network are considered, according to the architectures shown in Appendix I.

5 Results

Models are compared based on macro average f1-scores. Resulting from an exhaustive grid search over the hyper parameters over previously mentioned conventional models, Logistic Regression show the best results. The optimal Logistic Regression model has a regularization parameter equal to 1 and the *solver* parameter set to 'saga' while using the CountVectorizer on the unbalanced training data. Additionally, the simple Convolutional Neural Network architecture shows a higher f1-score than the deep complex architecture.

Table1: Logistic Regression

	precision	recall	f1-score	support
1	0.54	0.36	0.43	511
2	0.27	0.12	0.17	454
3	0.33	0.17	0.22	1021
4	0.44	0.19	0.26	3657
5	0.81	0.96	0.88	15761
accuracy			0.76	21404
macro avg	0.48	0.36	0.39	21404
weighted avg	0.70	0.76	0.72	21404

C=1, solver='saga'

Table2: Convolutional Neural Network (simple)

	precision	recall	f1-score	support
1	0.39	0.17	0.23	511
2	0.25	0.03	0.05	454
3	0.29	0.07	0.12	1021
4	0.33	0.24	0.28	3657
5	0.81	0.94	0.87	15761
accuracy			0.74	21404
macro avg	0.41	0.29	0.31	21404
weighted avg	0.68	0.74	0.70	21404

The optimal Logistic Regression model reaches an average f1-score of 39% over the 5 classes. Up to 88% f1-score is achieved for class 5, whereas only 17% f1-score is performed for class 2. Additionally, the Logistic Regression model has an average precision of 48% and an average recall of 36%, together with an overall accuracy of 76%. In contrast to the Logistic Regression model, the Convolutional Neural Network performs worse on all four classification metrics. An average f1-score of 31% is achieved, resulting from an average precision rate of 41% and a mean recall score of 29%. The convolutional neural network reaches an overall accuracy rate of 74% [2].

Since the Logistic Regression model shows superior performance, this model is evaluated on the test data. For the classification report see Appendix H. On the test data the Logistic Regression model achieves an average f1-score of 38%, resulting from a mean precision rate of 47% and a recall of 35% on average over all classes. Similar to results on the validation data, an overall accuracy of 76% is achieved.

In general, balancing the dataset increases recall rates, whilst using the tf-idf vectorizer does not significantly contribute to an improvement in classification performance.

6 Discussion

In contrast to what Ting, Ip et al. [24] and Lee and Dernoncourt [17] find, Naive Bayes does not show superior classification results and neither do different Convolutional Neural Network architectures. A simple Logistic Regression model shows the highest performance in terms of f1-score and is therefore considered to be the superior model. The optimal results are achieved when implementing an exhaustive grid search over the hyper parameters of a Logistic Regression model using the CountVectorizer and considering an unbalanced training set. Worth to mention is that although balancing the dataset does not lead to outperformance, generally it increases recall rates, see Appendices D and E.

Remarkably, both CNN architectures, despite using a dropout rate of 50% do not seem to learn properly and tend to overfit on the training data, see figure 5 in Appendix G. Resulting from this failure to learn is that both CNNs fail to beat the Naive Bayes baseline model. Important to mention is that when comparing conventional models implemented through the *scikit-learn* library and convolutional neural networks through Keras, different pre-processing steps are required. Where the conventional models feed the data in a vectorized form, either CountVectorized or tf-idf vectorized, Neural Networks require a different input form as mentioned in section 4. Additionally, the Convolutional Neural Networks have an embedding layer in the model.

[2]Improvements to the CNN design are desirable, a practitioner on Kaggle.com manages to reach an accuracy of 80% under a simple CNN architecture without pre-processing data. [19]

A limitation of this paper is that it does not consider running times of the different algorithms. On a relatively small dataset as in this paper, differences in running times are neglectable, however in model selection for real life purposes the tradeoff between running time and performance cannot be ignored. Many possibilities to expand and improve this research are present. It is relevant is to assess the classification results for Convolutional Neural Networks on a much larger dataset, as it is commonly known that deep learning models tend to perform better when having large amounts of data available. Additionally, Krizhevsky et al. [15], show that Convolutional Neural Networks clearly outperform other models on a dataset of 1.2 million observations in an image recognition problem. A more feasible way to train models on more data would be to suggest a different train test validation split in which more of the data is used for training.

Naturally, one could suggest different models or a combination of models. Lai, Xu et al. [16] introduce a Recurrent Convolutional Neural Network that outperforms CNNs and RNNs on four different datasets. Also, Zhou, Sun et al. [26] combine a Convolutional Neural Network and a long short-term memory network (LSTM), into a C-LSTM model which outperforms both CNN and LSTM.

7 Conclusion

This paper confirms that more complex models do not consistently outperform more simple conventional models. Considering a vast amount of models under many different setups, a more comprehensive CNN fails to beat a classical Logistic Regression model. Therefore this paper is unable to find evidence that Convolutional Neural Networks outperform conventional Machine Learning models in a text classification problem. Important to mention is that these results are solely based on the Coursera Review dataset and therefore generalizing results based on this paper only would be too simplistic. More research on this topic as suggested in section 6 is therefore required.

8 Appendix

Appendix A - Baseline

Table 3: Naive Bayes

	precision	recall	f1-score	support
1	0.67	0.21	0.32	511
2	0.13	0.01	0.02	454
3	0.27	0.09	0.13	1021
4	0.38	0.25	0.30	3657
5	0.81	0.95	0.88	15761
accuracy			0.75	21404
macro avg	0.45	0.30	0.33	21404
weighted avg	0.70	0.75	0.71	21404

Table 4: Most Frequent Class

	precision	recall	f1-score	support
1	0.00	0.00	0.00	511
2	0.00	0.00	0.00	454
3	0.00	0.00	0.00	1021
4	0.00	0.00	0.00	3657
5	0.74	1.00	0.85	15761
accuracy			0.74	21404
macro avg	0.15	0.20	0.17	21404
weighted avg	0.54	0.74	0.62	21404

Appendix B - CountVectorized Unbalanced

Table 5: Logistic Regression

	precision	recall	f1-score	support
1	0.54	0.36	0.43	511
2	0.27	0.12	0.17	454
3	0.33	0.17	0.22	1021
4	0.44	0.19	0.26	3657
5	0.81	0.96	0.88	15761
accuracy			0.76	21404
macro avg	0.48	0.36	0.39	21404
weighted avg	0.70	0.76	0.72	21404

c=1, solver='saga'

Table 6: Decision Tree

	precision	recall	f1-score	support
1	0.28	0.27	0.28	511
2	0.17	0.11	0.13	454
3	0.18	0.12	0.15	1021
4	0.29	0.21	0.24	3657
5	0.80	0.88	0.84	15761
accuracy			0.70	21404
macro avg	0.34	0.32	0.33	21404
weighted avg	0.66	0.70	0.68	21404

split = 40, criterion='entropy', max_depth=None

Table 7: RandomForest

	precision	recall	f1-score	support
1	0.52	0.11	0.18	511
2	0.13	0.00	0.01	454
3	0.28	0.02	0.04	1021
4	0.42	0.10	0.16	3657
5	0.76	0.98	0.86	15761
accuracy			0.75	21404
macro avg	0.42	0.24	0.25	21404
weighted avg	0.66	0.75	0.67	21404

split = 40, criterion = 'gini'

Table 8: XGBoost

	precision	recall	f1-score	support
1	0.54	0.30	0.39	511
2	0.26	0.06	0.10	454
3	0.29	0.10	0.15	1021
4	0.42	0.20	0.27	3657
5	0.80	0.96	0.88	15761
accuracy			0.76	21404
macro avg	0.46	0.33	0.36	21404
weighted avg	0.69	0.76	0.71	21404

learning rate = 0.1, max_depth = 0.64

Appendix C - Tf-idf Unbalanced

Table 9: Logistic Regression

	precision	recall	f1-score	support
1	0.57	0.33	0.42	511
2	0.29	0.07	0.11	454
3	0.33	0.15	0.21	1021
4	0.43	0.20	0.28	3657
5	0.81	0.96	0.88	15761
accuracy			0.76	21404
macro avg	0.49	0.34	0.38	21404
weighted avg	0.70	0.76	0.72	21404

c=1.5, solver = newton-cg

Table 10: Decision Tree

	precision	recall	f1-score	support
1	0.25	0.23	0.24	511
2	0.14	0.10	0.12	454
3	0.18	0.13	0.15	1021
4	0.27	0.23	0.25	3657
5	0.80	0.85	0.83	15761
accuracy			0.68	21404
macro avg	0.33	0.31	0.32	21404
weighted avg	0.65	0.68	0.67	21404

split = 20, criterion = entropy max_depth = None

Table 11: RandomForest

	precision	recall	f1-score	support
1	0.41	0.13	0.19	511
2	0.16	0.02	0.04	454
3	0.22	0.06	0.09	1021
4	0.35	0.14	0.20	3657
5	0.78	0.96	0.86	15761
accuracy			0.74	21404
macro avg	0.38	0.26	0.28	21404
weighted avg	0.66	0.74	0.68	21404

default parameters

Table 12: XGBoost

	precision	recall	f1-score	support
1	0.55	0.29	0.38	511
2	0.29	0.06	0.09	454
3	0.28	0.09	0.14	1021
4	0.42	0.19	0.26	3657
5	0.80	0.97	0.88	15761
accuracy			0.76	21404
macro avg	0.47	0.32	0.35	21404
weighted avg	0.69	0.76	0.71	21404

Appendix D - CountVectorized balanced

Table 13: Logistic Regression

	precision	recall	f1-score	support
1	0.23	0.52	0.32	511
2	0.11	0.31	0.16	454
3	0.17	0.35	0.23	1021
4	0.25	0.38	0.30	3657
5	0.88	0.63	0.74	15761
accuracy			0.57	21404
macro avg	0.33	0.44	0.35	21404
weighted avg	0.71	0.57	0.62	21404

default parameters

Table 14: Decision Tree

	precision	recall	f1-score	support
1	0.11	0.52	0.18	511
2	0.07	0.31	0.11	454
3	0.11	0.30	0.16	1021
4	0.20	0.28	0.23	3657
5	0.86	0.50	0.63	15761
accuracy			0.45	21404
macro avg	0.27	0.38	0.26	21404
weighted avg	0.68	0.45	0.52	21404

split = 10 criterion = entropy max_depth = None

Table 15: RandomForest

	precision	recall	f1-score	support
1	0.16	0.52	0.24	511
2	0.08	0.28	0.12	454
3	0.14	0.29	0.19	1021
4	0.24	0.28	0.26	3657
5	0.87	0.64	0.74	15761
accuracy			0.55	21404
macro avg	0.30	0.40	0.31	21404
weighted avg	0.69	0.55	0.60	21404

split = 30 criterion = gini

Table 16: XGBoost

	precision	recall	f1-score	support
1	0.14	0.60	0.23	511
2	0.10	0.33	0.16	454
3	0.17	0.25	0.20	1021
4	0.24	0.35	0.29	3657
5	0.88	0.61	0.72	15761
accuracy			0.54	21404
macro avg	0.31	0.43	0.32	21404
weighted avg	0.70	0.54	0.60	21404

learning_rate = 0.2 max_depth = 4

Appendix E - Tf-idf balanced

Table 17: Logistic Regression

	precision	recall	f1-score	support
1	0.19	0.57	0.28	511
2	0.11	0.38	0.18	454
3	0.15	0.36	0.21	1021
4	0.25	0.32	0.28	3657
5	0.89	0.63	0.74	15761
accuracy			0.56	21404
macro avg	0.32	0.45	0.34	21404
weighted avg	0.71	0.56	0.61	21404

C = 0.5 Solver = newton-cg

Table 18: Decision Tree

	precision	recall	f1-score	support
1	0.12	0.47	0.19	511
2	0.06	0.34	0.11	454
3	0.10	0.23	0.13	1021
4	0.17	0.19	0.18	3657
5	0.84	0.55	0.67	15761
accuracy			0.47	21404
macro avg	0.26	0.36	0.26	21404
weighted avg	0.66	0.47	0.54	21404

split = 50 criterion = entropy max_depth = None

Table 19: RandomForest

	precision	recall	f1-score	support
1	0.13	0.51	0.21	511
2	0.09	0.36	0.14	454
3	0.13	0.26	0.17	1021
4	0.23	0.26	0.24	3657
5	0.87	0.62	0.72	15761
accuracy			0.54	21404
macro avg	0.29	0.40	0.30	21404
weighted avg	0.69	0.54	0.59	21404

split = 40 criterion = gini

Table 20: XGBoost

	precision	recall	f1-score	support
1	0.14	0.58	0.22	511
2	0.09	0.34	0.14	454
3	0.15	0.30	0.20	1021
4	0.23	0.32	0.27	3657
5	0.88	0.58	0.70	15761
accuracy			0.52	21404
macro avg	0.30	0.42	0.31	21404
weighted avg	0.70	0.52	0.58	21404

learning_rate = 0.25 max_depth = 4

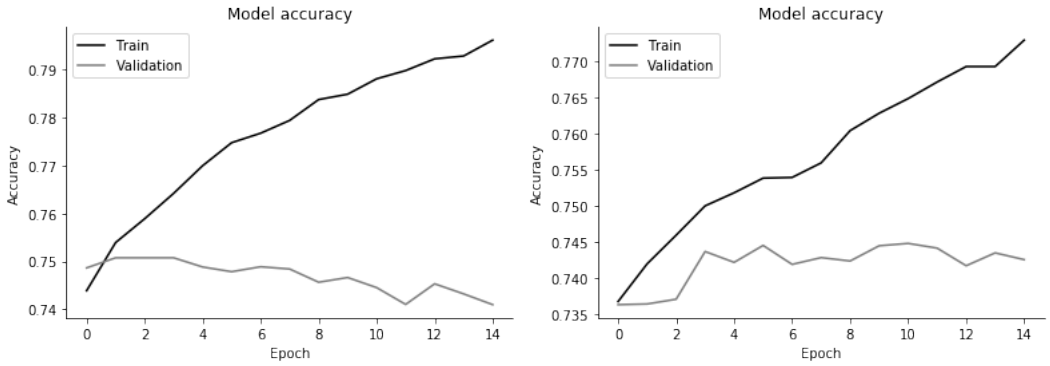
Appendix F - Performance complex CNN architecture

Table 21: CNN complex

	precision	recall	f1-score	support
1	0.75	0.02	0.03	511
2	0.00	0.00	0.00	454
3	0.00	0.00	0.00	1021
4	0.27	0.15	0.19	3657
5	0.79	0.97	0.87	15761
accuracy			0.74	21404
macro avg	0.36	0.23	0.22	21404
weighted avg	0.65	0.74	0.68	21404

Appendix G - CNN learning

Figure 5: Learning of simple (left) and complex (right) Convolutional Neural Networks



Appendix H - Test Data

Table 22: Logistic Regression on test data

	precision	recall	f1-score	support
1	0.49	0.37	0.42	459
2	0.29	0.09	0.14	468
3	0.33	0.15	0.21	1011
4	0.46	0.19	0.26	3633
5	0.81	0.97	0.88	15832
accuracy			0.76	21403
macro avg	0.47	0.35	0.38	21403
weighted avg	0.71	0.76	0.72	21403

C=1, solver='saga'

Appendix I - CNN architectures

Simple CNN architecture

Layer	Parameters
Embedding	input_dim = 10000, output_dim =200, input_length = 471
Conv1D	filters = 200, kernel_size = 2, padding = 'same', activation = 'relu'
MaxPooling	pool_size =2
Dropout	rate=0.5
Conv1D	filters=8, kernel_size = 2, padding='same', activation ='relu'
MaxPooling	pool_size=2
Dropout	rate=0.5
Flatten	
Dense	units = 5, activation = 'softmax
Compile	loss='categorical_crossentropy', optimizer='adam', metrics='accuracy'

Complex CNN architecture

Layer	Parameters
Embedding	input_dim = 10000, output_dim =1024, input_length = 471
Conv1D	filters = 1024, kernel_size = 2, padding = 'same', activation = 'relu'
MaxPooling	pool_size =2
Dropout	rate=0.5
Conv1D	filters=512, kernel_size = 2, padding='same', activation ='relu'
MaxPooling	pool_size=2
Dropout	rate=0.5
Conv1D	filters = 256, kernel_size = 2, padding = 'same', activation = 'relu'
MaxPooling	pool_size =2
Dropout	rate=0.5
Conv1D	filters = 128, kernel_size = 2, padding = 'same', activation = 'relu'
MaxPooling	pool_size =2
Dropout	rate=0.5
Conv1D	filters = 64, kernel_size = 2, padding = 'same', activation = 'relu'
MaxPooling	pool_size =2
Dropout	rate=0.5
Conv1D	filters = 32, kernel_size = 2, padding = 'same', activation = 'relu'
MaxPooling	pool_size =2
Dropout	rate=0.5
Conv1D	filters = 15, kernel_size = 2, padding = 'same', activation = 'relu'
MaxPooling	pool_size =2
Dropout	rate=0.5
Flatten	
Dense	units = 5, activation='softmax'
Compile	loss='categorical_crossentropy', optimizer='adam', metrics='accuracy'

References

[1] From a single decision tree to a random forest, https://towardsdatascience.com/from-a-single-decision-tree-to-a-random-forest-b9523be65147.

[2] Kaggle, https://www.kaggle.com/septa97/100k-courseras-course-reviews-dataset.

[3] Loss functions: Cross entropy, https://ml-cheatsheet.readthedocs.io.

[4] Sci-kit learn, https://scikit-learn.org/stable/.

[5] Text classification, applications and use cases: https://towardsdatascience.com/text-classification-applications-and-use-cases-beab4bfe2e62.

[6] Xgboost, https://towardsdatascience.com/machine-learning-part-18-boosting-algorithms-gradient-boosting-in-python-ef5ae6965be4.

[7] Charu C Aggarwal and ChengXiang Zhai. *Mining text data*. Springer Science & Business Media, 2012.

[8] Jehad Ali, Rehanullah Khan, Nasir Ahmad, and Imran Maqsood. Random forests and decision trees. *International Journal of Computer Science Issues (IJCSI)*, 9(5):272, 2012.

[9] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[10] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[11] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[12] Chris Drummond, Robert C Holte, et al. C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. In *Workshop on learning from imbalanced datasets II*, volume 11, pages 1–8. Citeseer, 2003.

[13] Raúl Garreta and Guillermo Moncecchi. *Learning scikit-learn: machine learning in python*. Packt Publishing Ltd, 2013.

[14] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

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

[16] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[17] Ji Young Lee and Franck Dernoncourt. Sequential short-text classification with recurrent and convolutional neural networks. *arXiv preprint arXiv:1603.03827*, 2016.

[18] Andreas C Müller, Sarah Guido, et al. *Introduction to machine learning with Python: a guide for data scientists*. " O'Reilly Media, Inc.", 2016.

[19] Anindita Pani. https://www.kaggle.com/aninditapani/nlp-with-mlp-cnn-and-glove.

[20] Joël Plisson, Nada Lavrac, Dunja Mladenic, et al. A rule based approach to word lemmatization. *Proceedings of IS-2004*, pages 83–86, 2004.

[21] Foster Provost. Machine learning from imbalanced data sets 101. In *Proceedings of the AAAI'2000 workshop on imbalanced data sets*, volume 68, pages 1–3. AAAI Press, 2000.

[22] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[23] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[24] SL Ting, WH Ip, and Albert HC Tsang. Is naive bayes a good classifier for document classification. *International Journal of Software Engineering and Its Applications*, 5(3):37–46, 2011.

[25] S Vijayarani, Ms J Ilamathi, and Ms Nithya. Preprocessing techniques for text mining-an overview. *International Journal of Computer Science & Communication Networks*, 5(1):7–16, 2015.

[26] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis Lau. A c-lstm neural network for text classification. *arXiv preprint arXiv:1511.08630*, 2015.