Imperial College London

Imperial College London

Department of Computing

An Evaluation of Machine Learning Approaches to Natural Language Processing for Legal Text Classification

Author:
Clavance Lim

Supervisors:
Prof Alessandra Russo
Nuri Cingillioglu

Submitted in partial fulfillment of the requirements for the MSc degree in Computing Science of Imperial College London

September 2019

Contents

Abstract

This project provides a comprehensive comparative study of the performance of supervised machine learning models in the natural language processing task of text classification, specifically in the legal context. We distill a dataset of European Union legislation for multi-label classification into one for a single-label, multi-class classification task. We provide visualisations and analysis of the dataset. We then draw a distinction between 'machine learning' models, including the Naive Bayes classifier, logistic regression and support vector machines, and more contemporary 'deep learning' approaches, such as convolutional neural networks, long short-term memory networks and the hierarchical attention network. We experiment with traditional count-based vectorizers for feature embedding with the machine learning models, and pre-trained word embeddings for the deep learning models. We critically evaluate the performance of each model on its own, and with those in its group, before proposing a final model. Finally, we discuss the potential uses of such a classifier in professional legal practice.

Acknowledgements

I would like to express my utmost gratitude to the following people, without whom this project would not have been possible:

- Professor Alessandra Russo and Nuri Cingillioglu, for their continued guidance, encouragement and time throughout the course of this project.
- My family, for everything.

Chapter 1

Introduction

1.1 Motivation

The application of technology to assist legal professionals with the provision of legal services, a sector known as *legal tech*, has received tremendous investment and interest in recent years [29]. In particular, with the recent successes of *machine learning* methods in fields such as computer vision and pattern recognition, expectations that these methods will provide the panacea for the ills of the legal profession, such as repetitive administrative work, have begun to arise [66].

The broad aim of this project is thus to apply the latest methods used in machine learning and in *natural language processing* (NLP) to a dataset in the legal context. More specifically, the goal will be to experiment with and compare the performance of several machine learning and *deep learning* methods for the task of *text classification*. Text classification has many potential uses in the legal domain, particularly for categorising legal documents and cases which can aid the process of legal research, and for the development of a knowledge management system (for a detailed example of such an implementation, see [5, 6]).

The task is an interesting one from an academic perspective, for several reasons. While text classification as an NLP task in general is well-studied, the specific study of text classification methods in the legal domain has remained relatively under-explored [28, 67]. Applying text classification methods specifically to the legal context is not a trivial problem, i.e. simply because a method has proven to be useful in classifying texts of a general subject matter does not mean that the method will necessarily work equally well in the legal context. This is because the structure of legal language can be distinguished from that of ordinary language in terms of vocabulary, syntax, semantics and other linguistic features [5, 65]. The types of texts used in NLP research tend to be user reviews, where the language used tends to be colloquial or informal (such as the IMDB dataset [45]), posts scraped from Twitter (which are a maximum of 280 characters) and other documents of a much shorter length than a legal judgement or piece of legislation [28].

1.2 Aims and objectives

The broad aim of this project is to present a framework through which a document in the English language with legal subject matter can be classified into one of several predefined classes. Specifically, documents will be drawn from a dataset of European Union legislation, with each document belonging to one of 20 classes. Concretely, the goal will be to propose a classification model. Given an unseen legal text document of length n, $X = (x_1, x_2, ..., x_n)$, where x_i is an individual token in the document, the model will assign X to one of k classes, where $k = 20$ in our case.

This aim will be achieved by fulfilling the following objectives:

- Extracting relevant sections of legal texts from their raw HTML source obtained from a publicly accessible European Union law repository
- Preprocessing the unstructured data to a structured format
- Analysing the characteristics of the dataset (e.g. distribution of classes)
- Exploring different methods of evaluating the performance of classifiers
- Drawing a distinction between two groups of classifiers, 'machine learning' methods and 'deep learning' methods, and analysing each group separately, with different methods of feature extraction:
  - Classifiers based on various machine learning methods, with count vectorization and TF-IDF vectorization:
    1. Naive Bayes classifier
    2. Decision tree
    3. Random forest
    4. Logistic regression
    5. Support vector machines (SVMs)
    6. K-nearest neighbours (k-NN)
    7. Multilayer perceptron (MLP)
  - Comparing the performance of classifiers based on deep learning methods, with pre-trained word and sentence embeddings:
    1. Multilayer perceptron
    2. Convolutional neural network (CNN)
    3. Recurrent neural network (RNN) (specifically, uni-directional and bi-directional long short-term memory networks (LSTM and bi-LSTM))
    4. Hierarchical attention network (HAN)
- Concluding the project by evaluating the utility of such a model in practice

1.3 Outline

The remainder of this report is organised as follows:

- Chapter 2 provides an overview of the theoretical background of the project. This includes an introduction of the text classification task, the experimental setup, and the metrics used to evaluate classifiers. It outlines the standard text classification pipeline and discusses the methods of feature extraction used in this project. Each model used in this project is then introduced in the context of the current project.

- Chapter 3 provides an analysis of the dataset which will be used in this project, a distilled form of the EURLEX dataset. It describes the repository from which the dataset was scraped, steps taken to extract the data from its raw source and to reshape the data into the form necessary for a single-label, multi-class classification task. It also discusses the features of the dataset and provides visualisations of the dataset.

- Chapter 4 describes the experiments undertaken in this project. It first discusses the steps taken to preprocess the data. Evaluation of classifiers is then separated into 'machine learning' and 'deep learning' models, with each model considered individually and also compared against the other classifiers within its group. The machine learning models are run with count-based vectorizers for feature extraction, and the deep learning models are run with pre-trained embeddings.

- Chapter 5 discusses academic work related to this project. It considers the latest academic work in NLP, in text classification, and finally, in the legal context.

- Chapter 6 concludes this report by summarising its academic contributions. It then evaluates the effectiveness in practice of a model such as the best performing model proposed. It then discusses the challenges faced, the ways in which this project could have been improved on hindsight, and future work which can be done. Finally, it considers the legal and ethical implications of implementing this project.

- The Appendices contain a completed ethics checklist, and a copy of the readme which accompanies the code archive.

Chapter 2

Background

2.1 Overview

2.1.1 Text classification

In general, and for present purposes, text classification is a *supervised learning* task. The aim in supervised learning is to construct a model which, given a *training set* of English language legal documents, learns a function, *f*, that can map an input, X to an output, C_i:

$$f : X \rightarrow C_i, \ f \in H \tag{2.1}$$

The function f will be one of multiple possible mapping functions which the model can learn from the set of all possible functions, H, the *hypothesis space*. The model finds this function f by changing the *parameters*, θ, of the functions within H, and finding the function which best minimises some expected *loss function*, *L(x; f)*. The expected output C_i can be binary (one of two possible values, such as 'positive' or 'negative'), multi-class (only one of several possible values), or multi-label (one or more of several possible values). The present task is a *multi-class classification task*.

In the present case, the model takes as input a set of N text documents from a dataset D, where $D = \{X_1, X_2, ..., X_N\}$. X_i is a single document consisting of one or more sentences. Each sentence is made up of one or more *tokens* (see Section 2.2). Each token is made up of one or more characters. Each document X_i belongs to one and only one of k possible classes, $C = \{C_1, C_2, ..., C_k\}$. Having learned the function f which can map an input, $X_i \in D$, to an output, $C_i \in C$, the model can then be used to classify documents, and in particular, unseen documents from a *test set*.

2.1.2 Training, validation and test sets

In a supervised learning task, the dataset has to be divided into a training set and a test set. The model will be trained on the training set, the process during which it learns the mapping function *f*, and its performance will be evaluated and reported on the unseen test set. During the training phase, the progress of the model being trained should also be evaluated, to ensure that the model is not learning patterns specific to

the training data which cannot be observed in unseen data, a problem called *overfitting* [64].

However, this evaluation must be done without exposing the model to the test set. If such exposure has occurred, the test set will no longer be considered unseen data, and any evaluation on the test set will no longer be accurate, a problem which has been called *data leakage*. To avoid this but also achieve the aim of evaluating the model during training, a subset of the training set is set aside as the *validation set* [64]. The *validation loss*, an estimate of the *error rate* of the model computed from the loss function over the validation set, is monitored throughout the training process (on evaluation metrics, see Section 2.1.5).

While there has been research to estimate the optimum number of documents to be set aside in a test set and a validation set [24], for simplicity, we follow a rule-of-thumb and divide our dataset D by an 80:20 split, setting aside 20% of the training set as the test set. From the 80% training set, we set aside a 20% of the data from the training set to be used as a validation set. Thus:

$$number\ of\ samples\ in\ D = N$$
$$number\ of\ samples\ in\ training\ set\ (excluding\ validation\ set) = N * (0.8) * (0.8)$$
$$number\ of\ samples\ in\ validation\ set = N * (0.8) * (0.2)$$
$$number\ of\ samples\ in\ test\ set = N * (0.2)$$

2.1.3 Cross validation

Other than to prevent overfitting, the validation set is also used to determine which *hyperparameters* of a classifier result in the best performance on the test set (see Section 2.1.4). However, one problem with setting aside a validation set is that the validation loss may not provide a fair estimate of performance on the test set, depending on which specific samples are included in the training set and the validation set [34]. One method to alleviate this is *k-fold cross validation*, where the dataset is divided into k subsets of equal size. The divisions are often *stratified*, i.e. the proportions of samples belonging to each class are the same in each subset. The first subset is held out as the validation set, and the model is then fit on the other *k-1* subsets. This process is repeated k times, until all the subsets have each been held out as a validation set. Thus:

$$CV_{(k)} = \frac{1}{k} \sum_{i=1}^{k} E_i \tag{2.2}$$

where E represents the error rate, or any other performance metric being calculated (discussed in Section 2.1.5). This results in k estimates of the relevant metric E. The value for E obtained all k folds is then averaged, and this value is taken as an estimate of the performance on the training set. A common value used is k=10. However, because the number of samples in our least populated class, Class 20, is only 6 (see

Table 3.4), stratified 10-fold cross validation is not suitable. Thus, stratified 5-fold cross validation will be used instead.

Thus, k-fold cross validation attempts to reduce any statistical bias which may result from features unique to the samples in the validation set, affecting the effectiveness of the validation loss as an estimate of the error rate on the test set [34]. Of course, the obvious disadvantage is that instead of computation being performed once, it has to be performed k times.

2.1.4 Hyperparameter optimization

As noted above, the ultimate aim of a model is to learn the function f by optimising parameters, θ, to minimise the loss function L. However, the model itself has certain hyperparameters, λ, which are specific hyperparameters in the set of all possible hyperparameters, the search space Λ, i.e. $\lambda \in \Lambda$. λ is chosen before the training of the model, and is not learned during the training process. The problem of choosing good values for λ is termed *hyperparameter optimization* [3]. To find the best hyperparameters, the performance of each set of hyperparameters λ on the validation sets, averaged across k-fold cross validation, is measured.

2.1.4.1 Manual search and grid search

There are several methods of finding the best hyperparameters within Λ. The most common option is *manual search*, which is a trial-and-error method. This entails running classifiers based on manually selected hyperparameters, and choosing the hyperparameters which work best across k-fold cross validation. Between the three methods detailed here, this is perhaps the least rigorous, and as Bergstra and Bengio [3] note, choosing hyperparameters based on intuition leads to difficulties with reproducing results. However, due to time and computational constraints, this method is partly used here.

Another option is *grid search*, which is effectively an exhaustive search of every possible value of λ through a manually chosen subset of Λ. This is often combined with manual search, to identify regions within Λ that are promising and to perform a brute force search within those regions. However, Bergstra and Bengio [3] note that while grid search is reliable in low-dimensional (e.g. 1-D or 2-D) spaces, it suffers from the *curse of dimensionality* as the number of possible values grows exponentially with the number of hyperparameters to be tuned. Thus, a key drawback of grid search is the amount of computation involved. This is exactly the problem that may be faced here, given the number of classifiers being experimented with, and the number of hyperparameters in some of the models.

2.1.4.2 Random search

As an alternative to grid search, Bergstra and Bengio [3] propose *random search*, which, in their experimentation, is able to find hyperparameters which are as good as or better

within a much smaller computation time. Instead of an exhaustive search of Λ, random values of λ are sampled and tried. If the search space of possible hyperparameters is large, this is the method of hyperparameter optimization which will be used here.

2.1.5 Evaluation metrics

2.1.5.1 Classification rate

Once a model has been selected and trained, it is necessary to evaluate how effective the model is for the purposes of the classification task. Intuitively, one way to evaluate a model is to calculate the percentage of samples it has classified correctly (i.e. placed within the correct class). Thus, we can calculate the *accuracy* or *classification rate* of the model:

$$classification\ rate = \frac{number\ of\ samples\ correctly\ classified}{total\ number\ of\ samples} \tag{2.3}$$

Similarly, we can calculate the *classification error* as the complement of the classification rate. However, the classification rate is not always the most relevant metric, especially where the dataset is imbalanced [69], as in the current case (see Section 3.3.1). A common example given in the text classification context is where an email spam-filtering model is designed to classify emails as spam or not-spam, but only 1% of the samples in the dataset are labelled as spam. In this case, the model would achieve a classification rate of 99% by simply predicting all unseen samples as not-spam, but it would have not learned which features are indicative of an unseen sample being spam or not-spam at all.

2.1.5.2 Confusion matrix

Given the above, a confusion matrix is often used to evaluate a model's performance as well. This is a (k x k) matrix, where k is the total number of unique classes the samples in the dataset belong to. The confusion matrix is a visualisation of the counts of the predicted class of each sample, compared to their actual class.

For the purposes of illustration, an example of a (4x4) matrix is provided. In this case, the purpose of the model would be to classify samples into one of four possible classes. The confusion matrix can be used as a high-level overview of the performance of a model. The diagonal across the confusion matrix would provide the counts of the samples the model has labelled correctly (i.e. True Positives, as explained below) for each class:

		Predicted class			
		1	**2**	**3**	**4**
Actual class	**1**	TP			
	2		TP		
	3			TP	
	4				TP

Table 2.1: Correct classifications in a confusion matrix

Thus the overall classification rate of the model can be calculated by:

$$classification\ rate = \frac{\sum_{i=1}^{k} TP_i}{total\ number\ of\ samples} \tag{2.4}$$

recalling that k is the total number of unique classes, and TP_i is the count of True Positives for Class i. However, as explained above, computing an overall performance of a model may not always be the most useful metric. Thus, the confusion matrix should also be used to examine the model's performance for each specific class. Continuing with the example above, here the focus is on analysing the model's performance in relation to its ability to correctly predict samples belonging to Class 2. Each sample being classified by the model can fall into one of four possible outcomes:

- *True Positive (TP)*: the label predicted by the model is the same as the actual label of the sample
- *True Negative (TN)*: the model predicted that sample does not belong to Class 2, and it is true that the sample does not belong to Class 2
- *False Positive (FP)*: the model predicted that the sample belongs to Class 2, but the sample does not
- *False Negative (FN)*: the model predicted that the sample does not belong Class 2, but the sample belongs to Class 2

		Predicted class			
		1	**2**	**3**	**4**
Actual class	**1**	TN	FP	TN	TN
	2	FN	TP	FN	FN
	3	TN	FP	TN	TN
	4	TN	FP	TN	TN

Table 2.2: Example of a confusion matrix (focusing on Class 2)

The four outcomes above allow us to calculate several important metrics, as noted in the following section. The confusion matrix can also shed light into the specific misclassifications by the model, for example, the model may be specifically classifying samples belonging to Class 4 as those belonging to Class 1. A normalised confusion

matrix can also be provided, where each box reflects the number of predictions as a fraction of the number of samples labelled as belonging to that class. The confusion matrix thus provides a useful tool to analyse the performance of a model, and this will be done for the current project. In the distilled EURLEX dataset, the data is labelled into one of 20 possible classes (see Section 3.3.1), thus the confusion matrices provided will be of dimension (20x20).

2.1.5.3 Precision, recall and F-measure

From the confusion matrices, a model's *precision* can be calculated:

$$precision = \frac{TP}{TP + FP} \tag{2.5}$$

where a perfect score of 1 indicates that every sample the model identified as belonging to a specific class did in fact belong to that specific class (but does not reveal how many other samples also belonging to that class were misclassified by the model).

Similarly, *recall* can be calculated:

$$recall = \frac{TP}{TP + FN} \tag{2.6}$$

which represents the model's ability to detect the presence of samples belonging to a specific class, i.e. a perfect score of 1 would indicate that the model correctly classified all the samples belonging to that class (but does not reveal how many samples the model incorrectly identified as belonging to that class).

In general, the usefulness of precision and recall as metrics is context-dependent, i.e. in some tasks, having a high precision would be more important, and in others recall may be more crucial. Models can be tuned for either metric, depending on the use case, but optimising for one will result in a trade-off in the other. In this project, we will not choose to optimise models for either precision or recall in particular. As noted by Manning and Schutze [46], the trade-off between precision and recall may not make as much sense in some NLP applications as opposed to other contexts.

As such, we will also report the F1-measure, which is a measure of performance between 0 and 1 calculated from the harmonic mean of precision and recall (where a value of 1 indicates perfect precision and recall):

$$F_\alpha = (1 + \alpha^2)\ \frac{precision * recall}{precision + recall} \tag{2.7}$$

This is useful insofar as it provides a single number representing both precision and recall. The α-value can also be varied, where a higher α indicates greater emphasis on recall.

2.1.5.4 Macro vs. micro-averaging

Precision, recall, and F1-measure can be calculated for each individual class, but can also be averaged across classes to provide a general view of the performance of the classifier as a whole. Each metric can be reported as a macro-average or micro-average [17]. The former is an unweighted mean of the metric (calculated individually for each class, then divided across the number of classes), whereas the latter takes into account the predictions across all classes together (summing the counts of TP, FN and FP across all classes, and computing the precision, recall or F1 as a whole).

In the present case, the position taken is that precision and recall should be studied for each class in order to determine the model's ability to classify samples in specific classes. However, in order to report a metric which sums up the overall performance of a classifier, we report the macro-F1. As the dataset is imbalanced, the macro-F1 is relevant if we consider the ability of a model to identify samples in any of the 20 classes to be equally important.

It should be noted that if the view is that a model should be able to correctly classify as many samples as possible regardless of its performance on individual classes, then the focus should be on the micro-F1 score. However, in the specific case of a multi-class classification task, as is the present case, it should be clear from Table 2.2 that the sum of FN across all classes will be the same as that of the sum of FP. This effectively means that the averages of micro-precision, micro-recall and micro-F1 will all be equal to accuracy [47]. Thus, although we do not report the micro-F1 separately, this is equal to the average accuracy across all classes, which will be reported.

2.1.5.5 One-vs-rest and one-vs-one

There are several ways in which some classifiers can handle multi-class classification tasks.[1] A common method, which we will experiment with in logistic regression and SVMs, is one-vs-rest classification. This means one classifier is fitted per possible class. For k classes, k binary classifiers will be fitted. For example, when training the first classifier, all the training samples belonging to Class 1 will be positive samples, and all the other samples negative. The classifier is fitted on this basis. When used on the test set, the classifier outputs a confidence score for its decision, denoting whether each sample in the test set belongs to Class 1 or not. This process is repeated over all k classes, i.e. k classifiers are trained. On the test set, for a single sample, the confidence score for each of the k classifiers is compared. The classification representing the class with the highest score is taken as the classification for the purposes of the 'final' classifier.

In a one-vs-one scheme, each classifier is trained to distinguish between samples belonging to a subset of only two classes out of k classes in the dataset. This means that $\frac{k*(k-1)}{2}$ classifiers must be trained; in our case, where $k = 20$, this means 190 classifiers, compared to 20 in the one-vs-rest scheme. For the 'final' classifier, on the

[1] See Scikit-learn documentation at *https://scikit-learn.org/stable/modules/multiclass.html*

test set, each test sample is tested by all $\frac{k*(k-1)}{2}$ classifiers. The sample belongs to the class which the most number of classifiers has predicted [4].

2.1.5.6 McNemar's test

Where the performance of two models is similar, we can determine whether the difference between them is *statistically significant*, i.e., that the difference in their performance is not due to chance. Based on the argument in a seminal paper on statistical tests for comparing the performance of supervised learning models on classification tasks that it results in the lowest Type I error (an incorrect rejection of a true null hypothesis) [14], we use a variant of *McNemar's test* as amended by Edwards [16] to compare both classifiers based on statistical significance.[2]

	Classifier 2 (correct)	Classifier 2 (wrong)
Classifier 1 (correct)	A	B
Classifier 1 (wrong)	C	D

Table 2.3: Contingency table

The test uses a *contingency table* to compare the exact predictions made by two classifiers on the same test set. For the avoidance of doubt, a 'correct' result in the table is one which is a True Positive, i.e. the label predicted by the classifier matches the actual label of the sample. Any other output is 'wrong'. The table is a sample-by-sample comparison of the predictions of each classifier. Thus "A" is the count of the number of samples both classifiers predicted correctly, "B" is the count of the number of samples Classifier 1 predicted correctly but Classifier 2 predicted wrongly, and so on. From the counts in the table, we can calculate the modified McNemar's test statistic:

$$\chi^2 = \frac{(|B - C| - 1)^2}{B + C} \tag{2.8}$$

Our null hypothesis, H_0, is: "the performance of both classifiers equal". The test states that the *critical value* at a 95% confidence level is 3.84, thus if χ^2 >3.84, we can reject H_0. Similarly, we can also compute the p-value for significance testing:

$$p = 2\sum_{i=B}^{n} \binom{n}{i} (0.5)^i (1 - 0.5)^{n-i} \tag{2.9}$$

where $n = B + C$, and 2 indicates a two-tailed test. The critical value at a 95% confidence level is 0.05. Thus, if p <0.05, we can reject H_0.[3]

[2] See also Mlxtend documentation, *http://rasbt.github.io/mlxtend/user_guide/evaluate/mcnemar/*

[3] Formulae from mlxtend documentation, as above.

2.2 Text classification pipeline

As described by Kowsari et al. [38], text classification problems follow a fairly standard pipeline. This can be summarised in the following steps:

1. Preprocessing: the aim of this first stage is to remove noise from textual data to improve classification performance. This can involve various steps:

    - stopword removal: textual data often contains words which do not aid, and in fact can hinder the text classification task [38]. For example, articles such as 'the' and 'a' can appear multiple times in a given text, but do not provide a good indicator of the class to which a sample belongs. Similarly, punctuation and special characters, which may be important for human understanding, can be detrimental to the performance of classification algorithms [55]. It is common practice to remove these words and characters [38], and this is the approach taken.
    - lowercasing: converting all the words in a dataset to lowercase is desirable insofar as it makes intuitive sense, e.g. 'Apple' (the fruit) which appears as the first word of a sentence and is thus capitalised is not semantically different to 'apple' in the middle of a sentence, and it reduces the size of the vocabulary; however, it could lead to negative impact on performance, e.g. equating 'apple' (the fruit) with 'Apple' (the company) [7]. Given that it is common practice [7], and because the use of capitalisation in our dataset is inconsistent (as we will show), this is the approach taken.
    - lemmatisation and stemming: lemmatisation involves converting words to their basic forms (lemmas) by converting them or removing their inflectional endings e.g. 'is', 'are', 'being', all result in the lemma 'be'. Similarly, stemming involves removing the endings of words, but can lead to non-words, e.g. 'admiring' and 'admiration' are stemmed to 'admir', instead of the lemma 'admire'. Although these are traditionally regarded as standard preprocessing methods in text classification [71, 51], Camacho-Collados and Pilehvar [7] note that they are rarely used in neural-based systems, because they may result in the loss of important syntactic nuances. In addition, the main purpose of both lemmatisation and stemming is to reduce sparsity or dimensionality, i.e. to allow similar words to be represented as the same feature. However, this problem may be avoided through learning word embeddings in a defined vector space. For these reasons, the approach taken here is to not apply lemmatisation or stemming at the preprocessing stage.
    - tokenization: informally, each sentence in a document is made up of words. In linguistics, however, a word can be a type or a token (on the *type-token distinction*, see [75]). For present purposes it suffices to state that a token is a sequence of characters grouped together as a unit for processing [47]. This may or may not be a word, for example, the contraction 'aren't' may or may not be separated into two tokens, 'are' and 'n't', depending on

the tokenizer. In addition, an *n-gram* is a sequence of n tokens occurring together. The salient point to note here is that the same tokenizer will be used throughout this project for consistency.

2. Feature extraction: after preprocessing, the dataset will still consist of samples of textual data, which is unstructured data that cannot be passed into a classifier in its raw form. Thus, this step involves converting the textual data to a structured feature space. Methods of feature extraction are generally count-based (e.g. count vectorizer or TF-IDF vectorizer), or achieved through learning word embeddings or using pre-trained word embeddings.

3. Dimensionality reduction (optional): given that the dataset will consist of thousands of unique words, some methods, such as one-hot encoding which represents each unique word as its own feature, will result in feature representations of huge dimensions (in case of one-hot encoding, each unique feature results in an additional dimension) [20]. Such large dimensions can result in a high computational complexity, so dimensionality reduction techniques, such as principal component analysis (PCA) can be used if necessary [38]. Other methods, such as word embeddings, can map each feature into a defined dense vector space. In our case, dimensionality reduction is not used with the count and TF-IDF vectorizers. With word embeddings, each token is represented with a 300-dimension vector, and in order to preserve the meaning captured within this defined space (as explained in Section 2.3.3), we do not reduce this further.

4. Model selection: once textual data has been converted into a form which can be passed into a model, the next step is to define the architecture of the model. Different classifiers will be discussed, including the Naive Bayes classifier, logistic regression, support vector machine, multilayer perceptron and neural networks.

5. Evaluation: finally, the performance of each classifier must be assessed. A common simple metric used is accuracy, but, as Kowsari et al. [38] note, this is not appropriate for unbalanced datasets [31] (as is the case with the EURLEX dataset). Other methods which will be explored include the Area Under the Receiver Operating Characteristic Curve (AUROC), as well as confusion matrices, F1 score, precision and recall.

2.3 Feature extraction

2.3.1 Count vectorizer

This section is based on the implementation in the popular Scikit-learn package [56], which we use. The count vectorizer is a *bag of words* model, i.e. it disregards grammar or word order, but preserves counts of unique words. For our purposes, the dataset is first divided into training and test sets in an 80:20 split. The count vectorizer is then fitted on the samples in the training set. This means that each unique word in the entire vocabulary of the training set is given a unique integer ID (and thus the

number of IDs will be equal to the size of the vocabulary of the training set). Once the count vectorizer has been fitted, using the *transform* function with any text input returns an encoded output.

The following example should suffice for the purposes of clarity:

1. The training set consists of one document, with the text "the quick brown fox jumps over the lazy dog". An instance of the count vectorizer is fit over the document.

2. The vectorizer has thus learned the following encoding, with each unique word being assigned an integer ID, in alphabetical order:
{'brown': 0, 'dog': 1, 'fox': 2, 'jumps': 3, 'lazy': 4, 'over': 5, 'quick': 6, 'the': 7}.

3. A sample text is transformed using the fitted vectorizer. The text is "the quick dog jumps the sheep". This results in the encoded vector [0 1 0 1 0 0 1 2], with each vector index corresponding to the encoding given above. Thus, the integer at the first index in the vector, '0', corresponds to the number of times the first word in the vocabulary, 'brown', appears in the sample being transformed. The word 'sheep', which is not in the vocabulary which the vectorizer was fitted on, is not taken into account in the vector returned.

Where the vectorizer is trained on a large training set, it will have a large vocabulary. Each sample which is transformed will thus consist of a majority of zeroes representing each of the words which appear in the training set vocabulary but not in the sample. In the Scikit-learn library, each sample transformed by the count vectorizer is thus returned in the form of a sparse matrix for computational efficiency, with counts corresponding to each non-zero word index.

The count vectorizer is thus an efficient vectorizer based on the bag of words model, and the representation it returns takes into account the frequency with which words appear in a dataset. However, it has various limitations. It does not take the context or the order in which words appear into account. It also does not take into account n-grams, considering only the counts of each unique word appearing in isolation. Finally, it also does not take into account similarities between words, for example, intuitively, the words 'man' and 'woman' may be semantically more similar to each other than the words 'man' and 'table', but unlike word embeddings, the count vectorizer cannot account for such similarities.

2.3.2 TF-IDF vectorizer

One problem with the count vectorizer is that even after the removal of common stopwords, certain tokens in a dataset may appear many times, but may not be useful in determining which class a sample should be classified to. For example, in the EURLEX dataset, the tokens "shall" and "article" occur multiple times, as shown in Table 3.5, but these tokens do not provide any clues as to the class a sample belongs to.

One way to alleviate this is through the use of a vectorizer which takes into account 'term frequency-inverse document frequency' (TF-IDF). This was traditionally used in information retrieval tasks, but has been shown to improve performance in text classification tasks [35]. Term frequency refers to the number of times each term appears in a document (which the count vectorizer takes into account). Inverse document frequency takes into account the frequency with which words appear across the documents in a dataset, and scales down the importance of the words which appear across many documents in the dataset, based on the assumption that such words are less useful for classification than those which appear in fewer samples.

Concretely, given a term t and a document d, where n is the number of documents in the dataset, and *DF(t)* is the number of documents in the dataset in which t appears:

$$IDF(t) = log(\frac{n}{DF(t)}) + 1$$

taking the logarithm, a sub-linear function, thus reflects the fact that the relevance of a term to a document's classification does not increase linearly with an increase in the term's frequency [62]. 1 is added to the *IDF(t)* formula to prevent terms which appear in every document from being ignored (given that *log(1)=0*). Thus,

$$TFIDF(t, d) = TF(t, d) \cdot IDF(t)$$

In the context of the Scikit-learn library [56], a sample transformed by the TF-IDF vectorizer is similarly returned as a sparse matrix, with a TF-IDF score instead of a TF count. Overall, while the TF-IDF vectorizer takes into account the assumption that words which appear in many documents in a dataset may be less meaningful for a classification task, it faces many of the same limitations faced by the count vectorizer noted above.

2.3.3 Word embeddings

Similar to the count and TF-IDF vectorizer, a *word embedding* is a means of representing a word as a vector. However, one key difference is that each word or sentence is *embedded* into a fixed dimensional space, and represented as a vector within that space [20]. This is a key benefit, as a one-hot or bag of words model such as the count vectorizer results in sparse vectors with dimensions equal to the size of the vocabulary, which can be computationally inefficient to work with.

Another key advantage is that while a bag of words model assigns each word in the vocabulary some unique integer ID without taking into account similarities between the meaning of words, word embeddings allow words with similar meaning to be represented closely in the defined vector space. A famous example was achieved using the word2vec algorithm [49]:

$$vector(\text{“}King\text{”}) - vector(\text{“}man\text{”}) + vector(\text{“}woman\text{”}) = vector(\text{“}Queen\text{”})$$

It is possible to use *pre-trained* word embeddings in a model. These are word embeddings which have already been prepared based on a large training set, and made available in a library, such as word2vec [49] or GloVe [57]. The vector representations of word in a dataset can thus be directly obtained from these pre-trained word embeddings. However, this also means that any word in the dataset which is not within the vocabulary of the pre-trained word embeddings will not have a vector representation. Alternatively, word embeddings can be learned as part of the training of a neural network. An embedding layer will be added to the model, and the vector representation for each word in the vocabulary will be learned along with the training of the model.

For present purposes, pre-trained word embeddings are used from the spaCy library,[4] which maps each word into a 300-dimension vector. Sentence embeddings are also experimented with, where the vector representation of a sentence is simply the average of the word embeddings of all the words in a sentence. For our experiments, word embeddings are used with only the neural network models, to determine if they can be used to improve performance over the baseline models using bag of words-based vectors. This is because the machine learning classifiers do not work with the dimensions of the word embeddings, and would require them to be reshaped. However, reshaping word embeddings would result in the representations of words in the defined vector space being lost. An alternative approach would be to use a document representation such as doc2vec,[5] however, this was not attempted due to time constraints.

2.4 Classifiers

2.4.1 Naive Bayes classifier

The Naive Bayes classifier is commonly used in text classification applications such as email client spam filtering, and is equally applicable to multi-class classification problems. The comprehensive overview provided by Raschka [60] is summarised here. This is the baseline model used by Mencia and Furnkranz [48], who compiled and experimented with the EURLEX dataset.

The Naive Bayes classifier is a linear classifier based on Bayes' theorem, which is:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{2.10}$$

given two events, A and B, and where $P(B) \neq 0$. Alternatively, we can define Bayes' theorem in words:

$$posterior\ probability = \frac{conditional\ probability \cdot prior\ probability}{evidence} \tag{2.11}$$

Extended to the text classification context, Bayes' theorem can be formulated as such:

[4] See *https://spacy.io*
[5] See *https://radimrehurek.com/gensim/models/doc2vec.html*

$$P(c_j|x_i) = \frac{P(x_i|c_j) \cdot P(c_j)}{P(x_i)} \tag{2.12}$$

where x_i is the vector obtained after feature extraction from text sample i, $i \in \{1, 2, ..., n\}$, given n is the total number of samples; and where c_j denotes class j, $j \in C$, and $C = \{1, 2, ...20\}$ in our case.

However, Bayes' theorem only holds true given the *naive* assumption, or the *conditional independence* assumption, that the occurrence of every feature (i.e. in our case, every word) in a sample is independent of all other features.

This assumption is patently violated in most contexts, hence its characterisation as a 'naive' assumption. It is also violated in most textual datasets, and in particular in the EURLEX dataset, e.g. the bi-gram 'European Union' occurs repeatedly, thus the frequency of occurrence of the feature representing the word 'European' is not independent of that of 'Union'. However, we still choose to examine the performance the classifier as it has been shown to perform well in conditions where the assumption is violated [61], and even outperforms more complex models where sample sizes are small [15].

Broadly, this formula can be used for our multi-class text classification task: given a vector x_i, the formula can be used to find the probability of the vector belonging to the class c_j. Repeated over all classes of j, $j \in \{1, 2, ...20\}$, the class c_j which the vector x_i belongs to is the class with the highest probability. Thus, we can compute the *posterior probability*:

$$P(c_j|x_i) = \underset{c_j \in C}{\operatorname{argmax}} \frac{P(x_i|c_j) \cdot P(c_j)}{P(x_i)} \tag{2.13}$$

This can be further simplified; given that we are largely concerned with the class c_j to which a vector x_i belongs, the denominator $P(x_i)$ will remain the same for all values of j, thus merely serving as a scaling factor, and can be ignored for the purposes of defining our classifier. Thus, we have our simplified *decision rule*:

$$P(c_j|x_i) = \underset{c_j \in C}{\operatorname{argmax}}(P(x_i|c_j) \cdot P(c_j)) \tag{2.14}$$

Given the above, we can compute the *general probability of encountering a particular class*, or the *prior probability*, simply by finding the number of samples belonging to the class j as a proportion of the total number of samples, n:

$$P(c_j) = \frac{n_{c_j}}{n} \tag{2.15}$$

Next, the *conditional probability*, $P(x_i|c_j)$, must be found. We can compute the probability that we will observe a certain vector x_i given that we know that it will belong to the class c_j, by multiplying the probability of each feature of that vector occurring in that vector (recalling that this holds because of the *conditional independence* assumption). Thus, for a vector x_i of dimension d:

$$P(x_i|c_j) = P((x_i)_1|c_j) \cdot P((x_i)_2|c_j) \cdot ... \cdot P((x_i)_k|c_j) = \prod_{k=1}^{d} P((x_i)_k|c_j) \tag{2.16}$$

Calculating the individual probabilities of each feature in our case entails calculating the probabilities of each word occurring, by dividing the number of times the word k appears in the class c_j by the total number of words in the class c_j:

$$P((x_i)_k|c_j) = \frac{n_{(x_i)_k,c_j}}{n_{c_j}} \tag{2.17}$$

where $n_{(x_i)_k,c_j}$ is the number of times the k-th word of vector x_i appears in class c_j, and n_{c_j} is the total number of words in the class c_j. Multiplying all the probabilities for every word in a vector x_i thus returns the *conditional probability* $P(x_i|c_j)$, which is the probability that we will observe a certain vector x_i, given that we know that it will belong to the class c_j.

Thus, given all of the above, we can apply the *decision rule* above to unseen samples, to find the class that the sample is most likely to belong to based on the application of this classifier. Where there are words in the test samples which are not seen in the vocabulary of the training samples, to avoid having zero probabilities, *smoothing* can be used. Denoting the value added to both the numerator and denominator of the posterior probability as α, this is called *Laplace smoothing* where $\alpha = 1$, and *Lidstone smoothing* where $\alpha < 1$. Thus, Equation 2.17 becomes:

$$P((x_i)_k|c_j) = \frac{n_{(x_i)_k,c_j} + \alpha}{n_{c_j} + \alpha} \tag{2.18}$$

Overall, the Naive Bayes classifier is regarded as a linear classifier [36], and performs better the more the *conditional independence* assumption holds true, and the more the data is linearly separable.

2.4.2 Decision tree

The decision tree classifier is a non-linear classifier which learns a set of decision rules from training data. Starting with all the training samples at the root of the tree, the classifier splits the data into smaller subsets based on a decision rule. The aim of the decision rule is usually to split the data according to the division which will minimise or maximise some function, such as *Gini impurity* or *information gain* [64], both of which will be experimented with. Each splitting point is termed a *node*, and branches emerge from each node with the possible outcomes based on the decision rule at that node. The process continues until all the data is classified, and at this point, all the nodes are called *leaves*.

Gini impurity measures the frequency with which a randomly selected sample from the set of samples remaining at a node would be labelled incorrectly, if it was labelled randomly following the distribution of labels in the subset returned by the decision rule [23]. The decision rule which results in the greatest reduction to gini impurity

is chosen as the splitting rule. Information gain, on the other hand, is based on another measure, *entropy*, which effectively measures to what extent the samples in a set belong to the same class. At each node, the decision rule which then results in the greatest homogeneity in terms of class distribution of samples in the subset is then taken to be the most useful decision rule for that node.

Decision trees are prone to the problem of overfitting [64], and to prevent this, we will aim to tune the hyperparameters of the classifier by setting the maximum depth of the tree and also the minimum number of samples in each leaf node.

One key advantage of decision trees is that unlike 'black box' neural models, decision trees can be visualised and the exact rules on which splits are determined can be analysed. Thus, a visualisation of a decision tree will be provided to analyse the decision rules of the tree, even if the classifier does not perform as well as other models.

2.4.3 Random forest

The random forest is an *ensemble* machine learning method, i.e. it makes predictions based on a combination of several classifiers which are individually trained, but whose predictions are combined when classifying unseen samples. The random forest is based on multiple decision trees. The process of *bootstrap aggregating* can be applied, in which subsets of the training set are sampled with replacement, and individual decision trees are fit to these subsets. The intuition is that this should reduce the likelihood of overfitting. In addition, in each individual decision tree, the *random subspace* method is used: only a subset of all the available features are taken into account in determining the splits at each node, to reduce the correlation between trees. The predictions of these individual trees are then averaged [40].

As the intuition is that the ensemble should be more accurate than any of the single classifiers which make up the ensemble model, the expectation is that the random forest will result in a better performance than a single decision tree [40]. As with decision trees, the hyperparameters of a random forest classifier can be tuned, by, among other things, setting the maximum depths of trees or the maximum number of features to take into account at each node.

2.4.4 Logistic regression

The next classifier considered is multinomial logistic regression, a log-linear classifier [36]. A log-linear classifier is considered a generalised linear classifier, because its outcome depends on the sum of its inputs and parameters [59]. Again, the aim is to learn, from the training set, the mapping function f which maps an unseen input X, which is a vector of features, to a class C_i. The classifier achieves this by first learning from the training set a vector of *weights*, w and a bias, b. This can be represented by a variable, z, thus:

$$z = w \cdot X + b \tag{2.19}$$

where each number in the vector w corresponds to a feature in the input sample X, and represents how important that feature is to the classification decision. The bias b shifts the function to ensure that it does not necessarily have to intersect the y-axis at the origin, as the data may not necessarily be best separable by a function that passes through the origin.

However, our text classification task requires a probability (denoting the likelihood of a sample belonging to a certain class) as an output. Thus, it is passed through the *softmax* function:

$$softmax(z) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{2.20}$$

where the numerator e^{z_i} is the exponential function applied to each element of the input vector z, and the denominator normalises all the values to probabilities. Thus, the function results in a vector output which represents a probability distribution over j classes, where $j = 20$ in our case, where all the elements of the vector sum to 1, and the value of j with the highest probability in the vector is taken to be the output class for the purposes of the classifier. We denote this predicted output class $\hat{C}_i$.

In order to learn the values for w and b in Equation 2.19, a *loss function* is measured, representing the distance between the expected output C_i and the predicted output $\hat{C}_i$. The aim is to find the weights and biases which minimise the *categorical cross entropy* loss function (see Equation 2.23) through the process of *gradient descent* over a number of iterations.

2.4.5 Support vector machines

With the vectors obtained from feature extraction from our text samples, we can obtain the mapping of the feature vectors in our feature space. The support vector machine is a linear classifier which aims to find the optimal hyperplane which can separate the feature vectors in our multi-dimensional feature space. However, for a given set of data, there can be many possible hyperplanes that can separate the data, so the key question is how to define the *optimal* hyperplane. In other words, we need a method to find the hyperplane that best classifies the training data, and generalises best on the unseen test data.

For each separating hyperplane, it is possible to calculate the distance between that hyperplane, and the data points (i.e. the *support vectors*) closest to the hyperplane. The number of support vectors can be chosen. The distance between the support vectors and a hyperplane is the *margin*. The SVM classifier is thus the classifier that is based on the hyperplane that results in the maximum distance between the hyperplane and the support vectors, i.e. the *maximum-margin hyperplane*. In a seminal article, Joachims [35] demonstrated that the linear SVM performs well on text classification tasks.

However, as noted above, the SVM is a linear classifier. It is possible to also adapt the SVM to work as a non-linear classifier, by using a *kernel function*, as proposed by Aizerman et al. [1]. Crudely summarised, the function is used to project each data point in a dataset which is not linearly separable to a higher-dimension feature space, in which it then becomes possible to fit a maximum-margin hyperplane, i.e. in which the data becomes linearly separable. Some common kernel functions used are the *radial basis function* (RBF), the *sigmoid* function, and the *polynomial* kernel function. In the Scikit-learn implementation which we will use, the functions also take a coefficient which affects how the 'shape' of the data is captured.[6]

We recognise at the outset that, as Hsu et al. [30] note, where there are many features in the dataset, as in our present case, the feature space will already have many dimensions, and mapping the data to a higher dimensional space will not improve performance.

In our case, the performance of both linear SVM and non-linear SVM with a kernel function chosen by random search are reported, in order to illustrate the difference between their performance on the EURLEX dataset.

2.4.6 k-Nearest Neighbours

The performance of the k-NN classifier is also reported as it is a popular algorithm for text classification [72]. Specifically, Trstenjak et al. [72] report results using k-NN with TF-IDF feature extraction, however, in our case, we report performance on both the count vectorizer and the TF-IDF vectorizer, with classifications based on the 5-nearest and 10-nearest neighbours.

Similar to the case of a SVM, we refer to the mapping of the feature vectors in our feature space. Each feature vector can also be termed a *point*, and the feature space can also be termed the *Euclidean space*.

Each unseen sample can then be plotted as a point in the Euclidean space. It is then possible to calculate the *Euclidean distance* between the unseen sample point and its *k-nearest* points, where k is some positive integer representing the number of neighbouring points we want to take into consideration for our classification, e.g. 5.

Given two points in a m-dimensional space, $A = (x_1, x_2, ..., x_m)$ and $B = (y_1, y_2, ...y_m)$, the Euclidean distance between them can be calculated as such:

$$dist(A, B) = \sqrt{\frac{\sum_{i=1}^{m}(x_i - y_i)^2}{m}} \tag{2.21}$$

In k-NN classifiers, it is common to use the Euclidean distance to measure the distance between points in the Euclidean space, but other measures are possible. We experiment with both uniform distance, where the distance of all neighbouring points to a

[6] For detailed explanation, see *https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html*

sample are weighted equally, and inverse distance, where points are weighted by the inverse of their distance to the sample, thus the closer neighbouring points will have a greater impact on classification than those points further away.

In addition, it should be noted that the k-NN classifier is a *lazy learner*, as opposed to an *eager learner* such as the Naive Bayes classifier [60]. Thus, as Trstenjak et al. [72] note, it is not necessary to split a dataset into training and test sets for the purposes of the k-NN classifier. However, for consistency with the other classifiers, the dataset is split in a 80:20 proportion, and the accuracy of the k-NN classifier is measured on the test set.

The performance of the k-NN classifier is specifically included also as it is a non-linear classifier, and allows a general comparison between the success of linear and non-linear classifiers on the EURLEX dataset. Further, Trstenjak et al. [72] run their k-NN model on different sources of text data, from 'sport', 'politics', 'finance', and 'daily news', and show that the performance of the classifier can show a fairly large variation, from 65% to 92% accuracy, depending on the type of textual document. Thus, given that the EURLEX dataset is specifically a dataset of legal texts, the performance of the k-NN classifier specifically on texts from legal sources is also of interest.

2.4.7 Multilayer perceptron

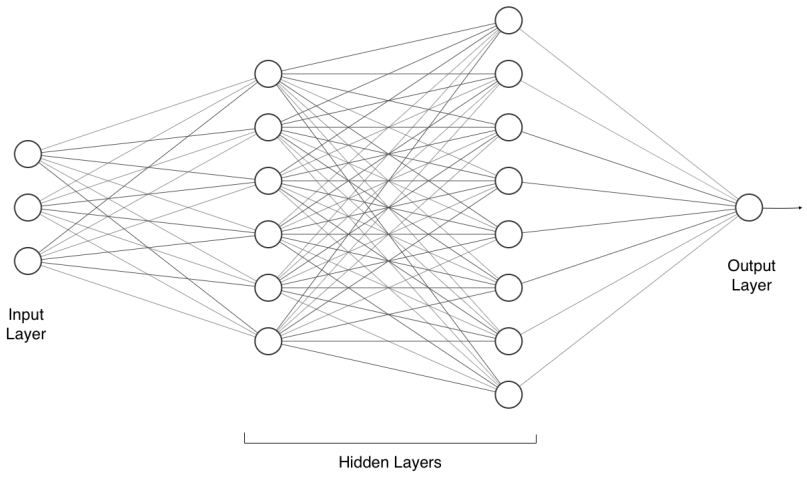


Figure 2.1: Multilayer perceptron, showing fully connected feedforward hidden layers

The MLP is a feedforward neural network, consisting of an input layer, one or more hidden layers, and an output layer, as shown in Figure 2.1. In general, each neuron in a layer is fully connected to the neurons in the next layer. Similar to other models, the model is trained on a labelled training set. The aim is, as per usual, to learn the mapping function f which maps takes an unseen sample vector X to a predefined class C_i. As with logistic regression, the classifier learns a set of weights and biases in the

hidden layers, updating them over several iterations with a gradient descent approach specifically termed *backpropagation*, by minimising the loss function.

The connections between the layers of a MLP do not form a cycle, unlike in some other neural network architectures. The input to neurons are also transformed by activation functions. Using linear activation functions across hidden layers would mean that all the layers in the network can simply be reduced to a single linear input and output mapping, making the network equivalent to a linear perceptron. To allow the network to learn non-linear relationships in the data, in the present case, for each hidden layer, the activation function used is the *rectified linear unit* (reLU). The reLU function is partly linear and thus called a *piecewise linear* function, however, the overall effect is that it makes the network a non-linear one. Its use as an activation function in neural networks is popularised by Glorot et al. [19], and is given by:

$$reLU(x) = x \; if \; x > 0, \; else \; 0 \tag{2.22}$$

Its popularity partly stems from the fact that it has been shown to avoid the *vanishing gradient problem* which the sigmoid activation function may face. Given that the task is a multi-class classification task, the output layer is a softmax layer, as per Equation 2.20, and *categorical cross entropy* is used as the loss function, a generalisation of the cross entropy function:

$$L(X^{(i)}, C^{(i)}) = -\sum_{k}^{n} X_k^{(i)} log(C_k^{(i)}) \tag{2.23}$$

where $C^{(i)}$ is the classification given by the model, and $X^{(i)}$ is the input to the model, and k represents the number of classes (i.e. 20 in our case).

Thus, as the MLP is a basic feedforward neural network, its performance is reported to facilitate comparisons with other more complex deep learning models.

2.4.8 Convolutional neural networks

CNNs, designed by LeCun et al. [42], have traditionally been used for computer vision, but have also shown strong results on text classification and other NLP tasks [10, 33]. The overview of CNNs provided by Goodfellow et al. [22] is crudely summarised and adapted to our context here. CNNs differ from feedforward neural networks in several ways.

First, it should be noted that *convolution* refers to the mathematical operation, denoted by an asterisk. Thus, given an input, X, and a weighting function or *kernel*, W, at time t, we can apply the convolution function, to obtain an output, s:

$$w(t) = (X * W)(t) \tag{2.24}$$

A layer in a neural network which uses the convolution function can thus be described as a convolutional layer. In a convolutional layer, neurons are also often connected

only to a restricted area, the *receptive field*, of the previous layer, because the kernel will have *sparse weights*, i.e. the kernel will be smaller than the input. The result of applying a convolutional layer is that given an input, X, we can obtain a *feature map*, the output w. The practical effect of this is that certain specific features of the input X are extracted. What exactly is extracted depends on the kernel used, W.

In addition, while in a feedforward network each element of the weights in a layer are multiplied element-wise to its input to obtain its output, the weights in a convolutional layer are shared or tied, i.e. the weights W are the same across the neurons in a convolutional layer. As with feedforward networks, a non-linear activation function is applied to each layer as well. Again, in our case, the function used will be reLU (Equation 2.22).

Finally, the output from the activation function is also modified further, with a *pooling function*. The effect of pooling is to downsample, or summarise, certain features within a region in a feature map. This downsampling can be done through taking the average of the features within a region in a feature map, in which case it is called *average pooling*, or through taking only the maximum of the features in a region, i.e. *max pooling*. Further, instead of focusing on a specific region, the pooling operation can be done on an entire feature map, e.g. *global max pooling*. The purpose of pooling is to make the network *invariant* to small changes in the input, so if the values of the input change by some small amount, the output of the pooling layer will remain the same.

In practice, perhaps the most common use of CNNs is with image data. Feature maps are used to extract specific objects or edges (i.e. the outlines of an object) in an image. Pooling ensures that the network is not sensitive to the particular location of an object in an image; for example, if the network has learned the shape of a dog, it can then detect the presence of a dog in an image, regardless of where in the image the dog is located, or if a dog in one image looks slightly different to another.

The use of CNNs was extended to the NLP context by Collobert and Weston [10]. In the context of text classification, feature maps are used to focus on specific tokens which are important to the classification task. In addition, pooling is used with the same intuition as in images: tokens which provide clues as to the actual label of a sample remain informative, regardless of where they appear in the sample.

This was taken further by Kim [37], based on the idea of n-grams. In NLP, an n-gram is simply a contiguous sequence of n tokens appearing in a text: a 1-gram or *uni-gram* is a single token, a 2-gram or *bi-gram* could be e.g. 'European Union', and so on. The idea is that applying layers with varying n-sized filters will be able to capture the most important n-grams for the purposes of the classification task. In our case, we will experiment with a basic CNN as well as a CNN based on Kim's architecture, with tri-gram, 4-gram and 5-gram filters.

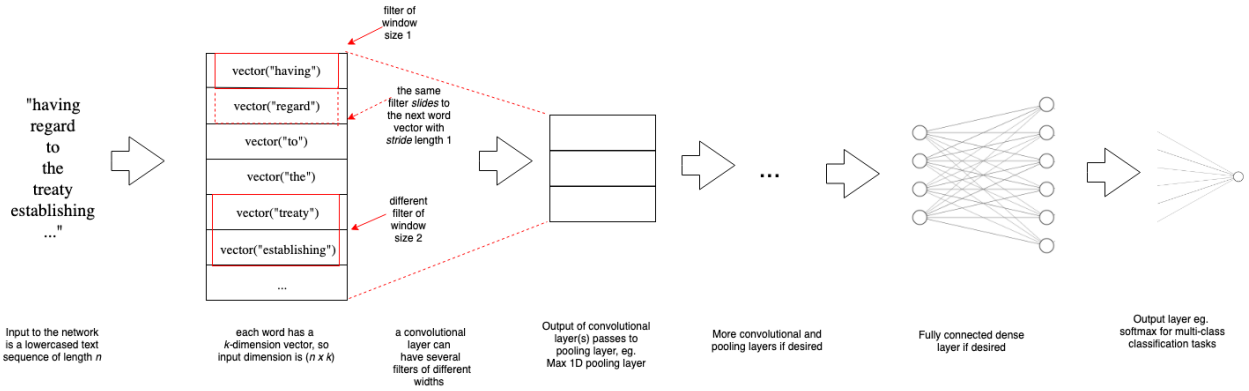


Figure 2.2: Diagram of a CNN with 1D convolutional layer, showing filters and strides

Concretely, we can summarise the vector transformations in a CNN. The following explanation is based on Goldberg [20], adapted to our context. We have attempted to represent this diagramatically in Figure 2.2. As usual, our input X is a sequence of n tokens, thus $X = \{x_1, x_2, ...x_n\}$. Each token $x_i \in X$ has a word embedding, a vector $v(x_i)$. Next, we have a 1D convolutional layer of size k, which has some number of filters. We move each filter across the sample; how much we move it by is called its *stride*; thus a filter of size $k = 2$ and stride length 1 will capture $(v(x_1), v(x_2))$, then $(v(x_2), v(x_3))$ and so on, up to $v(x_n)$. The effect of the filter, applying convolutions (Equation 2.24), is to transform the k tokens into a d-dimensional vector, w_i, which has focused on the important properties of the tokens in that window. To each output w_i we apply our network's weights W and biases b (which are updated as usual with backpropagation), and the non-linear activation function. We denote the output of this transformation p_i. Hence:

$$p_i = reLU(w_i W + b) \tag{2.25}$$

where, if we slide the window over the sample m times, we get m vectors $(p_1, p_2, ...p_m)$. We then apply a max pooling layer to each of the m vectors, obtaining m vectors $(c_1, c_2, ...c_m)$ of a reduced dimension, which are then concatenated to form a single vector c:

$$c = \max_{1 \leq i \leq m} p_i[j] \tag{2.26}$$

where $p_i[j]$ is a component j in the vector p_i, i.e. a specific region of the window. Alternatively, after a convolutional layer, we can apply a global max pooling layer instead, which samples the entire vector p_i instead of some region j, hence:

$$c = \max_{1 \leq i \leq m} p_i \tag{2.27}$$

In our case, we use both max pooling and global max pooling layers after convolutional layers. Lastly, again, the final layer for our task is a softmax layer, and the loss function is the categorical cross entropy loss function.

2.4.9 Recurrent neural networks

RNNs are generally used to process data in the form of a sequential input $(x_1, x_2, ...x_n)$. Of course, textual data is sequential in nature: words are formed from a series of alphabets, sentences from words, and documents from sentences; thus, RNNs are a natural candidate for NLP tasks. The following summary of RNNs is based on Olah [54].

The intuition behind the RNN is that a traditional feedforward network does not have *persistent memory*. This problem is relevant to us: it means that a network cannot use what it has learned in a certain section of a sample, e.g. in the introduction of a sample of legal text, where the subject matter may be introduced in context, to inform its learning about a later part of the sample. RNNs aim to alleviate this problem. Specifically, where neurons in a feedforward network only have connections to the next layer, i.e. they only 'feed forward', neurons in RNNs have loops.

However, in a basic RNN architecture, if the point at which relevant information appears is very far away in a document to the point at which the information is needed, i.e. the network needs to learn a *long-term dependency* [26], the RNN may not be able to connect the information. Specifically, Bengio et al. [2] explain that there are problems when training such a network with gradient descent, such as the vanishing gradient problem. Again, this problem is specifically relevant to us, given the length of legal documents as compared to e.g. an IMDB or Twitter dataset. However, a specific kind of RNN, a *long short-term memory* network (LSTM), first designed by Hochreiter and Schmidhuber [25], can be used to avoid this problem.

Broadly, LSTMs are made up of *units*, each of which has a cell and has gates. A cell has a *state* which can be regulated by a gate. There are several types of gates, such as a *forget gate* which uses a sigmoid function (resulting in an output between 0 and 1) to describe how much input is let through the gate (where 0 means 'forget everything'). Once some portion of the information has been let through, it passes through another sigmoid layer, the *input gate*, which decides which weights are updated. Finally, the output can also be passed through a tanh layer, the *output gate*, which serves as another form of filter.

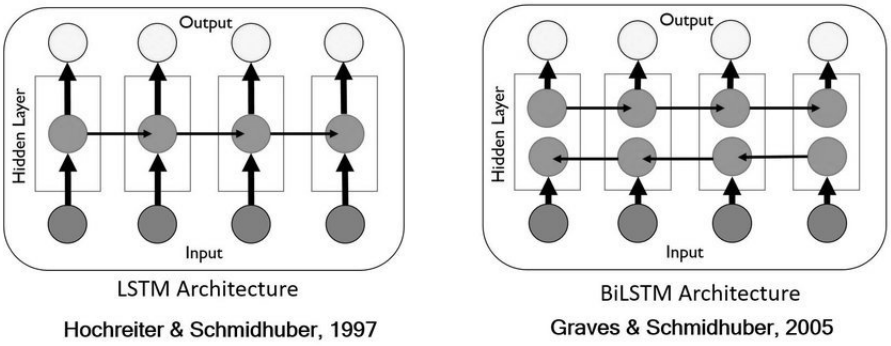


Figure 2.3: Diagram showing difference between LSTM and bi-LSTM (Figure 3, [50])

Thus, in effect, the LSTM cell's state preserves useful information from input that has passed through it. But one possible shortcoming of this is that it only preserves information from the past with reference to a specific point in time, i.e. for a given sample X of n tokens, $X = (x_1, x_2, ...x_n)$, the inputs are passed into the network in the order x_1, followed by x_2, and so on. As a result, e.g. at time $t = 2$ the cell state only has information about x_1, and so on. To allow the cell state to also preserve information from the future at a certain point in time, another LSTM layer is added to the network, but running in the opposite direction, as shown in Figure 2.3. The input X is passed into this additional LSTM in the opposite order, i.e. $(x_n, x_{n-1}, ...x_1)$. This allows the network to obtain information from the past and future simultaneously, thus at time $t = 2$, the network will have information about both x_1 and x_n. The intuition is that this will allow the network to better determine which information to retain in memory. For our purposes, we experiment with both a uni-directional LSTM, as well as a bi-directional LSTM to determine if the latter will improve performance.

2.4.10 Hierarchical attention network

Finally, the last classifier we will consider is the HAN, developed by Yang et al. [76]. This section is informed by Krankel and Lee [39]. The HAN is based on two broad ideas: first, that documents have an underlying *hierarchical* structure; words form sentences, and sentences put together form a document. Thus, we should not only work with word embeddings or sentence embeddings, but find a means to capture this hierarchical structure. Second, each word and sentence in a document is useful to a network in performing its task to a different extent. Thus, the network should pay more *attention* to some words and sentences than others.

As we are working with word embeddings and sentence embeddings individually, we decided to implement a HAN to determine if it would in fact improve on the performance of using word or sentence embeddings on their own. In addition, Yang et al. [76] develop their HAN for a document classification task, making it particularly relevant for our present case.

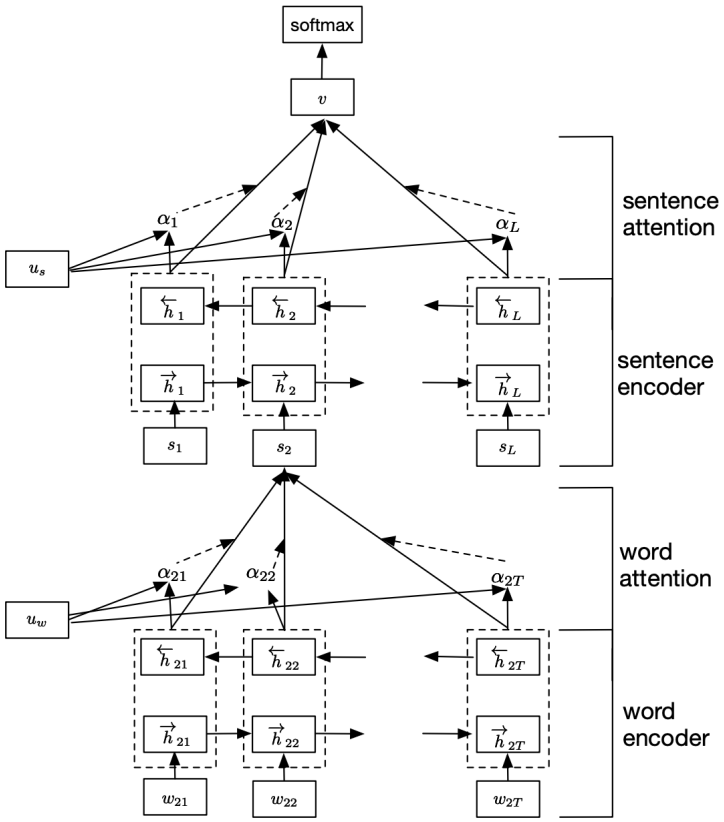


Figure 2.4: Structure of a HAN, with 4 hierarchical layers (Figure 2, [76])

The HAN is broadly comprised of 4 parts (excluding the standard softmax output layer), in the following order: the word encoder, word attention layer, sentence encoder, sentence attention layer. Its structure is shown above, in Figure 2.4. The word and sentence encoders are bi-directional *gated recurrent units*, which are a variant of LSTMs with only two gates, an *reset gate* (which effectively performs the functions of an input gate and a forget gate) and an *update gate*. Therefore, crudely put, the encoders are simplified bi-LSTMs. Their aim is to summarise the contextual information of the words or sentences passed into the network as input, and return a vector representation of what it has learned from each word or sentence, by concatenating the states of the forward GRU and the backward GRU.

This vector output, which is a summary of the input sample, is then passed to an attention layer. This consists of two parts: a multilayer perceptron with one hidden layer using a tanh activation and randomly initialised weights and biases; and a softmax function which serves to normalise the vector. The point of this layer is to capture the most important information to the task at hand from the vector.

Having discussed the constituent parts of the HAN, we can summarise its pipeline. A raw text sample, X, a document consisting of several sentences, is passed into the network. The input is split, sentence-wise, by a sentencizer. Each sentence is then split into individual tokens by the word encoder, which vectorises each token, and passes it

to the word attention layer, which modifies the vector by passing it through a one-layer MLP. The word vectors of all the words forming a single sentence are concatenated to form a sentence vector, and passed in to the sentence encoder. The same process is repeated at the sentence level, and the output of multiple modified sentence vectors is concatenated to form a document vector. We can then pass each document vector to the usual output layer, a softmax layer, to obtain a classification.

It should in particular be noted that for the previous models discussed, we pass in pre-trained word or sentence embeddings into the model in each case. However, in the case of the HAN, the raw text is passed in as input. The point of the encoder is to capture the most important contextual information from a given input, thus it returns a vector representation for each word or sentence passed in, i.e. a word or sentence embedding.

Chapter 3

The EURLEX Dataset

3.1 Structure of the EUR-Lex database

The dataset used for the purposes of this project was compiled by Mencia and Furnkranz [48] (the 'EURLEX dataset')[7], and was compiled from the publicly available EUR-Lex portal[8]. European Union law in general can be divided into, among other things, primary and secondary legislation. The former refers to the documents which relate to the formation of the European Union itself, such as the EU treaties, while the latter consists of the documents which affect the citizens of the EU more directly. The EUR-Lex portal consists of both primary and secondary legislation. These are divided into three sections: 'EU law', 'EU case law', and 'National law and case law'. Within 'EU law', there are 8 sub-classes, as shown in Figure 3.1. The EURLEX dataset consists specifically of texts from 'Legal acts', a form of secondary legislation.

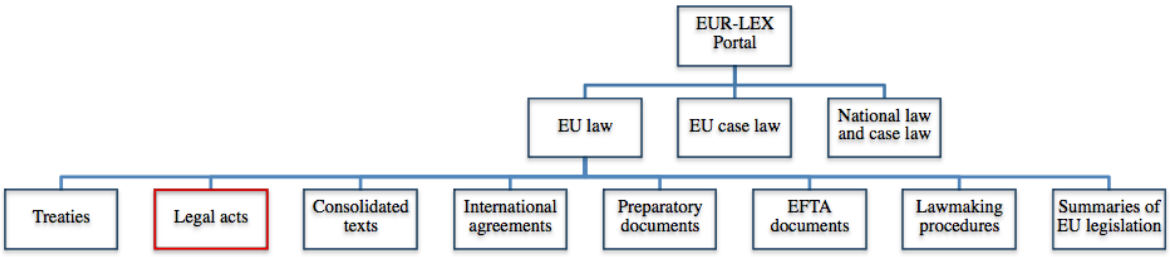


Figure 3.1: Structure of the EUR-Lex portal, with 'legal acts' boxed in red

3.2 The EUR-Lex paper

Although the task for this project has been modified to be slightly different to that of Mencia and Furnkranz [48], the approach taken in their paper will be briefly discussed. They first retrieved the HTML documents by scraping the EUR-Lex portal for all the

[7] The full repository is available at http://www.ke.tu-darmstadt.de/resources/eurlex
[8] See https://eur-lex.europa.eu/homepage.html
Data reused with permission. ('© European Union, https://eur-lex.europa.eu, 1998-2019')

legal acts available at the time of publication of their paper (2010).[9]

Each of these documents have been labelled by the European Union in three different ways: EuroVoc descriptor[10], subject matter and directory code (as in Figure 3.2). Each document has at least one label from each of the three categories, but can have multiple labels. This resulted in a total of 4,558 possible classes. Thus, Mencia and Furnkranz [48] termed their task an '*extreme multi-label multi-class text classification*' task, given the objective of classifying documents according to 4,558 possible classes.

Title and reference

Council Directive 91/250/EEC of 14 May 1991 on the legal protection of computer programs

Classifications

EUROVOC descriptor

- *data-processing law, computer piracy, copyright, software, approximation of laws*

Directory code

- 17.20.00.00 *Law relating to undertakings /* Intellectual property law

Subject matter

- *Internal market, Industrial and commercial property*

Text

COUNCIL DIRECTIVE of 14 May 1991 on the legal protection of computer programs (91/250/EEC)

THE COUNCIL OF THE EUROPEAN COMMUNITIES,

Having regard to the Treaty establishing the European Economic Community and in particular Article 100a thereof,

Having regard to the proposal from the Commission (1),

In cooperation with the European Parliament (2),

...

Figure 3.2: Example of raw HTML document [48]

For ease of reference, an example of a HTML document provided by the authors is reproduced here. From the HTML documents, the authors extracted the 'Text' section of the documents. Each HTML document has its own CELEX ID, the EUR-Lex portal's internal ID system. However, the authors numbered the documents according to the alphabetic ordering of the CELEX IDs. They then removed from the dataset those documents which were not in English, contained error messages, or were empty, resulting in 19,348 samples. They provide a file containing the mappings of each doc-

[9] Available at *http://www.ke.tu-darmstadt.de/files/resources/eurlex/eurlex_download_EN_NOT.sh.gz*

[10] For the full EuroVoc thesaurus, see *https://eur-lex.europa.eu/browse/eurovoc.html*

ument to the different classifications extracted from the HTML documents.[11]

Once this was done, they could then proceed with the standard text classification pipeline described in Section 2.2, i.e. preprocessing. They transformed each text to lower case, removed stop words from a common English stop word list,[12] and then applied the Porter stemming algorithm.

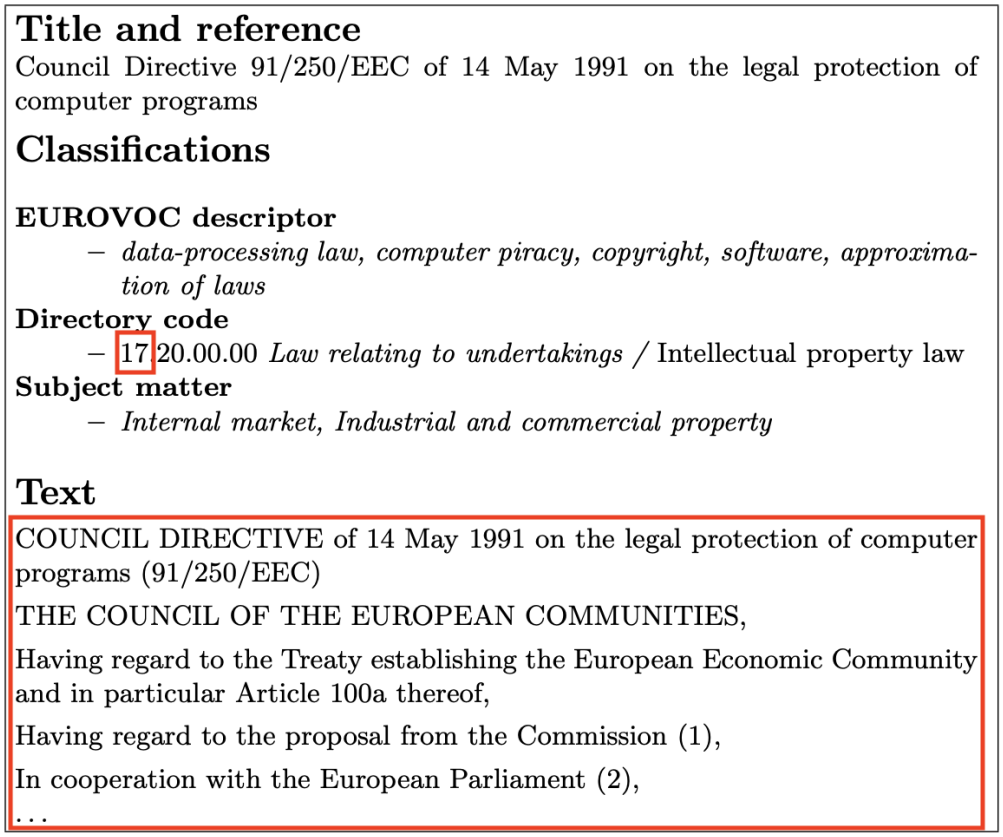
Mencia and Furnkranz [48] then proceed with the next steps as per the pipeline in Section 2.2, feature extraction and model selection. For feature extraction, they use TF-IDF vectorization. As a baseline model, they use the multinomial Naive Bayes classifier, and they compare its performance with the multilayer perceptron. All of these methods will be considered in this project (see Section 2.4), although they train a multilabel variant of the perceptron, which will not used (as will be noted below, the current project will not be a multilabel task). They report performance of these classifiers on several metrics: the 'is-error loss', 'one-error loss', 'ranking loss' and average precision. Only the latter metric will be used in this project, as the others relate only to the multilabel task.

3.3 Filtering the dataset

Having described the approach of the authors who compiled the EUR-Lex dataset, the focus will now be on the methods used in the current project. Our starting point is the full set of raw HTML documents scraped from the EUR-Lex portal by Mencia and Furnkranz [48].[13]

Although each document is classified in three different ways, the label of interest for this project is the directory code. Directory codes are organised in a 4-tier hierarchical structure, with 20 classes at the highest level, up to a maximum of 401 different classes at the lowest level. Each document must have at least one (and can have more than one) directory code at the highest tier, but may or may not have directory codes at the lower tiers.

[11] Available at http://www.ke.tu-darmstadt.de/files/resources/eurlex/eurlex_id2class.zip
[12] For their full list see http://www.ke.tu-darmstadt.de/files/resources/eurlex/english.stop
[13] Available at http://www.ke.tu-darmstadt.de/files/resources/eurlex/eurlex_download_EN_NOT.sh.gz



Title and reference

Council Directive 91/250/EEC of 14 May 1991 on the legal protection of computer programs

Classifications

EUROVOC descriptor

– *data-processing law, computer piracy, copyright, software, approximation of laws*

Directory code

– 17.20.00.00 *Law relating to undertakings* / Intellectual property law

Subject matter

– *Internal market, Industrial and commercial property*

Text

COUNCIL DIRECTIVE of 14 May 1991 on the legal protection of computer programs (91/250/EEC)

THE COUNCIL OF THE EUROPEAN COMMUNITIES,

Having regard to the Treaty establishing the European Economic Community and in particular Article 100a thereof,

Having regard to the proposal from the Commission (1),

In cooperation with the European Parliament (2),

...

Figure 3.3: Specific sections extracted from raw HTML document

A Python script is first written based on the BeautifulSoup parsing library to parse the body text of each raw HTML document based on their HTML tags.[14] Each document is then matched with its directory code(s), but only at the highest tier. For the avoidance of doubt, the relevant sections of each raw HTML document have been highlighted in Figure 3.3, and the rest of the information is ignored. Similar to the approach taken by Mencia and Furnkranz [48], the documents not containing text, containing corrigendums, or not in English are filtered out and discarded.

At this stage, each document has at least one highest-level directory code, but may have more than one such code. The documents are sorted according to the number of directory codes each document is labelled with, and the counts are summarised in Table 3.1.

Number of classes	Count
1	16169
2	2968
3	187
4	12

Table 3.1: Summary of number of classes per sample

[14] See *extract.py*

Datasets of documents with multiple labels are appropriate for *multi-label* multi-class classification tasks. However, the aim of this project is to examine a single-label, multi-class classification task (as noted in Section 2.1.1, this means each sample has only one of several possible labels). While it would have been possible to re-categorise documents having more than one label to any one of those labels, whether arbitrarily chosen or based on contextual knowledge of the legal subject matter, this was not considered an appropriate choice. A model may classify an unseen sample (which has had one or more of its class labels removed) to a certain class based on the features that sample has, which would have been a correct classification if not for that label having been removed. This would cause problems with assessing the performance of the model, and may even affect the mapping learned by the model. Thus, while it is not ideal to reduce the size of the dataset, all the documents having more than one label are discarded, and 16,169 samples remain. The data is then prepared and saved in a format which can easily be loaded into a 'pandas' dataframe for analysis.[15] We refer to this modified dataset as the 'distilled EURLEX dataset'.

3.3.1 Distribution of the distilled EURLEX dataset

Each sample thus has one of 20 possible labels from the classes in the EUR-Lex directory of legal acts.[16] Table 3.2 shows the distribution of samples in each class with their counts, together with the class descriptions. Figure 3.4 charts the distribution of samples across the 20 classes.

It is crucial to note at the outset that the dataset is imbalanced, as should be clear from the following figures. This will greatly affect the choice and analysis of performance metrics when critically evaluating models; for example, accuracy may not be the most useful metric in the case of an imbalanced dataset [69] (as noted in Section 2.1.5). In our case, samples in classes 3 and 11 form approximately 53% of the dataset, thus any model which can only accurately classify samples in those classes can already achieve 53% accuracy.

[15] See *data.csv*

[16] For the full list, see https://eur-lex.europa.eu/browse/directories/legislation.html

Class	Count	Description
1	698	General, financial and institutional matters
2	599	Customs Union and free movement of goods
3	4798	Agriculture
4	486	Fisheries
5	298	Freedom of movement for workers and social policy
6	156	Right of establishment and freedom to provide services
7	429	Transport policy
8	1227	Competition policy
9	200	Taxation
10	267	Economic and monetary policy and free movement of capital
11	3743	External relations
12	197	Energy
13	997	Industrial policy and internal market
14	432	Regional policy and coordination of structural instruments
15	617	Environment, consumers and health protection
16	154	Science, information, education and culture
17	64	Law relating to undertakings
18	415	Common Foreign and Security Policy
19	386	Area of freedom, security and justice
20	6	People's Europe
Total	16169	Total number of samples

Table 3.2: Distribution of samples and class descriptions

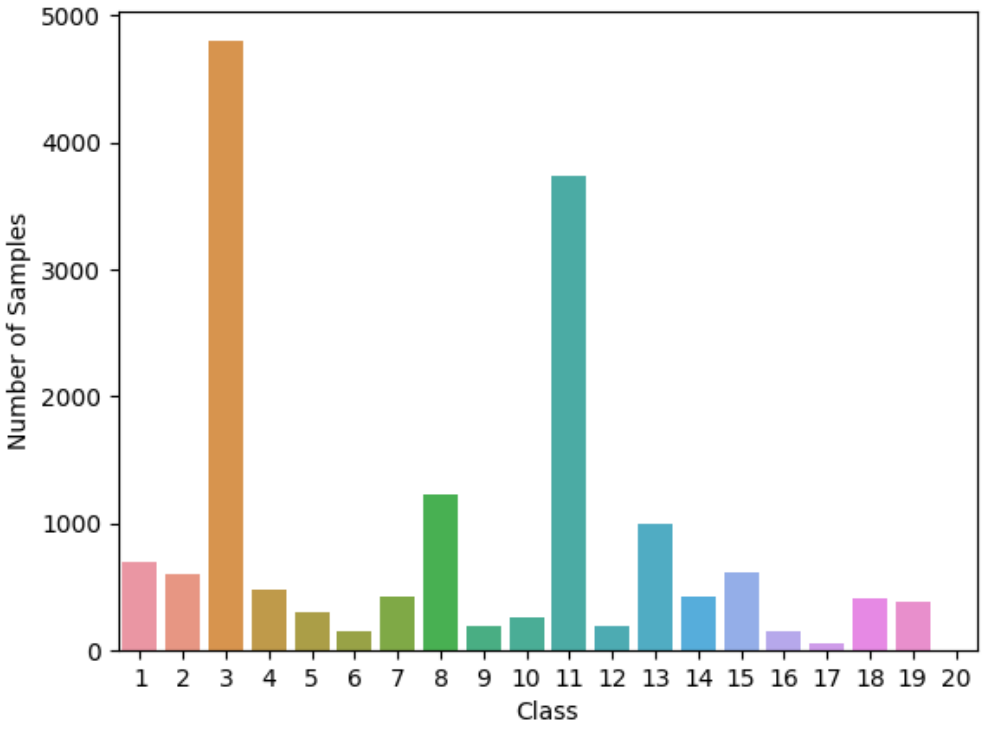


Figure 3.4: Chart showing distribution of samples

3.4 Analysis of the distilled EURLEX dataset

In this section, some further analysis of the dataset is provided, such as the most commonly occurring tokens in each class, and the average number of tokens and sentences per sample.[17] The aim of this is to provide information which can later be used in analysis, especially when providing hypotheses to explain the performance of 'black box' neural network models. As noted in Section 1.1, these characteristics of legal documents are particularly important, as they differentiate the task of text classification from that of analysis of an IMDB or Twitter dataset.

For the avoidance of doubt, an explanation of the calculation of these counts follows. Each sample in the dataset is preprocessed (for the detailed steps, see Section 4.2). Each sample can be split into sentences using the spaCy sentencizer. Each sample or sentence can also be split into tokens using the spaCy tokenizer. From this, the following calculations are provided, rounded off to the nearest integer:

$$average\ token\ count\ per\ sample\ in\ Class\ C_i = \frac{total\ number\ of\ tokens\ in\ Class\ C_i}{total\ number\ of\ samples\ in\ Class\ C_i}$$

[17] See *dataset.py*

$$average\ sentence\ count\ per\ sample\ in\ Class\ C_i = \frac{total\ number\ of\ sentences\ in\ Class\ C_i}{total\ number\ of\ samples\ in\ Class\ C_i}$$

$$average\ token\ count\ per\ sentence\ in\ Class\ C_i = \frac{total\ number\ of\ tokens\ in\ Class\ C_i}{total\ number\ of\ sentences\ in\ Class\ C_i}$$

Class	Tokens per sample	Sentences per sample	Tokens per sentence
1	1835	326	6
2	1015	127	8
3	851	121	7
4	852	121	7
5	1438	147	10
6	2468	235	10
7	1720	216	8
8	4466	442	10
9	767	75	10
10	1108	112	10
11	1176	146	8
12	922	105	9
13	2184	334	7
14	1532	172	9
15	1496	253	6
16	885	79	11
17	1969	194	10
18	714	84	9
19	1172	133	9
20	620	56	11

Table 3.3: Summary of average counts of samples in each class

Thus, while there is some variation in the numbers of tokens and sentences per sample across classes, it is not clear that a classifier will be able to accurately classify samples to certain classes simply based on the length of a sample, perhaps with the exception of samples from Class 8. In addition, to provide an overview of the vocabulary across the samples in the dataset, the top 20 most commonly occurring tokens in each class are compiled in Table 3.5. At first blush, it is also clear that the samples across classes share many tokens. We provide a few additional counts in Table 3.4, which we will refer to later.

Statistics from the entire dataset	
Average tokens per sample	1441
Number of tokens in longest sample	281204
Number of unique tokens in training set	261737
Number of unique tokens in test set	223842

Table 3.4: Additional information measured from the entire dataset

Class	20 most common tokens in each class
1	shall, article, 1, european, commission, 2, appropriation, 3, council, member, regulation, decision, commitment, payment, financial, community, committee, p.m., 4, p.
2	article, shall, 1, regulation, 2, good, community, product, custom, use, 3, member, commission, authority, annex, code, tariff, eec, import, apply
3	article, shall, 1, regulation, 2, commission, member, community, 3, product, directive, 1, p., oj, annex, ec, decision, states, animal, 4
4	shall, regulation, article, 1, vessel, fishing, member, commission, ec, 2, community, 3, states, catch, annex, fishery, oj, p., state, I
5	shall, member, article, 1, 2, regulation, benefit, state, person, institution, work, social, worker, 3, states, legislation, european, commission, employment, provision
6	shall, member, article, directive, state, authority, 1, undertaking, states, 2, competent, provide, provision, information, 3, institution, insurance, service, commission, contract
7	shall, article, 1, member, 2, commission, regulation, ship, state, 3, use, states, service, provide, directive, community, system, authority, transport, x
8	market, commission, aid, article, price, 1, state, agreement, product, company, party, decision, case, member, provide, million, service, cost, regard, measure
9	article, member, shall, 1, directive, tax, states, 2, good, community, council, state, 3, commission, person, supply, duty, apply, regard, value
10	article, shall, member, council, 1, european, states, economic, euro, programme, gdp, rate, market, 2, policy, commission, government, deficit, 3, regulation
11	shall, article, agreement, community, 1, product, 2, european, regulation, commission, council, annex, decision, 3, party, I, member, import, p., oj
12	shall, article, commission, energy, 1, member, 2, community, material, treaty, nuclear, states, european, party, 3, report, decision, use, state, measure
13	shall, 1, directive, 2, article, test, use, member, type, annex, 3, vehicle, product, commission, 4, european, (c), states, 5, approval
14	article, regulation, shall, commission, project, 1, community, member, state, assistance, financial, 2, eec, decision, expenditure, document, 3, single, programming, measure
15	shall, article, 1, member, directive, commission, community, 2, states, e, use, n, product, 3, annex, regulation, information, european, council
16	european, community, member, council, programme, commission, states, action, information, education, shall, article, 1, training, decision, 2, cultural, cooperation, development, activity
17	shall, article, 1, member, mark, community, 2, trade, office, application, 3, right, company, directive, states, audit, state, regulation, provide, registration
18	shall, article, european, 1, council, union, 2, member, regulation, eu, 3, operation, states, commission, action, decision, security, staff, annex, republic
19	shall, member, article, state, states, 1, european, 2, convention, council, decision, national, 3, authority, visa, information, (c), accordance, person, union
20	member, state, european, national, passport, article, states, 1, shall, community, union, page, candidate, right, residence, vote, citizen, 2, stand, people

Table 3.5: Top 20 most commonly occurring tokens in each class (ordered by frequency).
This is provided for an overview of some common examples of words which may occur in each class.
We can observe that many tokens appear in most or all of the classes,
thus TF-IDF vectorization may prove to be an effective form of feature representation.

3.5 Dataset visualisation

Here, we provide visualisations of the distilled EURLEX dataset.[18] As the count vectorizer and TF-IDF vectorizers result in different features, visualisations for both are provided. To generate these visualisations, we fit and transform the vectorizers on all the samples in the dataset. This resulted in a dimensionality of 261,737 (the size of the vocabulary of the entire dataset). Next, we apply *latent semantic analysis* (LSA) to reduce the dimensionality to 50D, using the TruncatedSVD tool provided in Scikit-learn, which works particularly well with sparse vectors, as is the current case. We then further reduce the dimensionality to 2D, using *t-distributed Stochastic Neighbor Embedding* (t-SNE). With the samples in 2D, we produced the visualisations. We use this two-step dimensionality reduction method as it is the recommended approach for our case.[19]

From a brief empirical observation of the visualisations which follow, we can observe some clusters in the data, with, for example, Classes 3 (green) and 11 (blue) being more prominent, and with Classes 8 and 18 (shades of grey). It should be noted, however, that most of these are classes with the highest sample counts, per Figure 3.4, thus they are bound to dominate the visualisation; further, it should also be noted that Class 13 is a similar shade of green, and Class 1 a similar shade of blue (despite experimenting with multiple colour palettes, it was difficult to choose one which had 20 completely distinct colours). We note that this is not meant to be an exact representation of the data in any way, given that the dimensionality has been reduced from 261,737 to 2 (an over 100,000-fold dimensionality reduction). Rather, the visualisations are provided merely as a form of reference.

[18] See *dataset.py*

[19] See Scikit-learn documentation at
https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

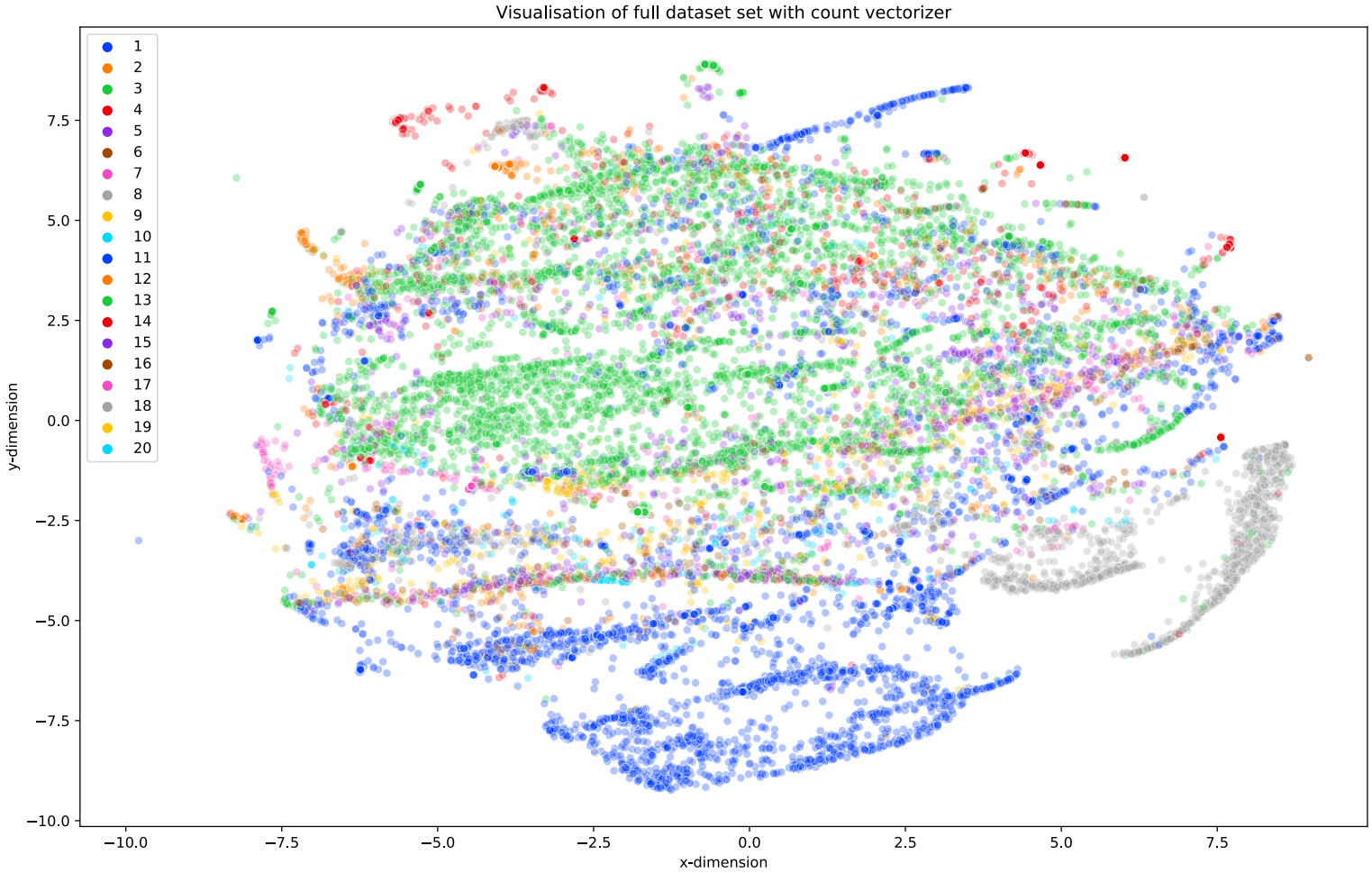


Figure 3.5: Visualisation of entire dataset with Count vectorizer for feature representation, with dimensions reduced to 2D with LSA and t-SNE. Each point on the plot represents a sample. Each sample's class label is denoted by its colour. There is some evidence of clustering in samples belonging to the same class.

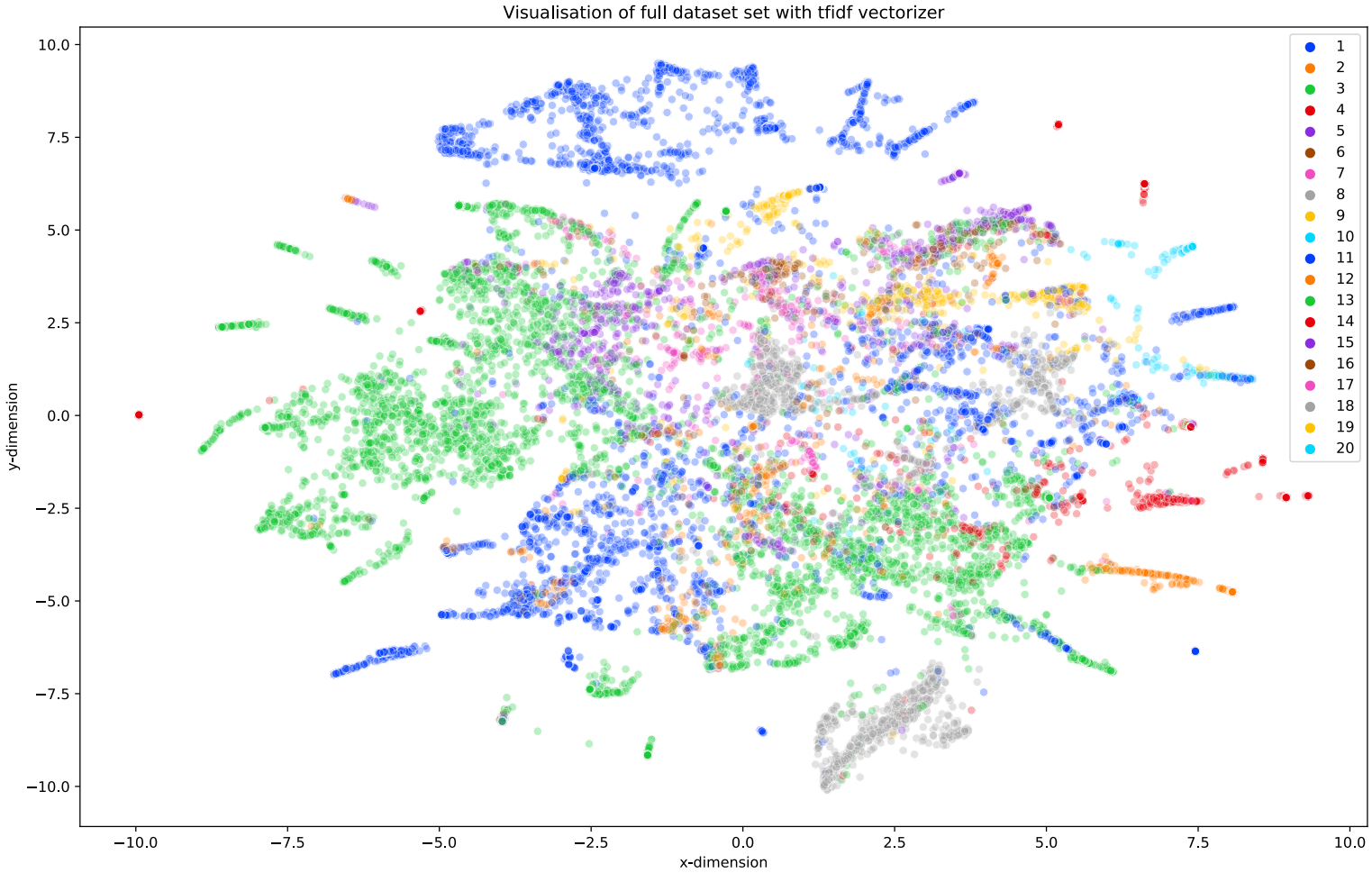


Figure 3.6: Visualisation of entire dataset with TF-IDF vectorizer. The aim is to show the difference in feature representation which could result from using a count or TF-IDF vectorizer.

Chapter 4

Experiments

4.1 Overview

In this chapter, we draw a distinction between non-deep learning models and deep learning models. We refer to the former as 'machine learning' models (see Section 2.4.1 to Section 2.4.7) and the latter as 'deep learning' models (see Section 2.4.8 to Section 2.4.10), although by definition all the models considered in this project can be described as machine learning models. We provide summaries of performance of the models in both sections separately, as the approach taken to feature representation in each section differs (count-based vectorizers vs. word embeddings). All models will be described and evaluated in line with the pipeline described in Section 2.2.

At the risk of repetition, we recall that we denote the dataset $D = \{X_1, X_2, ..., X_N\}$, where N is the total number of samples, X_i is a single sample consisting of one or more sentences, each sentence consists of one or more tokens, and each token consists of one or more characters. Each sample X_i belongs to only one of k possible classes, $C_i \in C$, where $C = \{C_1, C_2, ..., C_k\}$.

4.2 Preprocessing

The first step in the text classification pipeline is preprocessing. For consistency, the preprocessing steps taken for text input to all models will be the same, unless otherwise stated. First, each character in each sample X_i is lowercased. As noted above, this is a common standard step in preprocessing in text classification tasks, since it is assumed that the meaning of a word does not change simply because it is uppercased or lowercased [73]. In the legal context, however, it must be recognised that capitalised words in the middle of a sentence may indeed carry a different meaning. For example, legislation may appear in the middle of a sentence, such as "Article 100a". This refers to a specific Article in legislation, and thus carries a special meaning. However, looking back to Figure 3.3, we can see that capitalisation is not used consistently in the EURLEX dataset. Some words in the preamble are completely capitalised, e.g. "COUNCIL DIRECTIVE", when they would not be semantically different to 'Council Directive'. Similarly, whole phrases in the preamble also tend to be capitalised, such

as "THE COUNCIL OF THE EUROPEAN COMMUNITIES", although they may semantically carry the same meaning when not capitalised. Thus, for consistency, we employ lowercasing to all text in our preprocessing stage.

Next, the tokens which are stopwords and punctuation marks are removed using the spaCy library and stopword list, another standard preprocessing step in NLP [73].[20] In addition, we remove three commonly occurring non-English special characters, since we are concerned with an English dataset in this project. An exception to punctuation removal is in the case of sentence embeddings, because we need the punctuation marks to determine the beginning and ending of sentences; this is explained in Section 4.6.

Finally, we do not stem or lemmatise, based on research that word normalisation can have a negative impact on English text classification [70, 73]. Although these are limited studies only on specific datasets, we take the view that many of the words occurring in the dataset are best left in their original lexical form, given the fragility of meaning that words can have in a legal context. We then tokenise each sample, and thus, each sample X_i then consists of a list of tokens.

4.3 Machine learning models

For the machine learning models, all implementation is achieved through the Scikit-learn library [56] in Python.[21] Each sample in D is vectorized using a count vectorizer or TF-IDF vectorizer. D is then split into training and test sets in an 80:20 split. The training set is then split into 5 equal sets, with one set held out as a validation set and the remainder used for training. This is done until each of the 5 sets have been used as a validation set, following the 5-fold cross validation process. Either a manual search or a random search is conducted for hyperparameter optimization, and the validation loss is monitored throughout this process. With the best hyperparameters for each classifier, we then report the performance of each classifier on various metrics on the test set.

[20] For the full list see *https://github.com/explosion/spaCy/blob/master/spacy/lang/en/stop_words.py*
[21] See *machine_learning_models.py*

4.4 Summary of results

Classifier	Accuracy/Micro-F1		Macro-F1	
Vectorizer	**Count**	**TF-IDF**	**Count**	**TF-IDF**
Naive Bayes	0.887	0.903	0.780	0.811
Decision tree	0.838	0.839	0.671	0.665
Random forest	0.845	0.846	0.693	0.700
Logistic regression	0.935	0.918	0.888	0.801
k-NN	0.888	0.925	0.829	0.885
SVM (linear)	0.929	**0.956**	0.880	**0.930**
SVM (non-linear)	0.712	0.945	0.477	0.859
MLP	**0.953**	0.953	**0.921**	0.920

Table 4.1: Summary of performance of machine learning models. The MLP shows the best performance with the count vectorizer, and linear SVM is the best classifier with the TF-IDF vectorizer.

4.5 Analysis of results

4.5.1 Naive Bayes classifier

Smoothing	Laplace ($\alpha = 1$)		None ($\alpha = 0$)	
Vectorizer	**Count**	**TF-IDF**	**Count**	**TF-IDF**
Accuracy/Micro-F1	0.840	0.581	0.887	0.903
Macro-F1	0.657	0.174	0.780	0.811

Table 4.2: Performance of multinomial Naive Bayes, also showing the effect of Laplace smoothing.

As noted above, the multinomial Naive Bayes classifier is often used in text classification [60], and a variant for the multilabel case is also the baseline classifier used by Mencia and Furnkranz [48]. Based on a manual search, the hyperparameters of the classifiers were tuned.

We note an interesting observation that the use of Laplace smoothing, as per Equation 2.18, very severely worsens the performance of the classifier where the TF-IDF vectorizer is used, despite the fact that it is generally used to improve the performance of the model by avoiding zero probabilities. The impact is reflected in Table 4.2 above. We note that an accuracy of 58.1% is especially poor given, as noted in Section 3.3.1, 53% of the entire dataset consists of samples from only 2 out of the 20 classes.

In order to explain this anomaly, we have to examine the exact workings of the vectorizers as described in Sections 2.3.1 and 2.3.2. For clarity, the following is an exact

representation of the first sample in the training set, vectorized with the count and TF-IDF vectorizers and returned as sparse vectors:

Count	TF-IDF
(0, 1089), 1	(0, 1089), 0.004123345451722612
(0, 1872), 1	(0, 1872), 0.004562543389171481
(0, 4843), 2	(0, 4843), 0.0031717298187008036
(0, 5654), 2	(0, 5654), 0.003371296518197432
(0, 6532), 1	(0, 6532), 0.0015712004553782516
(0, 7320), 2	(0, 7320), 0.00381386999440877
(0, 7913), 3	(0, 7913), 0.005850604591265597
(0, 8455), 3	(0, 8455), 0.005649215635463998
(0, 8656), 1	(0, 8656), 0.004747057644849401
...	...

Table 4.3: Feature representations of first sample in training set

This means that the sample has 1 count of unique token number 1089 in the vocabulary, and its corresponding TF-IDF value is 0.00412..., and so on. However, we can see that the TF-IDF counts are extremely small. The probability in Equation 2.17 is based on the calculation of conditional probability in Equation 2.16, which is calculated from the probability of each feature of a vectorized sample occurring in that vector. The probability of each feature occuring in a vector is computed by taking the TF-IDF of each individual feature as a proportion of the total TF-IDF counts in that vector. Thus, given that the TF-IDF values for each feature are so small because the total dataset vocabulary is huge, as shown above, it is perhaps the case that Laplace smoothing, which adds $\alpha = 1$ to the posterior probability, skews the probabilities by far too much.

		1 epoch					2 epochs			5 epochs			10 epochs		
		FC	MLNB	BR	MMP	DCMLPP	BR	MMP	DCMLPP	BR	MMP	DCMLPP	BR	MMP	DCMLPP
subject matter	IsErr ×100	99.58	99.47	65.99	55.70	51.38	58.78	51.96	44.07	53.42	42.77	38.23	50.19	40.22	36.34
	OneErr ×100	77.83	98.68	35.71	30.58	22.78	27.13	27.09	17.29	22.69	18.38	13.49	20.64	15.97	12.55
	RankLoss	12.89	8.885	17.38	2.303	1.064	13.89	2.520	0.911	11.58	2.091	0.796	9.752	1.85	0.762
	Margin	40.16	25.04	62.31	10.11	4.316	52.28	11.22	3.757	44.77	9.366	3.337	38.45	8.177	3.214
	AvgP	22.57	11.91	59.33	74.01	78.68	66.07	76.95	82.73	70.69	82.10	85.64	73.30	83.75	86.52
directory code	IsErr ×100	91.51	99.34	52.80	47.68	36.55	46.26	40.01	32.38	40.76	33.28	29.22	37.55	31.39	28.30
	OneErr ×100	90.13	99.04	44.40	40.85	28.22	37.38	32.99	24.42	31.48	25.79	21.41	28.1	23.9	20.65
	RankLoss	14.17	7.446	19.40	2.383	0.972	15.09	2.058	0.863	11.69	1.874	0.824	9.876	1.529	0.815
	Margin	68.33	34.44	96.43	14.18	5.626	77.32	12.18	5.045	61.48	10.95	4.831	52.94	8.947	4.785
	AvgP	18.98	6.714	57.10	68.70	77.89	63.68	74.90	80.87	68.75	79.84	82.87	71.61	81.30	83.38
EUROVOC	IsErr ×100	99.82	99.82	99.25	99.14	98.20	98.70	98.00	96.75	97.46	96.14		97.06	95.13	
	OneErr ×100	93.52	99.58	53.11	78.98	34.76	44.93	56.88	28.01	36.69	39.46		33.84	34.99	
	RankLoss	12.97	22.34	39.78	3.669	2.692	35.25	4.091	2.398	30.93	4.573		28.59	4.509	
	Margin	1357.10	1623.72	3218.12	562.81	426.28	3040.01	670.65	387.51	2846.47	757.01		2716.63	740.12	
	AvgP	5.504	1.060	25.55	27.04	46.79	30.71	38.42	52.72	35.95	47.65		38.31	50.71	

Figure 4.1: Reproduction of results from Table 3 in Mencia and Furnkranz [48]

This may be interesting in light of the performance of the multilabel variant of the multinomial Naive Bayes classifier ('MLNB', as highlighted) used by Mencia and Furnkranz [48], who represent each sample using a TF-IDF vectorizer throughout their paper. From a table of their results reproduced in Figure 4.1, we can see that their MLNB baseline had a precision of a paltry 6.714%, compared to much higher

scores for all the other classifiers. Unfortunately, from their paper, it is not clear if they used Laplace smoothing, which could explain the low precision score.

Regrettably, a direct comparison cannot be made between their paper and the current project, because their task is an extreme multi-label classification task. Even for classifications according to directory code, there are over 500 possible classes, compared to 20 in the current case. However, the especially poor performance of the Naive Bayes classifier in their task was highlighted, as it seemed to be interesting in light of the effect of Laplace smoothing on the TF-IDF vectorization.

Overall, the performance of the Naive Bayes classifier in the present case with TF-IDF vectorization but without smoothing, with 90.3% accuracy, prima facie suggests two things: first, that the conditional independence assumption that the occurrence of every feature in a sample is independent of other features partly holds true; and second, that the feature vectors of the test set are to a considerable degree linearly separable, given that the Naive Bayes classifier is a linear classifier, as noted in Section 2.4.1.

4.5.2 Decision tree

	Count	**TF-IDF**
Accuracy/Micro-F1	0.838	0.839
Macro-F1	0.671	0.665
Split criterion	Gini	Gini
Min. samples per leaf	1	1
Min. samples per split	2	2
Node count	2709	2409
Leaves count	1355	1205
Tree depth	53	61

Table 4.4: Similar performance with both methods of feature representation with decision trees

First, decision trees with both the count and TF-IDF vectorizer were ran. The criterion for splitting at each node, gini impurity and information gain, as defined in Section 2.4.2, were experimented with. In both vectorizers, the trees achieved better performance splitting based on gini impurity. Based on a manual search with reference to the initial trees, the hyperparameters of the decision tree were tuned, such as setting the maximum depth of the tree, the minimum number of samples in each split, and the minimum number of samples in each leaf. The intuition was that this would be especially important since, as noted above, decision trees are prone to overfitting. However, setting the maximum depth of the tree to various manually selected values, such as 50, 45, and 40, i.e. *pre-pruning* the tree, did not improve the performance of the classifier, and in fact marginally worsened it. In addition, using a count or TF-IDF vectorizer also does not greatly affect the performance of a decision tree classifier.

Table 4.4 summarises the characteristics of the decision trees with both the count vectorizer and TF-IDF vectorizer. A visualisation of the complete count vectorizer-based tree, which is marginally the better-performing tree in terms of macro-F1, is also provided.[22] As the complete tree is far too large to be analysed, a few levels of the tree, up to a depth of 5, where leaf nodes begin to appear in some branches, have been reproduced in Figure 4.2. For the avoidance of doubt, this is the decision tree of depth 53 with most of the branches of the tree cut off, not a tree with a maximum depth of 5. The visualisation provides an index referring to a token in the vocabulary of the count vectorizer, the gini impurity at each split, and the number of samples remaining in total and in each class, at each node. For ease of explanation, a corresponding figure with each of the exact features on which the splits were determined is also provided in Figure 4.3. The colours of the nodes reflect the distribution of the classes of the samples remaining at that node, thus, for example, the greater the proportion of samples belonging to Class 3 in a node, the yellower the colour of that node.

It can be seen from Figure 4.3 that the token 'agreement' is at the root of the decision tree. Thus, the decision rule "Does the token 'agreement' appear in the sample less than twice?" results in the greatest reduction in gini impurity. Relating this to Table 3.5, we can see that the token 'agreement' is the third-most common token in Class 11. This makes sense when read with Figure 4.2, which shows that the response 'False' to the decision rule only reduces the number of samples remaining in Class 11 from 2994 to 2508. In other words, the token 'agreement' appears more than twice in 2508 samples in Class 11. It should thus be clear that one of the key advantages of a decision tree is that each and every decision rule can be examined.

Overall, the decision tree classifier is not as effective as other classifiers, as shown by its accuracy and macro-F1. This is to be expected, given that the decision tree is generally regarded as an *unstable* classifier. This means that small differences between the training samples and the test samples may cause dramatically large changes in the decision tree's classification rules [43]. As noted in Table 3.3, the samples in the dataset in each class are of varying length. Even though the test set was split from the complete dataset with stratified sampling, i.e. the proportion of samples belonging to each class in the test set remains the same as that of the training set, it should be noted that even within each individual class, each sample may also be of varying lengths and features. This is likely to affect the performance of an unstable classifier such as the decision tree. Thus, a random forest classifier was tested next to determine if this would improve the performance of the decision tree.

[22] See *decision_tree.pdf*

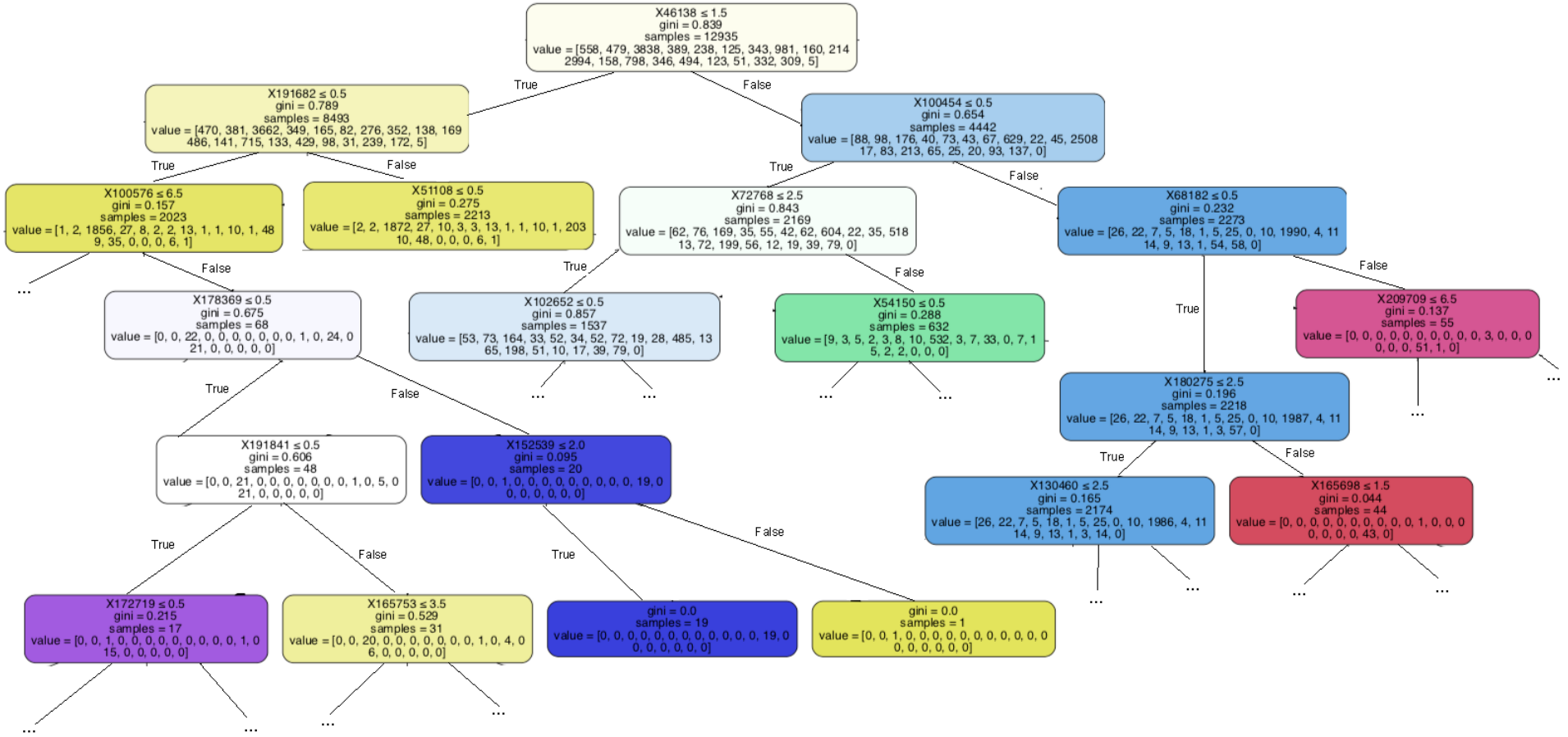


Figure 4.2: Partial visualisation of decision tree with the count vectorizer used for feature representation. Each box is a node. Leaf nodes begin to appear at a depth of 5. Each node contains the decision rule which refers to the counts of a specific vocabulary index in the vectorizer (e.g. $X46138 <= 1.5$), gini impurity at that node, number of samples remaining in total, and number of samples per class remaining in that node.

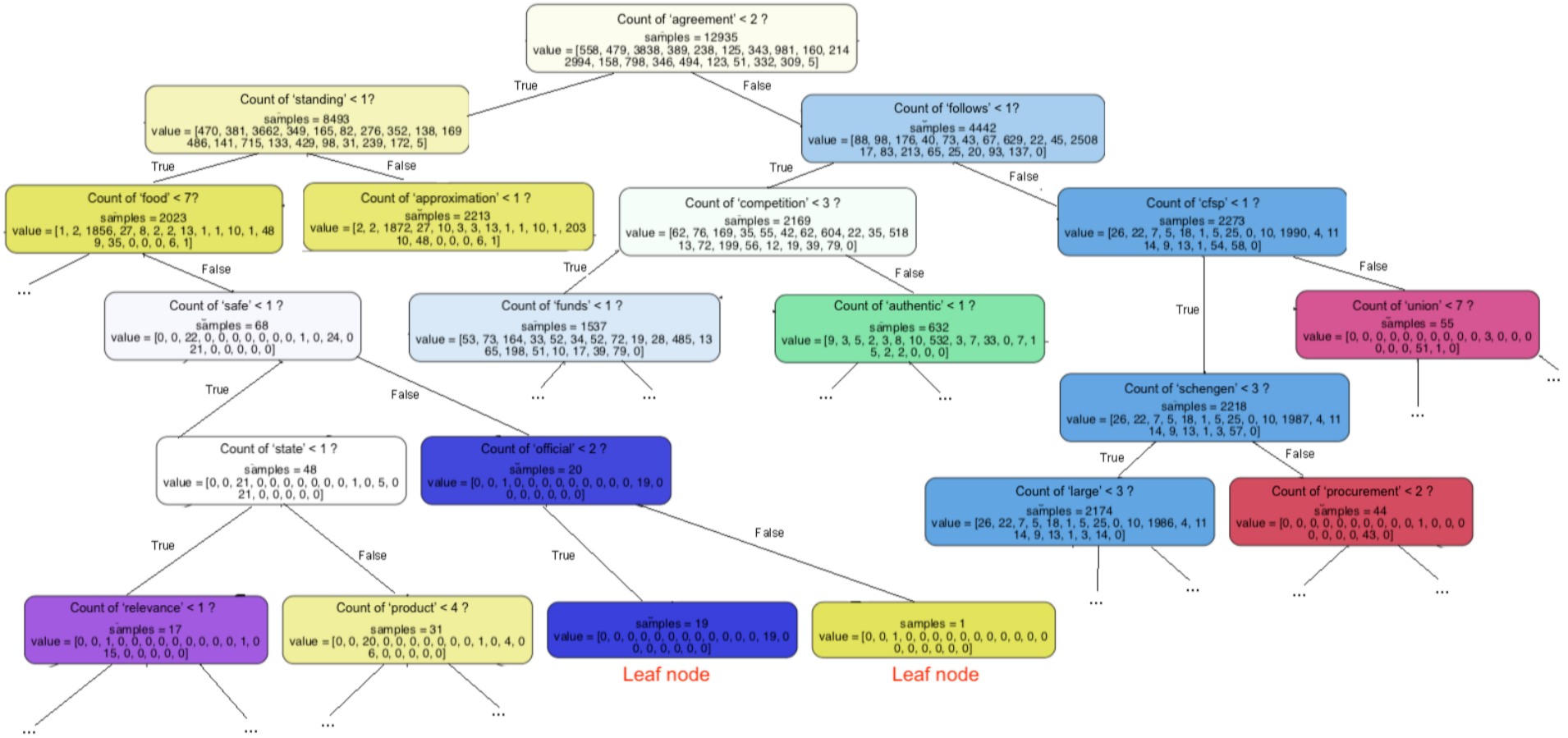


Figure 4.3: The partial visualisation above, with the vocabulary indices in the decision rule replaced with the actual tokens.

4.5.3 Random forest

	Count	TF-IDF
Accuracy/Micro-F1	0.845	0.846
Macro-F1	0.693	0.700
Split criterion	Gini	Gini
Min. samples per leaf	1	1
Min. samples per split	1	1
Bootstrap aggregating	False	False
Max depth	None	None
Number of trees	100	100
5 most important tokens	treaty, economic, decide, agreement, hereinafter	committee, treaty, agreement, measure, follows

Table 4.5: Similar performance with both methods of feature representation with random forest. Statistical testing confirmed that there was no significant difference between the performance of the decision tree and random forest.

As noted in Section 2.4.3, the random forest is an ensemble based on multiple decision trees. Thus, the expectation is that it will outperform the single decision tree. As there were many hyperparameters to be tuned, a random search within a manually defined search space was conducted for hyperparameter tuning. The best hyperparameters chosen are summarised above. As expected, as with the decision tree, splits were determined on gini impurity rather than information gain. It should also be noted, however, that on a random search, not using bootstrap aggregating provided better results, thus every sample in the training set was used to train each tree. This goes against the intuition stated above that bootstrap aggregating would stabilise the decision tree.

A visualisation of a randomly selected decision tree out of the 100 trees, with samples vectorized with the TF-IDF vectorizer as input, is provided.[23] Due to its similarity with the decision tree visualised in Section 4.5.2, it is not reproduced here.

Overall, the random forest classifier only marginally outperforms the single decision tree, regardless of whether samples are vectorized with the count vectorizer or TF-IDF vectorizer. Comparing the case of the classifiers with the TF-IDF vectorizer, we see that the decision tree resulted in an accuracy of 0.839, compared to that of the random forest with 0.846. Because at first glance this seems to be a very similar performance, and given that a random forest is generated with the random subspace method, i.e. each time the algorithm is run could result in slightly different results, we perform a McNemar's test (as explained in Section 2.1.5.6), to determine if the difference in

[23] See *random_forest.pdf*

performance between the random forest and decision tree is statistically significant or not. This will allow us to decide if the random forest outperforms the decision tree.

	Random forest (correct)	Random forest (wrong)
Decision tree (correct)	2462 [“A”]	273 [“B”]
Decision tree (wrong)	240 [“C”]	259 [“D”]

Table 4.6: Contingency table of decision tree and random forest

The contingency table for the test is outlined above. For clarity, at risk of repetition we recall that a ‘correct’ result in the table is one which is a True Positive, i.e. the label predicted by the classifier matches the actual label of the sample, any other output is ‘wrong’. For clarity, each box above has been alphabetically labelled, just as in Table 2.3.

Our null hypothesis, H_0, is: “the performance of both classifiers is equal”. Computing McNemar’s test statistic per Equation 2.8, we obtain a χ^2 value of 1.996. Following a rule-of-thumb, we set the significance level, $\alpha = 0.05$, thus χ^2 <3.84, and we *cannot* reject H_0. Further, following a two-tailed test per Equation 2.9, we obtain a p-value of 0.158. Thus, as $p > \alpha$, we again have 95% confidence that we *cannot* reject H_0. This leads us to the perhaps surprising conclusion that the random forest classifier does not outperform the decision tree.

From this, we note at this point that the performance of the Naive Bayes classifier, a linear classifier, has outperformed that of the decision tree and random forest, which are non-linear classifiers.

4.5.4 Logistic regression

	Count	**TF-IDF**
Accuracy/Micro-F1	0.935	0.918
Macro-F1	0.888	0.801
Solver	L-BFGS	L-BFGS
Max iterations	1,000	1,000
Classifier	One-vs-rest	One-vs-rest
Penalty	L2 regularization	L2 regularization
Inverse regularization strength	1.0	1.0

Table 4.7: Summary of logistic regression

As noted in Equation 2.19, the logistic regression classifier learns a vector of weights and biases to be applied to an input vector X. Thus, the equation can be expanded to:

$$z = (w_1, w_2, ..., w_j) \cdot (x_1, x_2, ..., x_j) + b \tag{4.1}$$

Each term in the vector of weights thus corresponds to a term in the input vector X, which is a vector in which each element represents the counts of a unique token in the vocabulary of the training set. This means that when the classifier has been trained, we can find the exact weights applied to each feature (i.e. each token) in the vocabulary, by examining each element of the vector w. By sorting the vector and finding its *argmax* and *argmin*, and we can easily find out the individual tokens which the classifier places the greatest weight on, and those that the classifier places the least weight on, for the purposes of classification for each class. The 5 highest-weighted tokens and lowest-weighted tokens for each class, along with the corresponding coefficient values, are provided in Tables 4.8 and 4.9.

The highest-weighted tokens are of interest because they are the tokens which serve as the best indicator of a sample belonging to a certain class. For example, if the token 'fishery' appears in a sample, it is then very likely that the sample in which it appears belongs to Class 4. Similarly, however, the lowest-weighted tokens are also important, because they tell us the converse, which is also useful for classification: for example, if the token 'tariff' appears in a sample, then it is very likely that the sample does *not* belong to Class 4. Concretely, we can hypothesise that the classes with many heavily-weighted tokens are likely to have many True Positives, and are likely to have a high accuracy, and those with many lowly-weighted tokens are likely to have many True Negatives.

From a cursory observation of the descriptions and the most important tokens of each class, it appears clear that there is some similarity between the two. Taking Class 5 as an example, the most heavily weighted tokens in the 'Freedom of movement for workers and social policy' class are 'social', 'worker', 'labour', 'employment', and 'administrative', all of which relate to the class description. Similarly, the numbers which appear also tend to refer to important articles related to the subject matter of the classes. For example, Class 8 refers to 'Competition policy', and one of the most important tokens is '85'. Article 85 of the EEC Treaty is a key piece of legislation in European competition law [24]. Other tokens which may not appear meaningful at first glance may also be related to the subject matter of the class on closer examination. For example, in Class 3 which is 'Agriculture', two of the heaviest-weighted tokens are 'franz' and 'fischler', both of which have a coefficient weight of 0.648. With some background research, we can find out that Franz Fischler was in fact the European Union's Commissioner for Agriculture.

In addition, Figure 4.6 below shows, for each class, the distribution of the logistic

[24] For the full text, see
https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:12002E081&from=EN

regression weights. Each point on the bubble plot is a single weight (i.e. coefficient) in relation to a feature. From the top-end and bottom-end tails of the plots of the weights for each class, it can be seen that most of the classes have several tokens which are particularly important for the purposes of classification. The 5 points at the top-end and tail-end of each class are the tokens listed in Tables 4.8 and 4.9.

We can relate Figure 4.6 to the normalised confusion matrix in Figure 4.5. Taking Class 17 as an example, in Figure 4.6, we can see that the plot does not show a wide distribution compared to the other classes; it has short top-ends and tail-ends. This correlates with the normalised confusion matrix, which shows that the classifier only has a 0.538 classification rate in Class 17.

Thus, analysing the figures provided in this manner hopefully explains to a certain extent the effectiveness of the logistic regression classifier, despite the fact that each text sample is simply represented by a count-based vectorizer which merely counts the frequency of occurrences of each token, but does not take into account the contexts in which each token occurs, or provide a representation which reflects the linguistic meaning of each word (unlike word embeddings). Although many of the same tokens occur in all of the classes, as noted in Table 3.5, the subject matter of each class is sufficiently distinct such that tokens occurring in certain contexts, such as fishing, are unlikely to appear in the context of any of the other classes, making logistic regression with a count-based vectorizer an effective option for many of the classes.

Further, we note that, as stated above in Section 2.4.4, the logistic regression classifier can be considered a linear model [59]. Thus, we observe that the linear models, the Naive Bayes classifier and logistic regression, have so far outperformed the non-linear models, the decision tree and random forest.

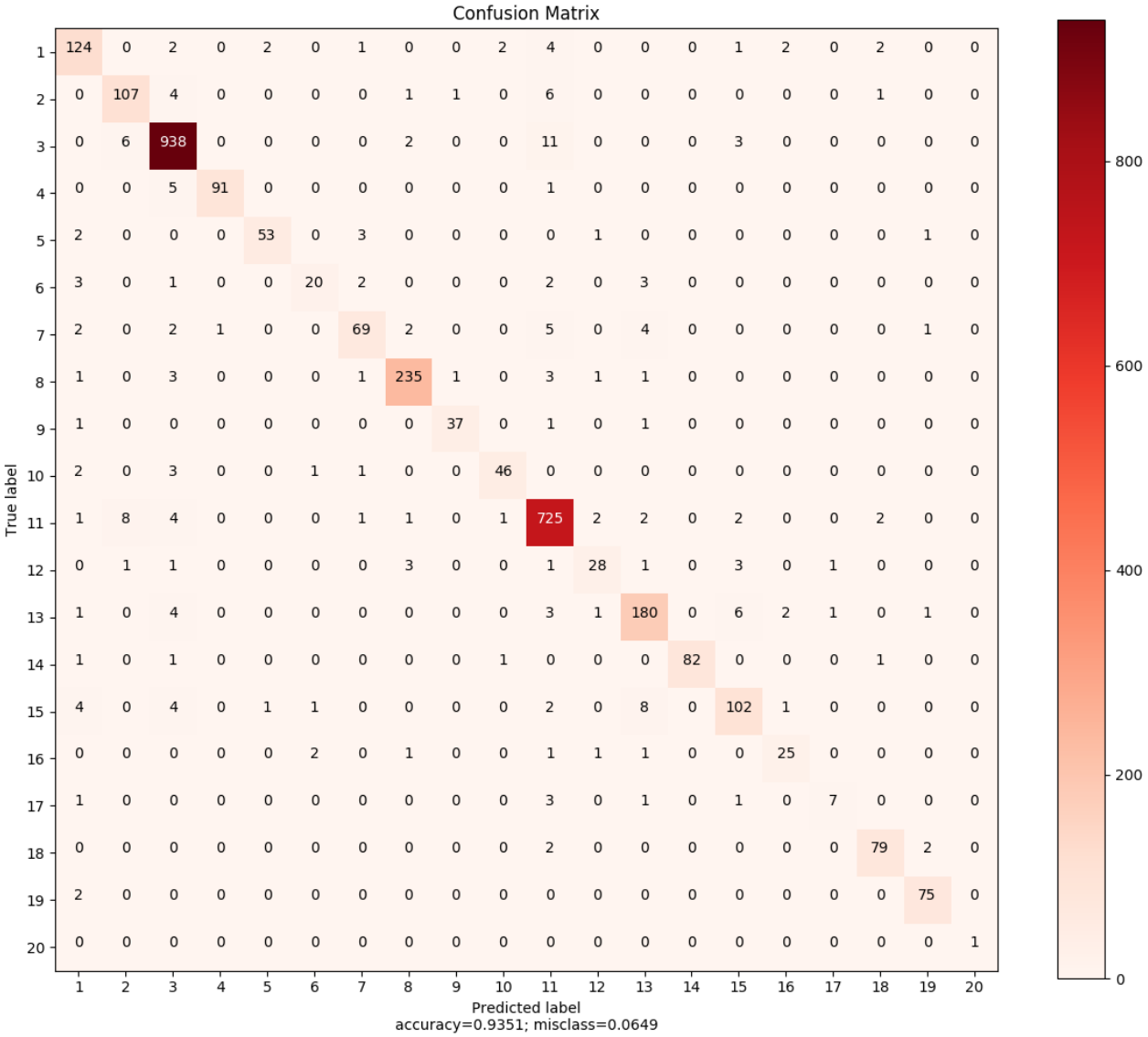


Figure 4.4: Logistic regression confusion matrix

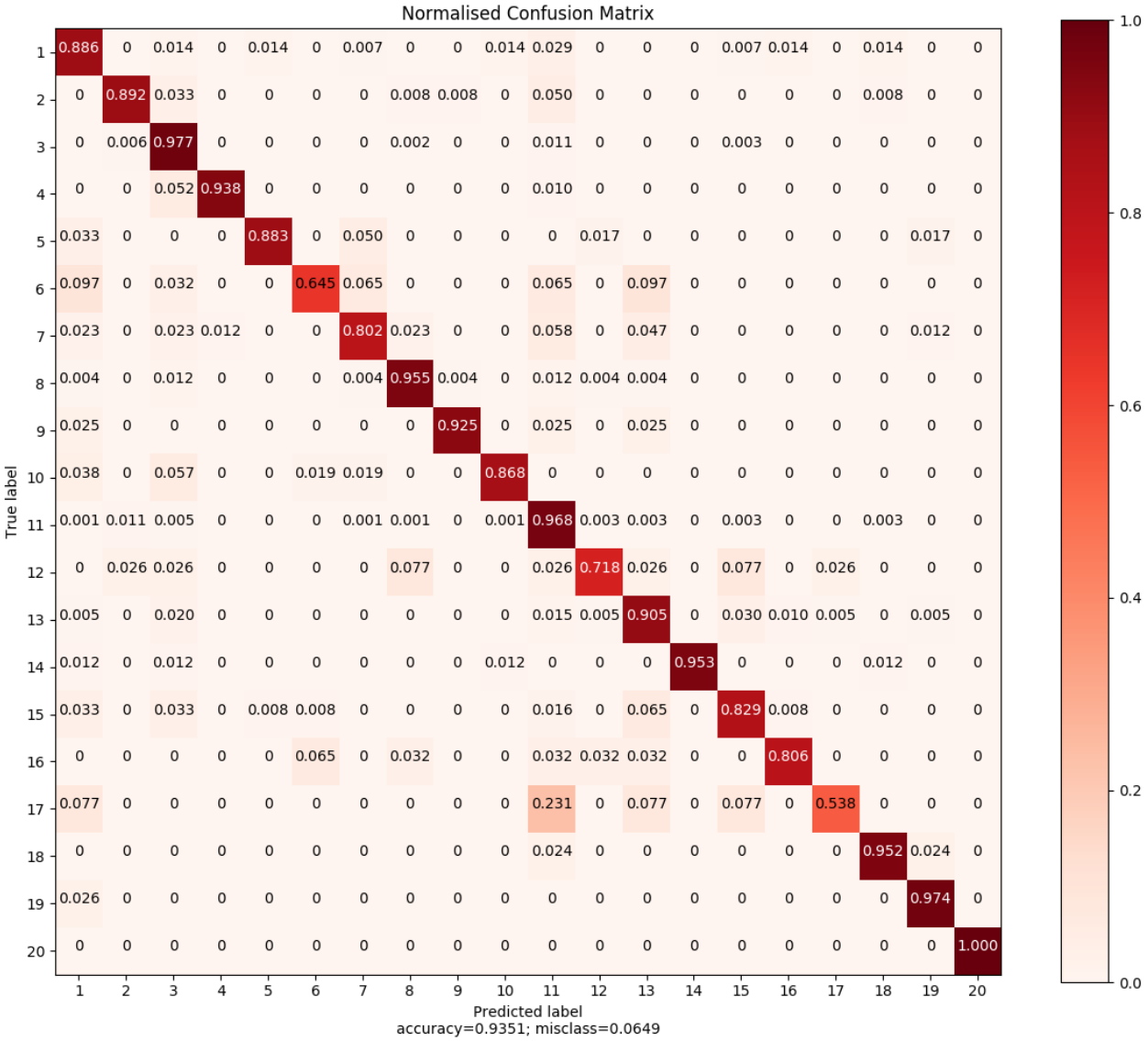


Figure 4.5: Logistic regression normalised confusion matrix

Class	Description	5 greatest and least weighted tokens (and corresponding weights)				
1	General, financial and institutional matters	euratom	appoint	rules	ecsc	gender
		0.981	0.769	0.554	0.499	0.451
		joint	common	association	agreement	latvia
		-0.647	-0.604	-0.482	-0.481	-0.471
2	Customs Union and free movement of goods	customs	iceland	transit	113	administration
		0.562	0.511	0.493	0.490	0.463
		union	conclude	fruit	management	programme
		-0.608	-0.467	-0.463	-0.426	-0.425
3	Agriculture	standing	fruit	franz	fischler	veterinary
		1.387	0.721	0.648	0.648	0.640
		2092	joint	government	foodstuff	936
		-0.821	-0.709	-0.625	-0.615	-0.568
4	Fisheries	fishery	fishing	fisheries	515	83
		1.146	0.679	0.612	0.370	0.346
		tariff	health	clean	protocol	493
		-0.261	-0.252	-0.242	-0.239	-0.227
5	Freedom of movement for workers and social policy	social	worker	labour	employment	administrative
		0.627	0.586	0.544	0.502	0.467
		purpose	oj	2002	common	aid
		-0.302	-0.279	-0.255	-0.248	-0.243
6	Right of establishment and freedom to provide services	statement	tourism	respect	entity	insurance
		0.525	0.459	0.393	0.372	0.351
		union	decide	eea	1988	1999
		-0.282	-0.239	-0.230	-0.227	-0.224
7	Transport policy	carriage	road	shipping	transport	drive
		0.708	0.596	0.547	0.525	0.454
		agreement	price	animal	health	trade
		-0.349	-0.340	-0.321	-0.296	-0.287
8	Competition policy	treaty	1962	85	17	notify
		0.460	0.394	0.354	0.344	0.299
		council	vessel	transport	annex	apply
		-0.256	-0.241	-0.223	-0.214	-0.210
9	Taxation	excise	388	latvia	tax	sixth
		0.699	0.442	0.410	0.394	0.381
		estonia	czech	aid	european	concern
		-0.359	-0.311	-0.263	-0.258	-0.216
10	Economic and monetary policy and free movement of capital	monetary	economic	deficit	investment	issue
		1.074	0.406	0.398	0.351	0.332
		provide	directive	joint	employment	protocol
		-0.271	-0.247	-0.222	-0.212	-0.202

Table 4.8: 5 greatest and least weighted tokens (and corresponding coefficients) in each class

Class	Description	5 greatest and least weighted tokens (and corresponding weights)				
11	External relations	quantitative	textile	eea	384	3906
		0.803	0.627	0.603	0.578	0.578
		transit	cfsp	spain	fruit	salad
		-0.681	-0.677	-0.663	-0.586	-0.578
12	Energy	energy	euratom	research	crude	transit
		0.680	0.505	0.377	0.371	0.336
		bradwell	eec	data	ec	18
		-0.543	-0.455	-0.358	-0.354	-0.339
13	Industrial policy and internal market	approximation	foodstuff	trans	representatives	approval
		0.887	0.463	0.460	0.446	0.438
		import	brussels	meeting	79	articles
		-0.422	-0.412	-0.408	-0.339	-0.332
14	Regional policy and coordination of structural instruments	structural	funds	1260	objective	specific
		0.438	0.433	0.393	0.349	0.329
		country	directive	agreement	financial	condition
		-0.227	-0.220	-0.211	-0.200	-0.199
15	Environment, consumers and health protection	2092	2037	environment	pollution	waste
		0.758	0.568	0.553	0.522	0.504
		education	veterinary	joint	condition	residue
		-0.453	-0.426	-0.421	-0.384	-0.366
16	Science, information, education and culture	culture	cultural	education	archive	schools
		0.746	0.575	0.521	0.435	0.335
		union	health	republic	agreement	regulation
		-0.315	-0.311	-0.300	-0.287	-0.250
17	Law relating to undertakings	560	mark	protection	da	40
		0.278	0.261	0.256	0.250	0.243
		2004	use	institution	committee	information
		-0.195	-0.182	-0.177	-0.172	-0.168
18	Common Foreign and Security Policy (CFSP)	cfsp	union	position	2368	entity
		0.913	0.652	0.488	0.462	0.389
		november	eec	opinion	product	ii
		-0.382	-0.376	-0.319	-0.263	-0.254
19	Area of freedom, security and justice	europol	sch	schengen	asylum	vi
		0.641	0.568	0.534	0.422	0.381
		eec	cfsp	comunity	economic	commission
		-0.496	-0.449	-0.421	-0.411	-0.314
20	People's Europe	passport	resolution	1981	june	citizen
		0.344	0.269	0.218	0.209	0.178
		commission	agreement	shall	regulation	information
		-0.178	-0.106	-0.094	-0.086	-0.080

Table 4.9: 5 greatest and least weighted tokens (and corresponding coefficients) in each class (continued)

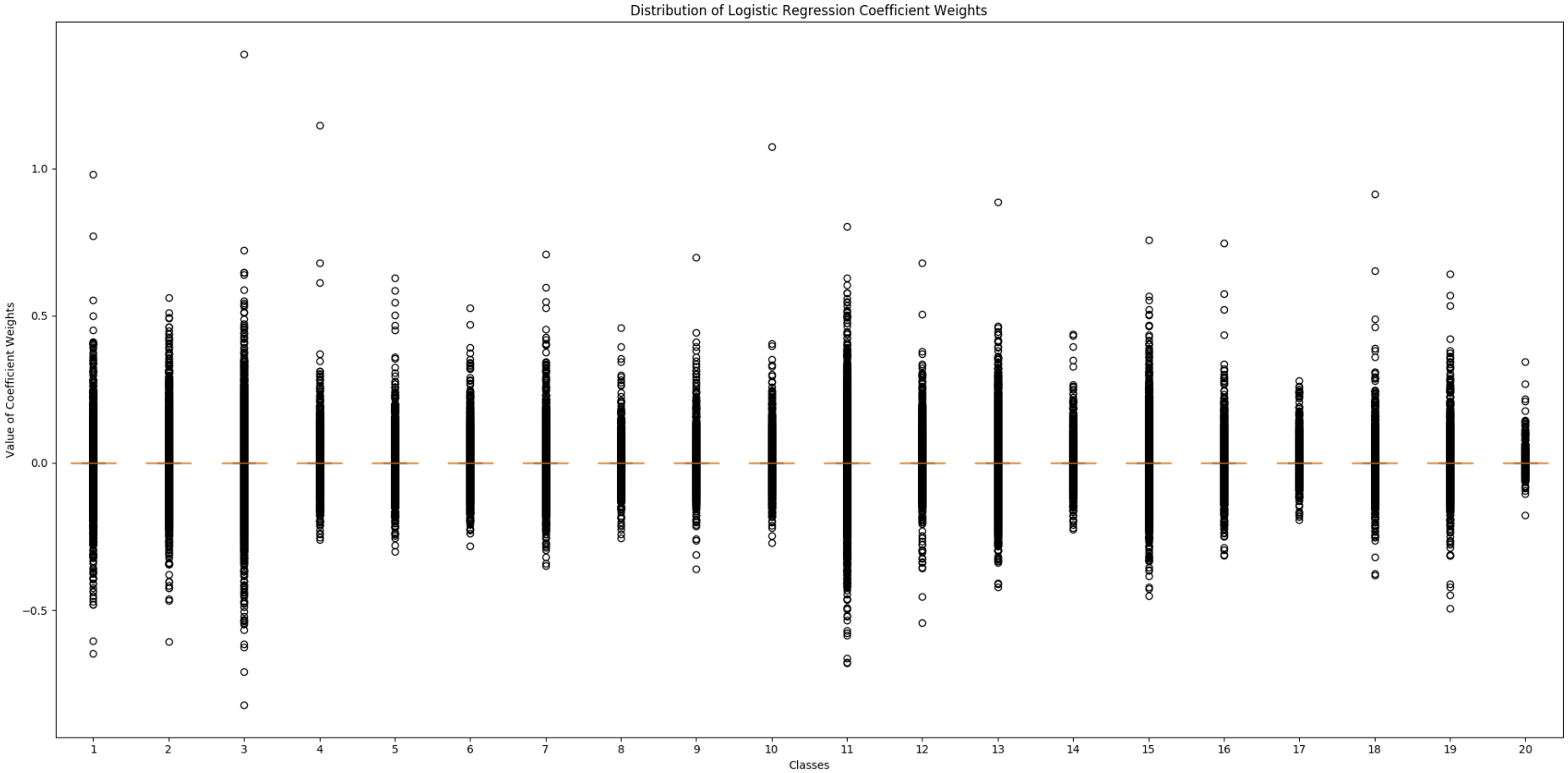


Figure 4.6: Visualisation of distribution of logistic regression coefficients, where the weight of a single token is a bubble on the plot. This shows the importance of each token in each class to the classifier. The classifier performs better in classes with longer top-ends and tail- ends. Heavily weighted tokens provide a strong indicator that a sample belongs to a certain class.

4.5.5 k-NN

	Count	TF-IDF
Accuracy/ Micro-F1	0.888	0.925
Macro-F1	0.829	0.885
k-neighbours	1	3
Distance weighting	-	Inverse

Table 4.10: Summary of k-NN performance. Surprisingly, with a count vectorizer, considering the sole nearest neighbour yields the best results.

As noted above, the k-NN classifier is a non-linear classifier based on Euclidean distance. We conducted a manual search for hyperparameter tuning to find the value for k producing the best results, as summarised above. Specifically, we ran the classifier for values of k from 1 to 15, and experimented with uniform and inverse distance weighting, the latter of which resulted in better performance. It should be noted that where only the single nearest neighbour is used (k=1), prioritising nearer neighbours with inverse distance weighting is not applicable, since there is only one neighbour being considered.

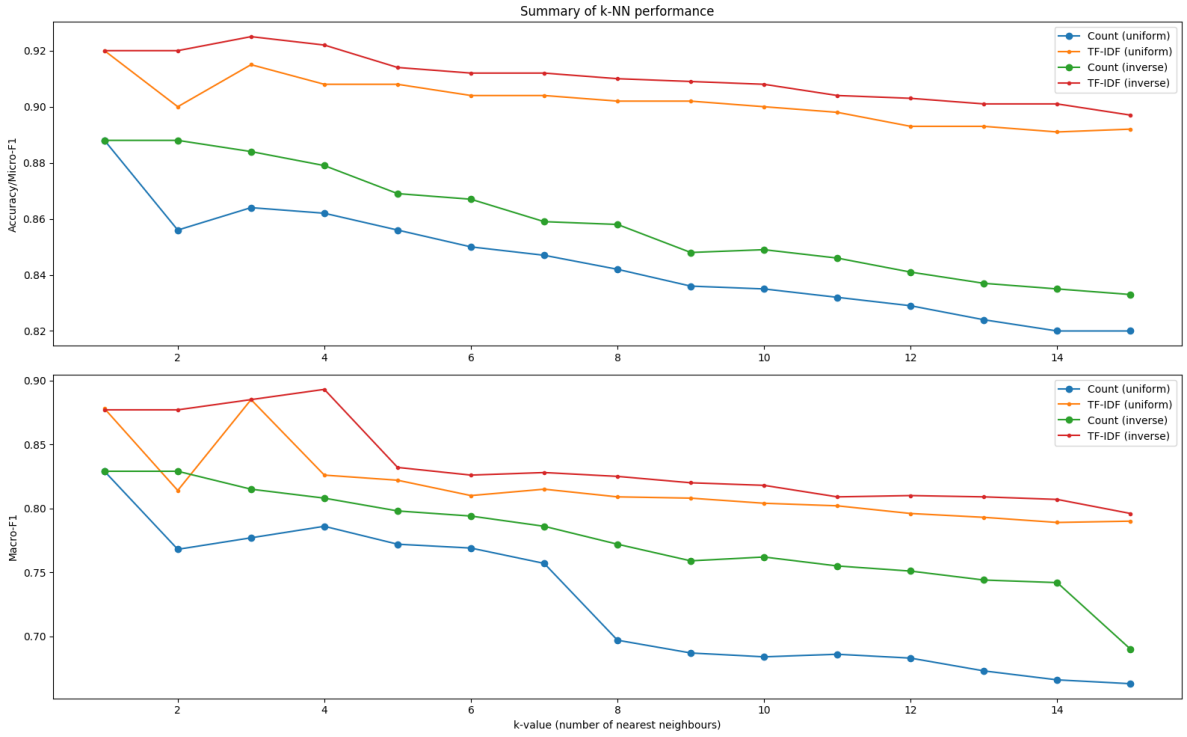


Figure 4.7: Performance of k-NN for k=1 to k=15

The figure above shows an interesting observation: that with a uniform distance weighting, with both the count and TF-IDF vectorizer, performance almost consistently decreases at a constant rate as k increases, with the exception of k=3. In both

cases, k=1, i.e. classifying an unseen sample based on the single nearest neighbour, yielded the best accuracy. This might tell us that many test and training samples belonging to the same class have similar counts of certain tokens, so the nearest neighbour is enough for a classification in many cases. The figure also shows that using inverse-weighted distances stabilises the performance of the classifier, so the decrease in performance is not as stark, but the general trend is still a decrease in performance as k increases.

Overall, TF-IDF vectorization with inverse distance weighting (in red in Figure 4.7) resulted in the best performance in all experiments. This is perhaps reflects the intuition that inverse distance weighting naturally works well with feature representation also based on inverse document frequency.

We provide a visualisation of the classifier for the best performing model, k=3 with TF-IDF vectorization and inverse distance weighting. This is only for the purposes of visualisation.[25] The visualisation is based on the decision boundaries learned on the training set. We note again that we employ a twofold dimensionality reduction method with LSA and t-SNE to provide a 2D visualisation, so these decision boundaries are very different from the classifier for which the performance is reported above, which has sparse vectors of dimension 223,842 (the vocabulary size of the training set) as its input.

Overall, the k-NN classifier with TF-IDF vectorization shows performance comparable to logistic regression, and considering the 3 nearest neighboura to a test sample is enough to result in accuracy of 92%.

[25] See *decision_boundary_visualisation.py*

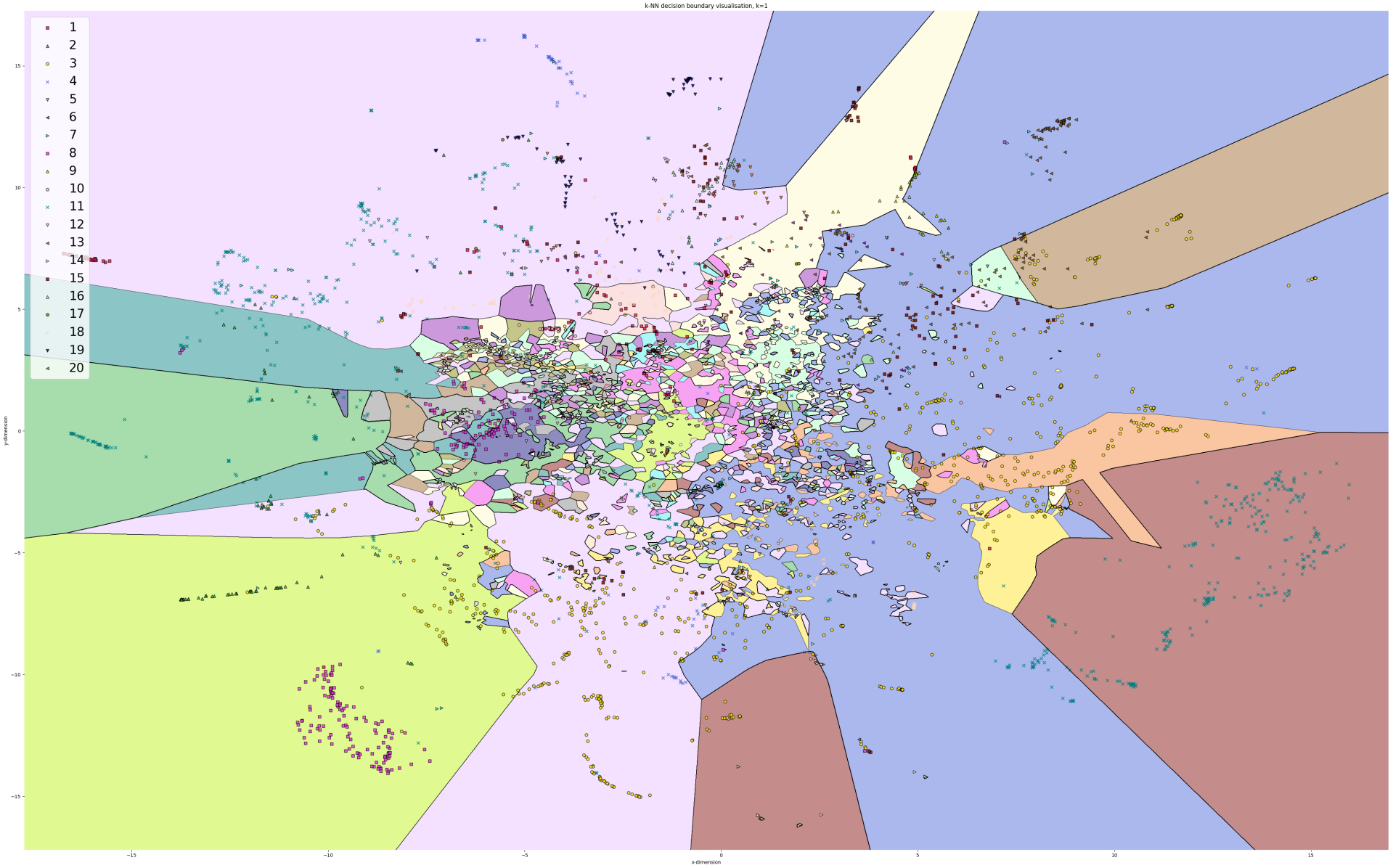


Figure 4.8: k-NN decision boundaries for the classifier where k=3 with inverse distance weighting. Each sample is a point on the visualisation and has a colour as per the legend, which indicates its class. The classifier labels all points within a colour boundary to the class associated with that colour.

4.5.6 Linear SVM

	Count	TF-IDF
Accuracy/Micro-F1	0.929	0.956
Macro-F1	0.880	0.930
Loss function	L2-loss	L2-loss
Max iterations	1,000	1,000
Classifier	One-vs-rest	One-vs-rest
Penalty	L2 regularization	L2 regularization

Table 4.11: Summary of linear SVM. A one-vs-rest SVM with TF-IDF vectorization is the best performing machine learning model.

From Table 4.1, it can be seen that the linear SVM shows one of the best performances across all of the machine learning classifiers, in terms of both accuracy and macro-F1. We perform a random search with the loss function and the classification strategy, with the best hyperparameters summarised above. In particular, the one-vs-rest classification strategy, which was the better performer, was explained above (in Section 2.1.5.5).

Our focus will be on the SVM classifier with TF-IDF vectorization, which was the best performing classifier in our random search. Given a high accuracy of 95.6% and Macro-F1 of 93.0%, we examine the performance of the classifier in each class more closely. We provide a complete breakdown of metrics per class, as well as confusion matrices.

First, from the normalised confusion matrix and Table 4.13, we note that the classifier has precision, recall, and F1 of 1.00 for Class 20. However, on closer examination, we can see from Table 4.13 that Class 20 has only *one* sample in the test set (and only 6 in the entire dataset). In general, we report the macro-average F1, as explained in Section 2.1.5.4, because it takes an average of the F1-score computed for each class separately, and our intuition is that this provides us with a fairer overview of the performance of the classifier, compared to a micro-average where the performance on the majority classes will dominate. However, in the case of Class 20, our reporting on the macro-F1 may have had the converse effect: classifying only one sample correctly in Class 20 pushes up the overall macro-average.

Secondly, we note that the classifier has the poorest performance in Class 17 from the confusion matrices. This is also the case with logistic regression, as per the normalised confusion matrix in Figure 4.5, and is also reflected by the fact that the coefficients for Class 17 have the smallest spread, as seen in Figure 2.4.4. However, we recognise again that Class 17 has the second-fewest number of samples, with only 13 samples in the test set out of a total of 64. In addition, the classifier has a precision of 1.000 for Class 17, and a recall of 0.692. From Equations 2.5 and 2.6, we know this means that each time the model identified a sample as Class 17, this was indeed the case (i.e. no False Positives). But the model has a much lower recall, with 4 False Negatives, thus it has missed 4 out of the 13 samples. This may be due to the subject matter

of the classes, for example, it classified one of the samples to Class 15, concerning 'Environment, consumers, and health protection', whereas one of the tokens with the greatest weights in Class 17 in logistic regression is 'protection'. Overall, the small sample size for Class 17 creates the opposite problem to that of Class 20: despite the fact that there were only 4 misclassifications, the average performance of the classifier is reduced greatly.

Thirdly, we can see that on the classes that form the majority of samples, Class 3 and Class 11, the model has an F1-score of 0.976 and 0.970 respectively. This does prop up the overall performance of the model, but is also a good sign that the model is in fact performing well on the test set.

Finally, we can compare the logistic regression confusion matrix (Figure 4.4) with the linear SVM confusion matrix (Figure 4.9). For Class 2, the SVM has 106 True Positives, only one less than logistic regression with 107. For Classes 5, 14, 16, 18 and 20, both classifiers have the exact same number of True Positives. In all other classes, the SVM outperforms logistic regression. Thus, apart from the one additional sample in Class 2 that the SVM misclassified, the SVM equals or outperforms logistic regression on all classes. From this we can perhaps state that the linear SVM performs better than logistic regression based on the test set.

	Linear SVM (correct)	Linear SVM (wrong)
Logistic regression (correct)	3001 ["A"]	91 ["B"]
Logistic regression (wrong)	23 ["C"]	119 ["D"]

Table 4.12: Contingency table of logistic regression and linear SVM

However, as with the decision tree and random forest, we should determine if the difference in performance between the two classifiers is statistically significant. Following the tests and the contingency table, we find that $\chi^2 = 39.377$, and $p = 3.494 * 10^{-10}$. Thus, $\chi^2 > 3.84$ and $p < 0.05$, and we *can* reject H_0. The difference in performance *is* statistically significant.

Overall, based solely on accuracy and macro-F1, the linear SVM is the best classifier of all considered so far. The performance of the linear SVM thus coheres with that of the seminal paper on text classification by Joachims [35], and also sits well with our observation so far that the linear models perform well on the test set.

Class	Metric			
	Precision	Recall	F1	Num. of samples (test set)
1	0.928	0.914	0.921	140
2	0.906	0.883	0.895	120
3	0.964	0.989	0.976	960
4	0.979	0.959	0.969	97
5	0.964	0.883	0.922	60
6	0.963	0.839	0.897	31
7	0.926	0.872	0.898	86
8	0.980	0.984	0.982	246
9	1.000	0.975	0.987	40
10	0.942	0.925	0.933	53
11	0.966	0.975	0.970	749
12	0.865	0.821	0.842	39
13	0.939	0.930	0.934	199
14	1.000	0.953	0.976	86
15	0.903	0.911	0.907	123
16	0.893	0.806	0.847	31
17	1.000	0.692	0.818	13
18	0.963	0.952	0.958	83
19	0.928	1.000	0.963	77
20	1.000	1.000	1.000	1
Micro-avg	0.956	0.956	0.956	3234
Macro-avg	0.950	0.913	0.930	(Total)

Table 4.13: Summary of linear SVM performance (TF-IDF vectorizer) per class

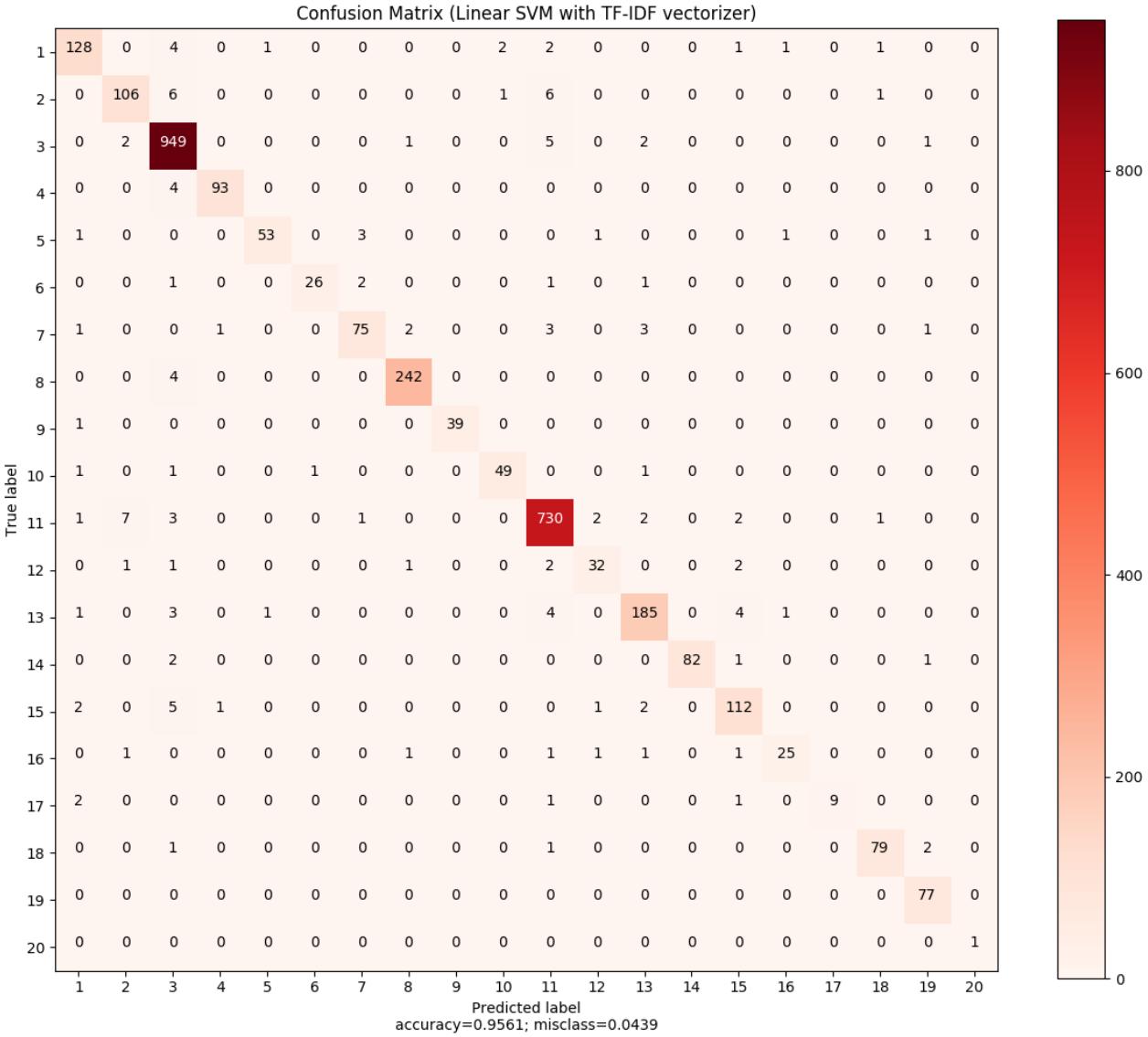


Figure 4.9: Linear SVM (TF-IDF vectorizer) confusion matrix

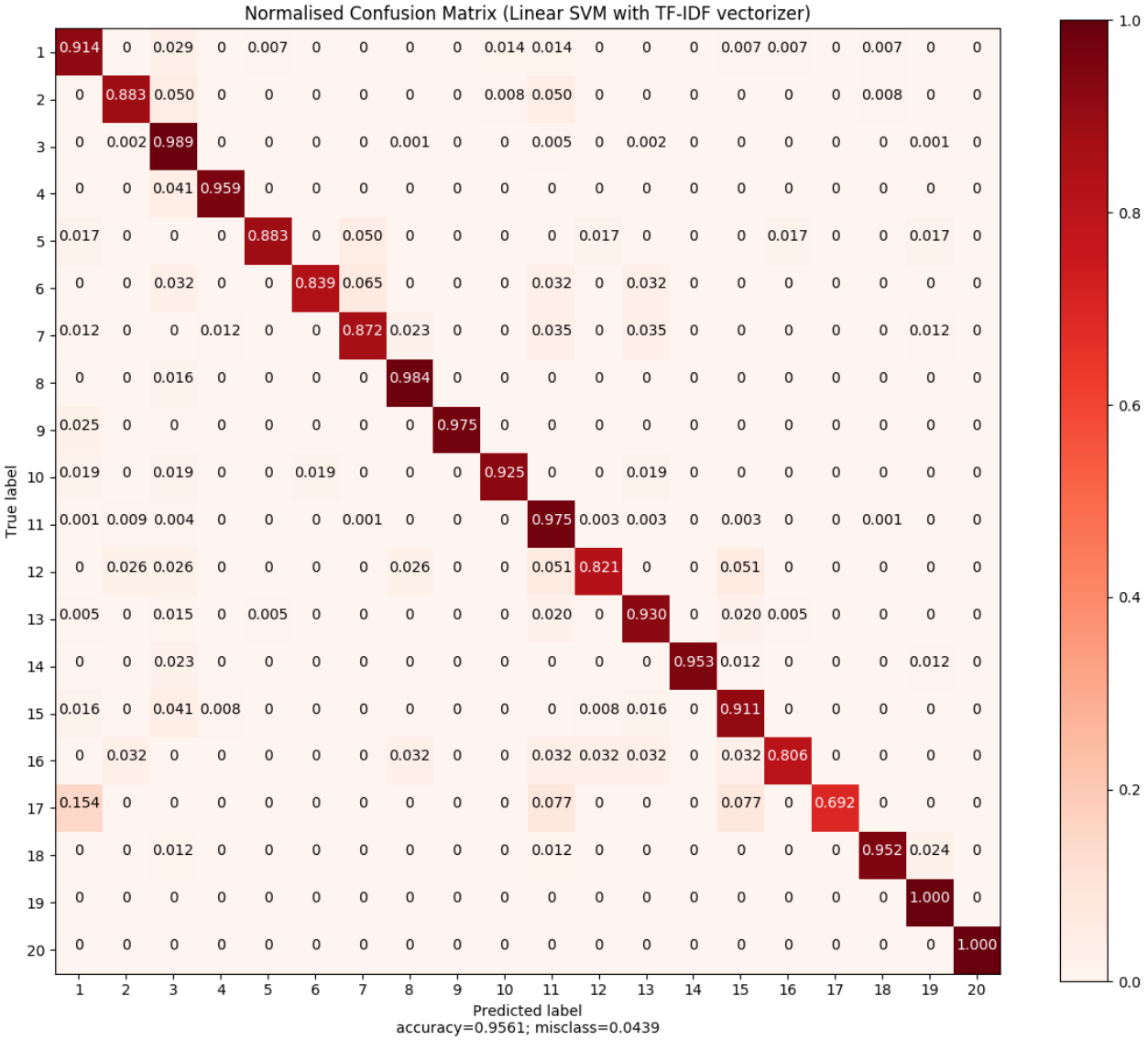


Figure 4.10: Linear SVM (TF-IDF vectorizer) normalised confusion matrix

4.5.7 Non-linear SVM

	Count	TF-IDF
Accuracy/ Micro-F1	0.712	0.945
Macro-F1	0.477	0.859
Kernel	RBF	Sigmoid
Kernel coeff.	'scale'	'scale'

Table 4.14: Summary of non-linear SVM

Next, we consider the performance of a non-linear SVM. We perform a random search to determine which non-linear kernel function (and which corresponding coefficient) works best. At the outset, comparing Tables 4.13 and 4.16, we note that the performance of the non-linear SVM with TF-IDF vectorization is similar to (although slightly worse than) that of the linear SVM, with an accuracy of 94.5%. We note that Macro-F1 is worse at 0.859, compared to 0.930, but this is largely due to slight differences in classifying a small number of samples in Classes 17 and 20, which have very small sample sizes, dragging down the macro average. We note that the non-linear SVM actually performs better on Class 8, with the same recall but slightly higher precision at 0.988 vs. 0.980. We may perhaps conclude from this the features of the samples in Class 8 would be more linearly separable if mapped into a higher dimensional space, but this does not work for the samples in other classes.

The similar performance is expected. Again, as noted above, the non-linear kernel works by mapping non-linearly separable features into a higher dimensional space where they can then be linearly separable. As Hsu et al. [30] argue, where the features in the dataset have many dimensions (as is the current scenario), mapping features to a higher dimensional space will not improve performance. However, it is not expected to *worsen* performance; the features which are linearly separable remain so, and are not mapped to a lower-dimensional feature space.

	Non-linear SVM (correct)	Non-linear SVM (wrong)
Linear SVM (correct)	3035	20
Linear SVM (wrong)	57	122

Table 4.15: Contingency table of linear and non-linear SVM

Again, we perform significance testing to compare the two models. We find that $\chi^2 = 16.831$, and $p = 4.086 * 10^{-5}$. As $\chi^2 > 3.84$ and $p < 0.05$, and we *can* reject H_0, and conclude that there is in fact a statistically significant difference between the performance of the linear and non-linear SVM.

It should be noted that the non-linear SVM performs much worse when feature representation is achieved through the count vectorizer. Since features which are linearly separable should simply remain so, the reason for the inconsistency is unclear.

Class	Metric			
	Precision	**Recall**	**F1**	**Num. of samples (test set)**
1	0.864	0.907	0.885	140
2	0.935	0.833	0.881	120
3	0.959	0.988	0.973	960
4	0.978	0.918	0.947	97
5	0.964	0.883	0.922	60
6	0.923	0.774	0.842	31
7	0.902	0.860	0.881	86
8	0.988	0.984	0.986	246
9	1.000	0.950	0.974	40
10	0.922	0.887	0.904	53
11	0.944	0.972	0.958	749
12	0.861	0.795	0.827	39
13	0.902	0.930	0.916	199
14	0.988	0.942	0.964	86
15	0.912	0.837	0.873	123
16	0.880	0.710	0.786	31
17	0.889	0.615	0.727	13
18	0.988	0.952	0.969	83
19	0.927	0.987	0.956	77
20	0.000	0.000	0.000	1
Micro-avg	0.944	0.945	0.944	3234
Macro-avg	0.886	0.836	0.859	(Total)

Table 4.16: Summary of non-linear SVM performance (TF-IDF vectorizer) per class

4.5.8 MLP

	Count	TF-IDF
Accuracy/ Micro-F1	0.953	0.953
Macro-F1	0.921	0.920
No. hidden layers	3	3
Neurons per layer	128	128
Activation function	ReLU	ReLU
Optimizer	Adam	Adam
Regularization	L2	L2

Table 4.17: Summary of MLP, which performs comparably to or better than all other machine learning models.

We perform a grid search for hyperparameter tuning within a manually defined search space; we only vary the number of hidden layers (from 1 to 3) and the number of neurons (64, 128, 256, 512) in each hidden layer. We use reLU activation functions across all hidden layers and the popular 'Adam' optimiser with L2 regularization. The general trend is that performance increases as the number of layers and number of neurons increases, but only up to 128 neurons per layer, with 3 hidden layers. We can also see that the network has almost no performance difference with count or TF-IDF vectorization for feature representation.

From the overall summary in Table 4.1, it can be observed that the MLP is the best performing model of all models on the test set when used with the count vectorizer, and is the second-best performing model when used with the TF-IDF vectorizer. Again, we run significance tests on these results:

Count vectorizer		
	MLP (correct)	**MLP (wrong)**
Logistic regression (correct)	2976	94
Logistic regression (wrong)	48	116

TF-IDF vectorizer		
	MLP (correct)	**MLP (wrong)**
Linear SVM (correct)	3043	42
Linear SVM (wrong)	49	100

Table 4.18: Contingency tables for significance tests of MLP against other models

Compared to logistic regression, we obtain the values $\chi^2 = 14.261$, and $p = 1.592 * 10^{-4}$, thus $\chi^2 > 3.84$ and $p < 0.05$. Thus, we *can* reject H_0 and conclude that the MLP does outperform logistic regression. Compared to the non-linear SVM, $\chi^2 = 0.396$, and $p = 0.529$, thus $\chi^2 < 3.84$ and $p > 0.05$, and we *cannot* reject H_0, and we cannot conclude that the linear SVM outperforms the MLP.

From this, we may argue that the MLP performs better than or comparably to all other models we have considered on the test set, with both the count and TF-IDF vectorizers for feature extraction. Thus, the MLP results confirm the effectiveness of the feedforward artificial neural network. This provides us with a good starting point from which we can determine if more complex neural network architectures will result in a better performance, as we will do in the next section.

4.5.9 Preliminary conclusions

At this stage, we can conclude that where features are extracted from samples with the count vectorizer, the samples in the test set seem to be linearly separable to a large degree. Linear models (logistic regression and linear SVM) outperform non-linear models (Naive Bayes, decision tree, random forest, k-NN, and SVM with a non-linear kernel), with the exception of the MLP. This relationship is less clear when the TF-IDF vectorizer is used, although the linear SVM and MLP still perform very well.

With this in mind, we can now proceed to consider whether using word embeddings and deep learning models will result in better performance than the models evaluated above.

4.6 Deep learning models

The implementation in this section is also done in Python with the Keras library,[26] using a Tensorflow backend.[27] For consistency, the preprocessing steps (stopword and punctuation removal and lowercasing) remain the same. The exception is in the case of sentence embeddings, where we first lowercase the text samples, without removing punctuation. The punctuation marks allow the spaCy sentencizer to determine the points on which to split each text sample into sentences; once this is complete, the punctuation marks are removed.

For pre-trained word embeddings, we use the spaCy library, which returns a vector of dimension 300 for each word.[28] Sentence embeddings are also from spaCy; to obtain a sentence embedding, the individual word embeddings of each word in a sentence are summed, and then divided by the number of words in that sentence. We use pre-trained embeddings from spaCy, as opposed to other popular options such as GloVe [57], word2vec [49] or fastText[29], because these have been extensively used in NLP research, thus it was thought that experimenting and reporting performance with spaCy would provide more of an academic contribution.

[26] For documentation, see *https://keras.io/*
[27] See *deep_learning_models.py*
[28] For documentation, see *https://spacy.io*
[29] See *https://fasttext.cc/*

To show how spaCy represents words in a defined vector space, we provide a visualisation of 25 commonly occuring words in the dataset in Figure 4.11. We obtain the 300-dimension embeddings from spaCy, and reduce the dimensions to 2D using t-SNE. Colours have been added to different tokens in the plot for the purposes of clarity. We can clearly see that words having greater semantic similarity are embedded closer to each other, despite the fact that the dimensions have been reduced from 300D to 2D.

Regrettably, we were unable to work with the word embeddings for the full samples in the dataset due to hardware constraints. This is because while there was an average of 1,441 tokens per sample in the dataset across all classes, as shown in Table 3.4, the longest sample had 281,204 tokens. In most cases each sample to be passed as input into the neural network has to have the same dimensions, thus, samples of different dimensions have to be *padded* (i.e. zeros are appended to those samples) for the dimensions to match. Padding all the samples to the length of the longest sample would result in dimensions of (281204, 300) per sample, which was not possible with the hardware available.

Thus, we first ran our experiments by shortening each sample to a maximum of 300 tokens, then obtaining the word embeddings for those tokens, and padding the samples shorter than 300 tokens. Thus, each sample was of dimensions (300, 300). This approach already required approximately 45GB to be loaded into memory. For sentence embeddings, where each sentence is represented with a 300-dimension vector, we again limited the maximum length of each sample to 300 sentences and padded where necessary, thus each sample was also of dimensions (300, 300). Lastly, we managed to increase the number of tokens per sample to 1,000, thus each sample was of dimensions (1000, 300).

We were fully aware that this approach would drastically affect the performance of the models tested, and would result in an unfair comparison between the performance of word embeddings and count-based feature representation, because each model would only have access to a limited portion of each sample. However, the silver lining to this is that we ultimately report that increasing the number of tokens per sample from 300 to 1,000 does not actually improve performance.

In addition, due to time constraints and the longer running time of more complex models, we perform hyperparameter tuning in this section with a manual search, as opposed to a random search or grid search. We use reLU layers for the MLP and CNNs (the LSTMs and HAN have sigmoid and tanh layers as part of their architecture), and in all models we use a softmax output layer for our task as expected. We run all models over 50 epochs with early stopping with a patience of 5 epochs, i.e. if the validation loss does not decrease after 5 epochs, training of the model stops. As before, we report accuracy and macro-F1.

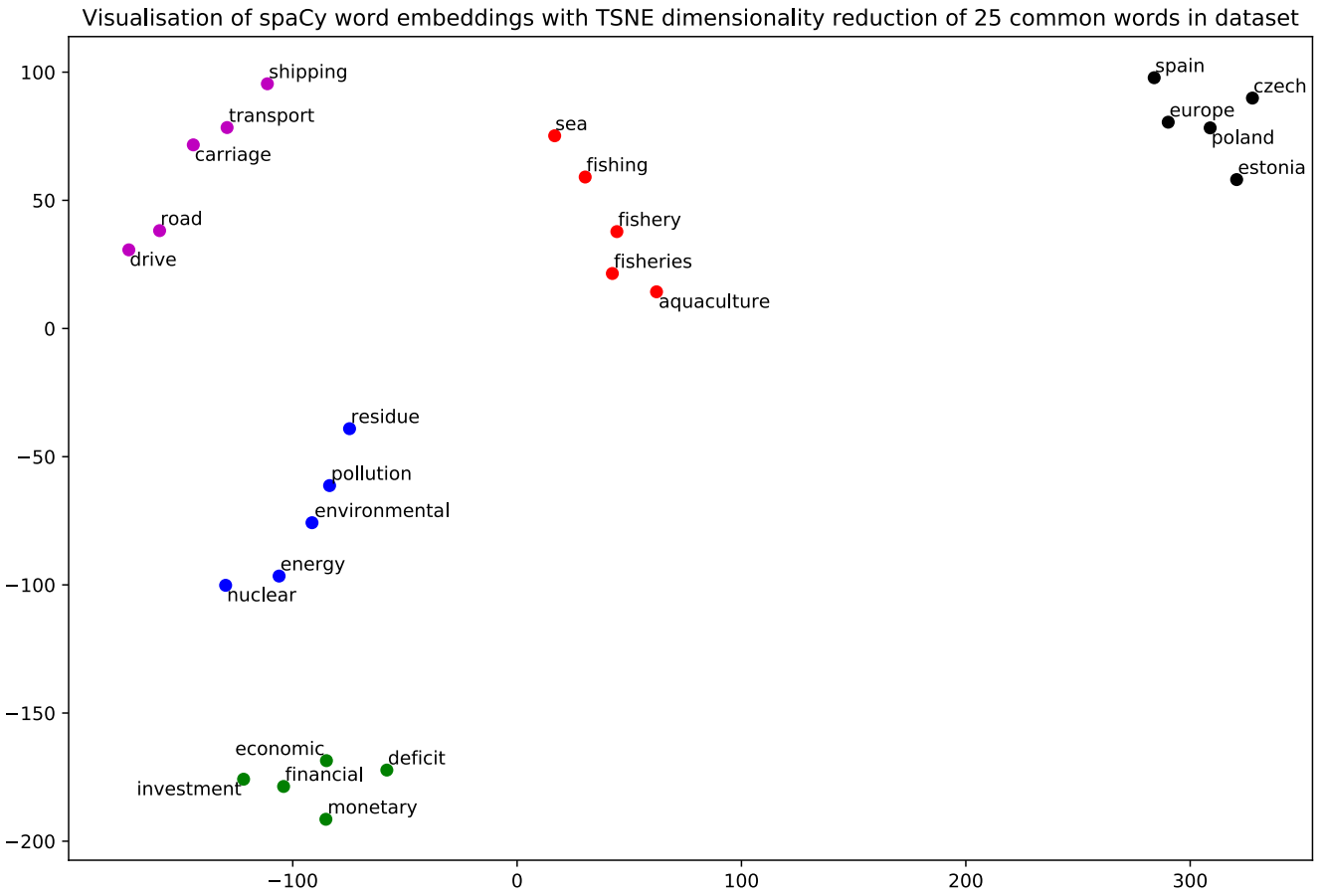


Figure 4.11: Visualisation of spaCy word embeddings for 25 common words in the dataset. 300-dimension embeddings have been reduced to 2D by t-SNE dimensionality reduction. Words with greater semantic similarity are embedded close together, as shown in the clusters.

4.7 Summary of results

Classifier	Accuracy/Micro-F1			Macro-F1		
Embeddings	**Word embeddings (300)**	**Word embeddings (1000)**	**Sentence embeddings (300)**	**Word embeddings (300)**	**Word embeddings (1000)**	**Sentence embeddings (300)**
MLP	0.793	0.728	0.459	0.595	0.508	0.215
CNN	**0.920**	**0.915**	**0.557**	**0.791**	**0.791**	0.300
CNN (n-gram)	0.900	0.870	0.552	0.740	0.684	0.291
RNN (LSTM)	0.882	0.868	0.512	0.718	0.673	0.274
RNN (bi-LSTM)	0.899	0.872	0.535	0.747	0.689	**0.307**
HAN	0.908			0.746		

Table 4.19: Summary of deep learning models. The CNN is the best performing model with pre-trained embeddings in almost all cases.

4.7.1 General trend

The table above perhaps shows an interesting observation: that performance of all the deep learning classifiers, in terms of both accuracy and macro-F1, does not increase as the length of the text passed as input increased from 300 to 1,000 tokens (excluding the HAN, which does not use pre-trained embeddings). In fact, providing the network with more information slightly worsens performance. From this we can perhaps make a very broad conjecture: that a lot of the key information in our texts which is indicative of the class to which a sample belongs is contained in the first 300 tokens of each sample, and that the next 700 tokens in fact contain some superfluous information which dampens performance. If this were true, then the approach we took to shorten samples due to hardware limitations may not be that drastic after all.

4.7.2 Sentence embeddings

At the outset, we note that the performance of all classifiers when used with sentence embeddings is particularly poor. In order to understand this, we examine the workings of the spaCy sentencizer and how the sentence embeddings are obtained more closely.

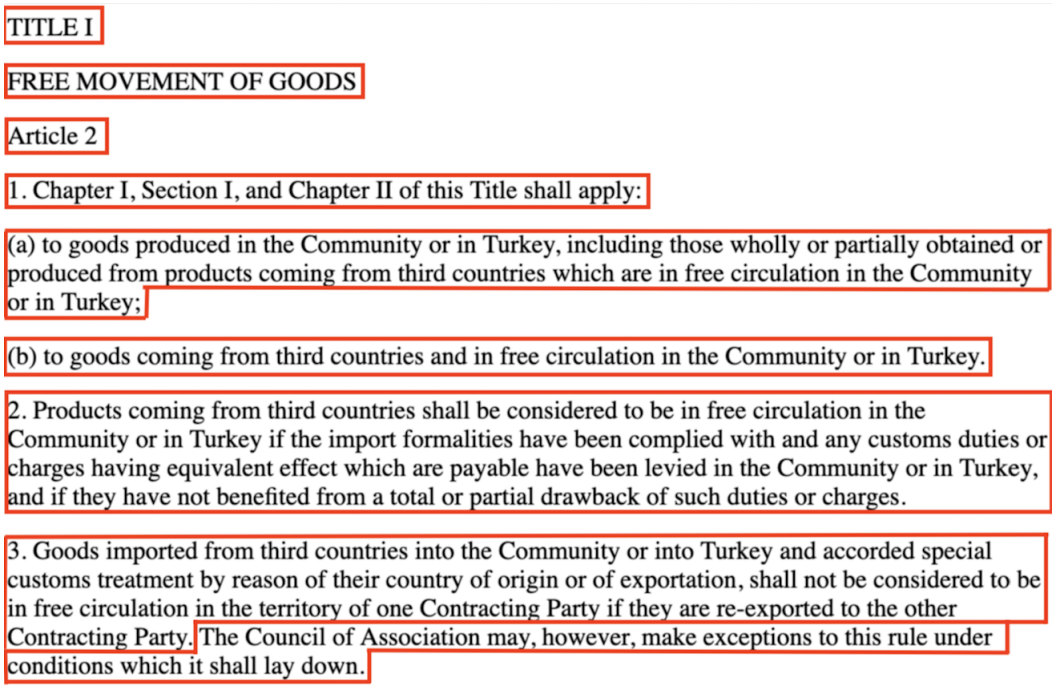


Figure 4.12: Splitting of a sample into sentences by the spaCy sentencizer

The figure above shows a section of a specific text sample from the dataset. Here we show a raw text sample only to show how the sentencizer works, although in the model the text would have been preprocessed and lowercased. The text sample is passed into the sentencizer, which then splits the sample into individual sentences based on punctuation and paragraphing. Each individual sentence, as determined by the sentencizer, has been boxed in red. From this, we can observe that lines which may ordinarily not be viewed as complete sentences, such as headings like 'TITLE I', 'FREE MOVEMENT OF GOODS', and 'Article 2', are regarded as complete sentences. Similarly, the preamble to certain paragraphs, such as "Chapter I...shall apply:" is regarded as a sentence on its own, as are individual clauses (such as (a) and (b) in the example given).

We may not ordinarily regard these as complete sentences; indeed, the question of what should constitute a proper sentence for the purposes of sentencization is itself a matter for debate. The salient point to note is that a sentence embedding is the average of the word embeddings of each token in a sentence. Thus, a single 300-dimension vector is used to represent the features in 'TITLE I', as well as that of a much longer sentence such as the whole of clause 2 in the example above. Intuitively, this would not lead to reliable results, and we can surmise that this is the reason for the uniformly poor performance of deep learning models with sentence embeddings. It would have been possible to experiment with a different approach to sentence embeddings, or a trainable embedding layer, but this has not been attempted due to time constraints.

However, the poor performance with sentence embeddings also led us to experiment with a HAN for our final model, which obtains vector representations of both words and sentences using the word encoder and sentence encoder, as introduced in Section

2.4.10 and as discussed below.

4.8 Analysis of results

4.8.1 MLP

	Word embeddings (300)	Word embeddings (1000)	Sentence embeddings (300)
Accuracy/ Micro-F1	0.793	0.728	0.459
Macro-F1	0.595	0.508	0.215
No. hidden layers	3	3	3
Neurons per layer	128	128	128

Table 4.20: MLP does not perform as well as before with word embeddings

The first classifier we consider is the MLP, which, as we noted above performs better than or comparably to all other machine learning classifiers considered. However, the basic feedforward network does not perform well in our experimentation with word embeddings. We can surmise that it was easier for the network to learn relationships between the sparse feature representations of a count-based vectorizer, as compared to word embeddings. With this in mind, we proceeded to consider more complex models.

4.8.2 CNN

	Word embeddings (300)		Word embeddings (1000)		Sentence embeddings (300)	
	CNN	**CNN (n-gram)**	**CNN**	**CNN (n-gram)**	**CNN**	**CNN (n-gram)**
Accuracy/ Micro-F1	0.920	0.900	0.915	0.870	0.557	0.552
Macro-F1	0.791	0.740	0.791	0.684	0.300	0.291
No. conv. layers	2	3	2	3	2	3
Filters per layer	256, 512	100	256, 512	100	256, 512	100
Filter sizes	3	3, 4, 5	3	3, 4, 5	3	3, 4, 5
Stride length	1	1	1	1	1	1

Table 4.21: Summary of CNN architecture. The n-gram approach was not as effective on our dataset.

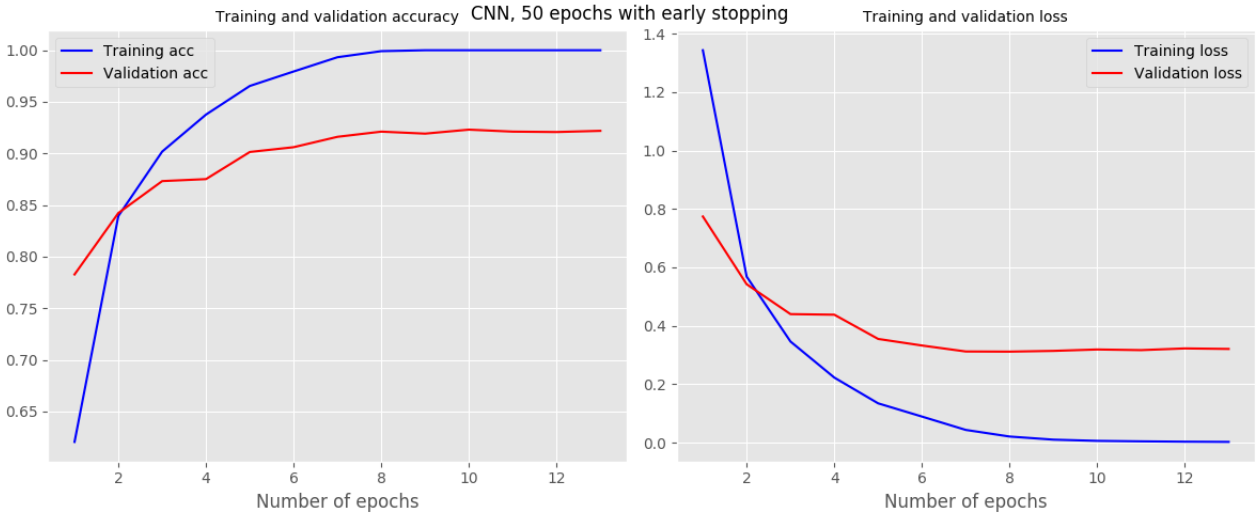


Figure 4.13: Training and validation accuracy and loss over epochs for the best performing CNN model, with word embeddings for the first 300 words.

In general, we experimented with CNNs with a convolutional layer with a max pooling layer, and another convolutional layer, with global max pooling. We tuned hyperparameters within this setup, including varying the number of filters and the size of the filter window, and report the most promising results. We do not change the stride length, as it was our understanding that a stride length of 1 was a sensible one in our context, i.e. the windows slide over each token in the sample, one token at a time. Ultimately, the CNN the best performance of all the deep learning models experimented with in almost all cases.

As explained above, we note that to ensure that the inputs are of equal length, the samples which are of a shorter length than the maximum length chosen (e.g. 300 words per sample) have to be padded to that length with zeros. Ideally, when running the network, we would add a masking layer which allows the network to ignore these padded zeros during training. Unfortunately, masking with 1D convolutional layers is not supported by the Keras API, and due to time constraints and technical difficulty, we did not implement masking with the convolutional layers. However, we can state that we were able to use a masking layer with the LSTMs, and this did not improve performance significantly.

In addition, we experiment with a CNN of the architecture used by Kim [37], a highly influential approach in the text classification literature which showed promising results when published. As noted in Section 2.4.8, the idea is to use convolution layers with filters of several sizes to capture information relating to different n-grams. The original model pairs convolutional layers with a 1D max pooling layer, and concatenates these layers. Thus, each convolutional layer has 100 filters with a certain window size (3, 4 or 5 for tri-grams, 4-grams and 5-grams), and is then connected a 1D max pooling layer, which is connected to the next convolutional layer, and so on. We experimented with this and different filter window sizes of 2, 3 and 4, but the original model showed a better performance. The original model also used a fully connected layer and dropout

between layers. We experimented with removing these layers, but this did not improve accuracy.

Finally, we note that the original model by Kim was used for sentence classification, where capturing tri-grams, 4-grams or 5-grams makes more sense. For example, if there is a specific tri-gram, 4-gram, or even 5-gram within a sample of a single sentence, that information is perhaps a good indicator of the class to which that sentence belongs. On the other hand, our task is document classification, and the documents can be of great length, so the presence of certain n-grams may not be that strong an indicator that the sample belongs to a certain class. The same n-grams may appear across many documents. We can surmise that due to the length of our documents, the CNN with layers of 256 and 512 filters performed slightly better.

4.8.3 LSTM

	Word embeddings (300)		**Word embeddings (1000)**		**Sentence embeddings (300)**	
	LSTM	**Bi-LSTM**	**LSTM**	**Bi-LSTM**	**LSTM**	**Bi-LSTM**
Accuracy/ Micro-F1	0.882	0.899	0.868	0.872	0.512	0.535
Macro-F1	0.718	0.747	0.673	0.689	0.274	0.307

Table 4.22: Summary of LSTM performance. Using a bidirectional LSTM improves performance to some extent.

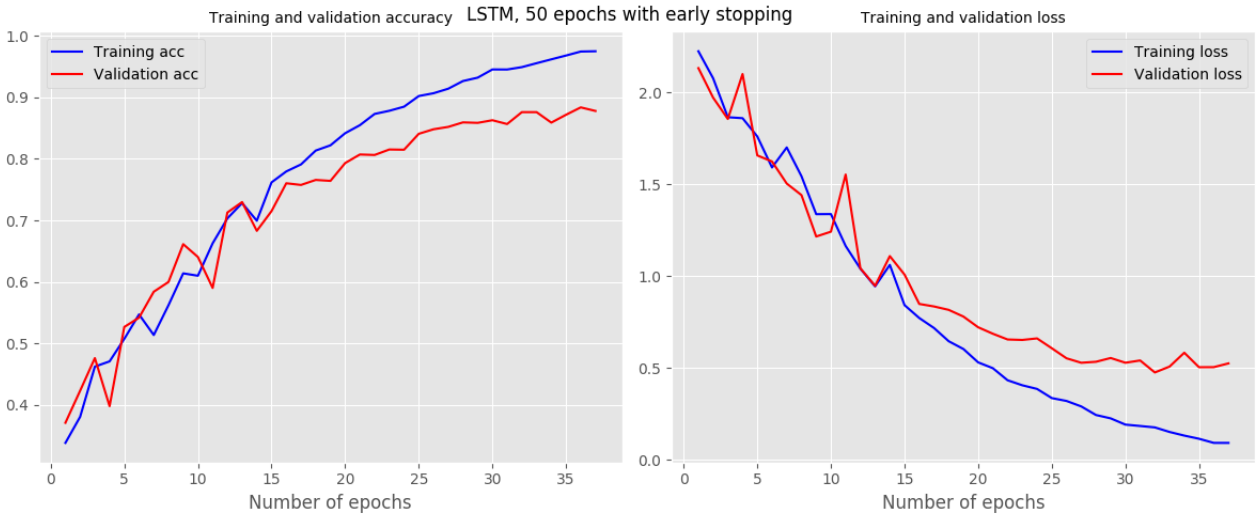


Figure 4.14: Training and validation accuracy and loss over epochs for the single layer LSTM, with word embeddings for the first 300 words.

As noted above, we tried using LSTMs because other networks may not have persistent memory, and given that our documents can be particularly long, as shown in Table 3.3, a vanilla RNN may not be able to learn long-term dependencies [26]. We thus experimented first with a single layer LSTM, varying the number of units for hyper-parameter tuning, from 128, 256 and 512. In all cases, the 256 units produced the best results. We attempted to stack LSTMs (multiple layers in the same direction), but this led to poor performance, thus we ultimately report performance on a single layer. We then experimented with adding a layer with inputs read in from the opposite direction, as explained above, a bi-LSTM. The intuition is that this should allow the LSTM's cell state to preserve not only information in the past (with reference to a certain point in time), but also in the future. This led to a slight improvement in performance.

4.8.4 HAN

	Hierarchical Attention Network (HAN)
Accuracy/ Micro-F1	0.908
Macro-F1	0.746
Max sentences	300
Max tokens per sentence	300

Table 4.23: Summary of our HAN set-up, which shows a good performance but does not outperform a CNN.

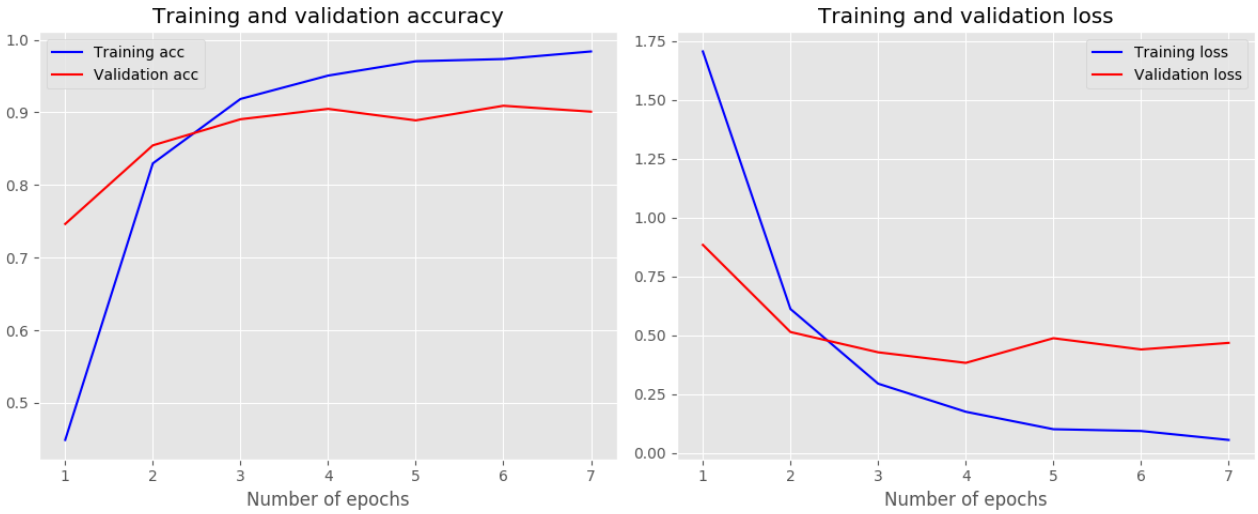


Figure 4.15: Training and validation accuracy and loss over epochs for the HAN.

As shown in Figure 2.4, the HAN has four layers (excluding the softmax output layer): a word encoder, word attention layer, sentence encoder, and sentence attention layer.

The point of the encoding layers is for it to capture the most important information from the text input in the form of word embeddings. Thus, the approach taken with the HAN model is slightly different to that of the other deep learning models discussed, in that we do not use spaCy's pre-trained word embeddings. Instead there is a trainable embedding layer. This is a layer which takes in an input of the dimensions of the number of unique tokens in the training set, and the desired size of each word embedding. Thus, each token in the training set will have a word embedding vector. These vectors get updated during the process of training the network, when the information from the encoding and attention layers gets propagated backwards.

As far as possible we maintain a similar setup: we limit the number of sentences in a sample to a maximum of 300, and the number of tokens in a sentence to 300.[30] The embedding layer returns a 300-dimension embedding for each unique token. We use the same spaCy tokenizer to tokenize samples. We then initialise a Keras tokenizer which gives each unique token in the training set an integer ID, much like the count vectorizer we used with the machine learning models. We then shorten the samples to our maximum lengths due to computational limitations. We then pass each vectorized sample, which is vectorized based on the tokenizer's integer IDs, into the first layer of the model, the embedding layer.

Overall, we note that the HAN performed well, but was not able to display better results than the CNN. Its accuracy justifies to some extent the intuition behind using a trainable embedding layer and attention layers to focus on key information in each sample. It also lends weight to the notion that documents have a hierarchical structure, being formed from words and sentences. However, compared to the CNN, the HAN took a much longer time to train, over 15 hours for each epoch with our hardware. Thus the CNN is not only more accurate, but also more efficient in our case. A key limitation of our work is that we have not analysed time complexity and training time, as we will discuss in Chapter 6.

4.8.5 Conclusions on deep learning models

It is not surprising that all the deep learning models we implemented outperformed the MLP by far. However, it was interesting for us to note that a CNN with fixed-size windows running uni-directionally outperformed many other models based on other models based on a more nuanced understanding of language, such as its hierarchical constructs and the idea that the meaning of a word should be captured not only from the words preceding it, but also those succeeding it. Our view is that this is probably due to the nature of the textual data specifically in this dataset. As we noted with our analysis of the logistic regression classifier, there are many single words occurring in samples which are strongly indicative of the true label of a sample which would not occur in the context of any of the other classes (such as 'fishing' and Class 4 (Fisheries)). This led to a fixed-size window approach with a stride length of 1 being effective. In addition, the max pooling and global max pooling layers focus the

[30] This model is implemented in *han.py*, not *deep_learning_models.py*

attention of the network on those single words which are particularly useful for the classification task. On the other hand, the effectiveness of some of the more complex models may have been dampened where the models ended up capturing superfluous information unrelated to the classification task, especially given the complex linguistic features of the samples. In effect, in our particular case, Occam's razor holds: "the simplest solution is often the best one".

4.9 Choice of best models

	Accuracy/Micro-F1	Macro-F1
Linear SVM	0.956	0.930
CNN	0.920	0.791

Table 4.24: Best performing models

Overall, due to the nature of our experimental set-up, we have to comment on the machine learning models and deep learning models separately before recommending a single model. This is largely because we had to limit the length of the samples used with the deep learning models, resulting in any comparison being a potentially unfair one.

Throughout this project, we have considered *performance* in terms of accuracy and macro-F1. However, in our conclusion and with regards to making a recommendation for which such a model can be used in practice, we have to consider the efficiency of training time relative to performance and the complexity of setting up models as well.

We consider the best machine learning model to be feature representation with a TF-IDF vectorizer and classification with a one-vs-rest linear SVM, having the best accuracy and macro-F1 overall. In terms of macro-F1, it significantly outperforms all deep learning models. It also has a faster training time than the MLP, which has comparable accuracy and macro-F1.

We consider the best deep learning model to be the CNN. The next best performing model is the HAN, but the running time of the HAN is over 15 hours for a single epoch on our limited hardware. In contrast, the CNN can be run over 10 epochs in several minutes (although this does not include the time taken to obtain pre-trained embeddings from spaCy, which we store and access in a dataframe). Regrettably, as we have noted, we did not monitor closely the running time of each model during our implementation, and are not able to report an exact runtime.

Between the two models, due to its simplicity in implementation, swift running time as well as higher accuracy and macro-F1 overall, on this particular dataset, we would choose the linear SVM.

Chapter 5

Related Work

5.1 Deep learning approaches to NLP

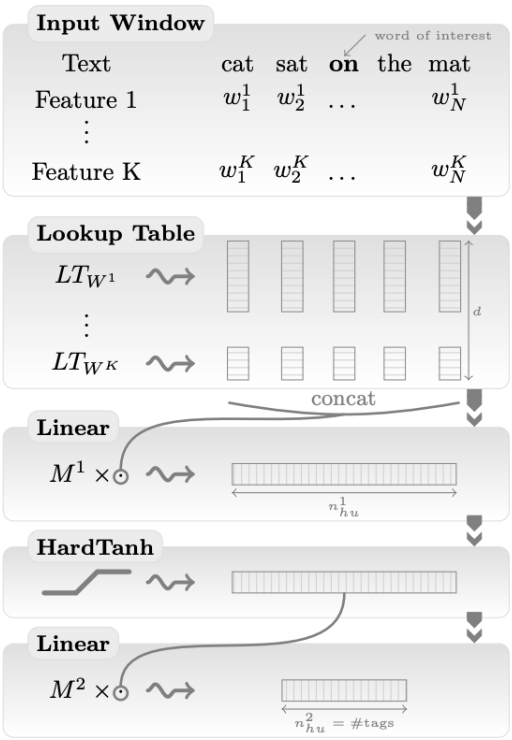


Figure 5.1: Collobert and Weston's CNN architecture (Figure 1, [11])

In this section, instead of reviewing machine learning research which may be less recent, we focus more on contemporary deep learning research (as per our distinction between both groups). In a seminal article published over a decade ago, Collobert and Weston [10] demonstrated that a single convolutional neural network (CNN) model, trained jointly on six standard NLP tasks (e.g. part-of-speech (POS) tagging, chunking, named entity recognition (NER), semantic role labelling(SRL)), could outperform systems built with hand-engineered features. This paper has proven to be highly influential in motivating the use of neural networks in NLP. This paper is particularly

useful as it explains the traditional approach to NLP (feeding hand-engineered features into a classification algorithm e.g. a support vector machine (SVM)), and then details how a neural network is built. It also details its use of word embeddings, which was considered state-of-the-art at the time. They followed this up with [11], and the figure of their CNN architecture is reproduced above. These papers catalysed the use of CNNs in NLP.

Goldberg [20] provides a useful primer on various neural network models for NLP, which we have relied on in our report. More recently, a comprehensive and very useful review of recent trends in deep learning-based approaches to NLP has been done by Young et al. [77]. In particular, it highlights research on RNNs and CNNs, which we have relied on to inform our understanding of the subject matter. In addition, they provide an overview of the success of pre-trained models such as BERT [13] and ELMo [58] for text classification, which we were unfortunately unable to experiment with due to time constraints.

	Source	Nearest Neighbors
GloVe	play	playing, game, games, played, players, plays, player, Play, football, multiplayer
biLM	Chico Ruiz made a spectacular play on Alusik 's grounder {...}	Kieffer , the only junior in the group , was commended for his ability to hit in the clutch , as well as his all-round excellent play .
	Olivia De Havilland signed to do a Broadway play for Garson {...}	{...} they were actors who had been handed fat roles in a successful play , and had talent enough to fill the roles competently , with nice understatement .

Figure 5.2: Effect of contextual representations, which captures different meanings of the word "play" (Table 4, [58])

ELMo [58] builds on the idea of pre-trained word representations from models such as word2vec [49], but take this further with *deep contextualised* word representations. Broadly, the key difference between ELMo and pre-trained word embeddings like word2vec is that instead of assigning an embedding to each token based on the vocabulary of the entire dataset, each token is assigned a representation based on an input sentence. They use a bi-LSTM trained with a language model (biLM) objective to obtain these representations. They show that the higher-level LSTM captures context, while the lower-level LSTM captures syntax, leading to state-of-the-art performance in several NLP tasks. The figure above provides an example of their approach, showing how the meaning of the word "play" is captured in context. While Peters et al. [58] do not consider multi-class text classification in their original paper, they provide results on sentiment analysis, and their model is designed to be used with existing architectures.

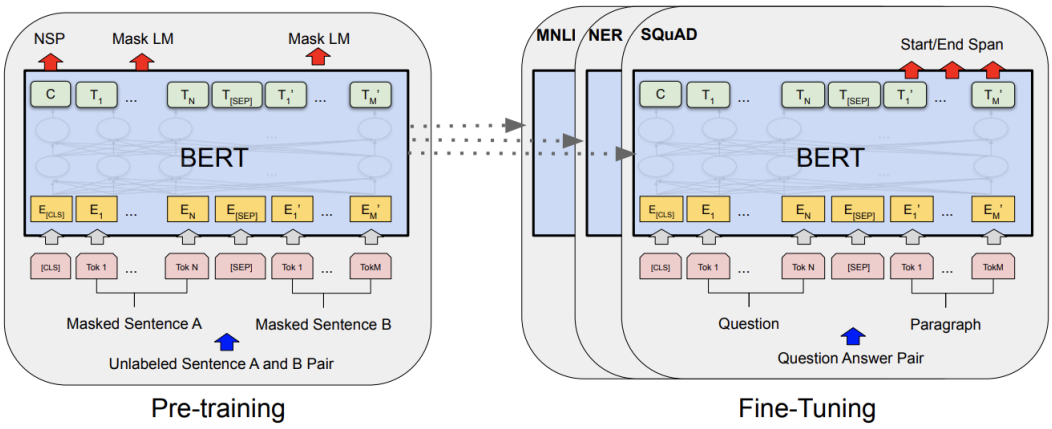


Figure 5.3: BERT's two stages. It is first trained on unlabelled data in several pre-training tasks, then fine-tuned using labelled data from a downstream task; in this example the task is question answering (Figure 1, [13]).

BERT [13] is based on an idea alluded to earlier: that language representations should be bi-directional. While ELMo uses task-specific architectures, it uses uni-directional language models to learn representations. BERT aims to overcome this constraint using the 'masked language model'. It has two steps, shown above: pre-training and fine-tuning. In the first stage, it is trained on unlabelled data, with different pre-training tasks. In the latter stage, it is initialised with the parameters obtained from pre-training, and these parameters are then fine-tuned, using labelled data from a downstream task. Thus, each downstream task will have a separate fine-tuned model. However, there is little difference between the pre-trained architecture and the final downstream architecture. A summary of both steps is shown in their figure reproduced above. Overall, the major contribution of BERT is that it applies unsupervised pre-training to a bidirectional architecture, presenting a pre-trained model that can be used across a range of NLP tasks with minimal fine-tuning.

5.2 Deep learning approaches to text classification

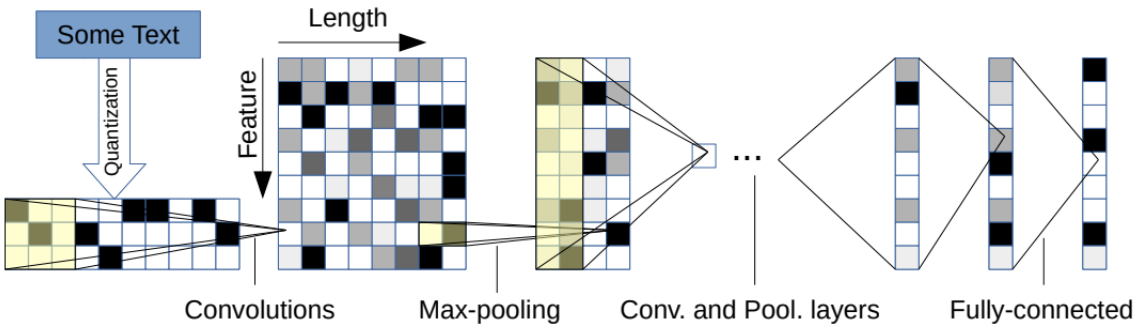


Figure 5.4: Architecture for character-level CNN for text classification (Figure 1, [78]).

More specifically, there have been many studies of the use of deep learning methods for the task of text classification. Zhang et al. [78] explore the use of character-level CNNs for text classification, applying their CNN only on characters rather than words and comparing the performance of their model against traditional NLP models such as multinomial logistic regression with count and TF-IDF vectorizers. They also compare their model against an LSTM with word2vec 300-dimension embeddings. A representation of their architecture is reproduced above, and is broadly similar to the CNN we attempted. They ultimately showed that their model was effective on several news text classification datasets.

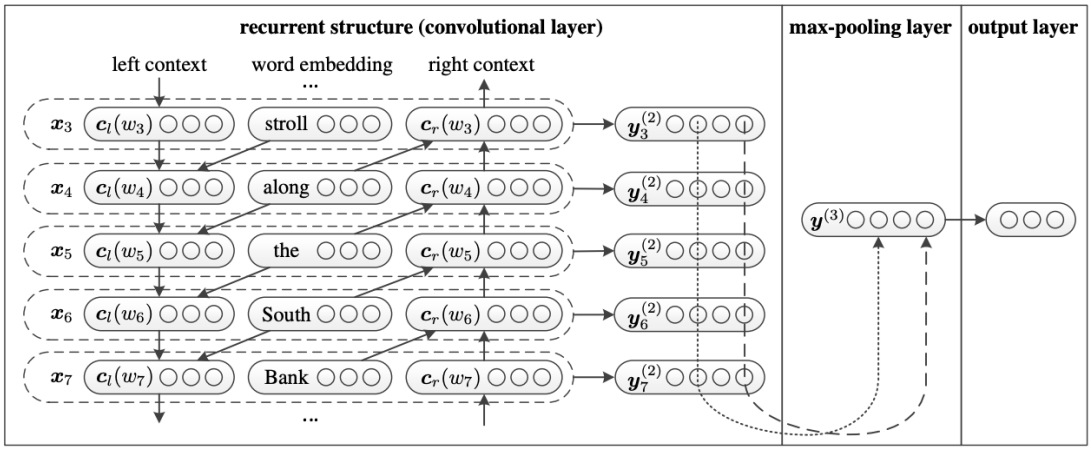


Figure 5.5: Architecture for R-CNN. Word embeddings for a specific word are obtained from those on its left and right (Figure 1, [41]).

Lai et al. [41] use a combination of a CNN and a bi-directional RNN, a recurrent convolutional neural network (RCNN). They obtain word embeddings of a word from the words on its left (the 'left context') through an RNN where input is read in backwards, and its right (the 'right context') from an RNN with input read in forwards. This recurrent structure allows the network to capture contextual information when learning word representations, as shown in their figure above. They then use a max pooling layer to underscore the importance of features which play key roles in the text classification task. Chen et al. [9] take a similar approach to the case of a multi-label text classification task, combining a word-vector based CNN feature extraction with an RNN architecture.

Liu et al. [44] use RNNs with a multi-task learning framework, to learn across multiple related tasks with the aim of avoiding the problem of insufficient training data. Training their models on four different text classification tasks, they show that the joint learning of multiple related tasks together can improve the model's performance across all four text classification tasks, as opposed to learning the tasks separately.

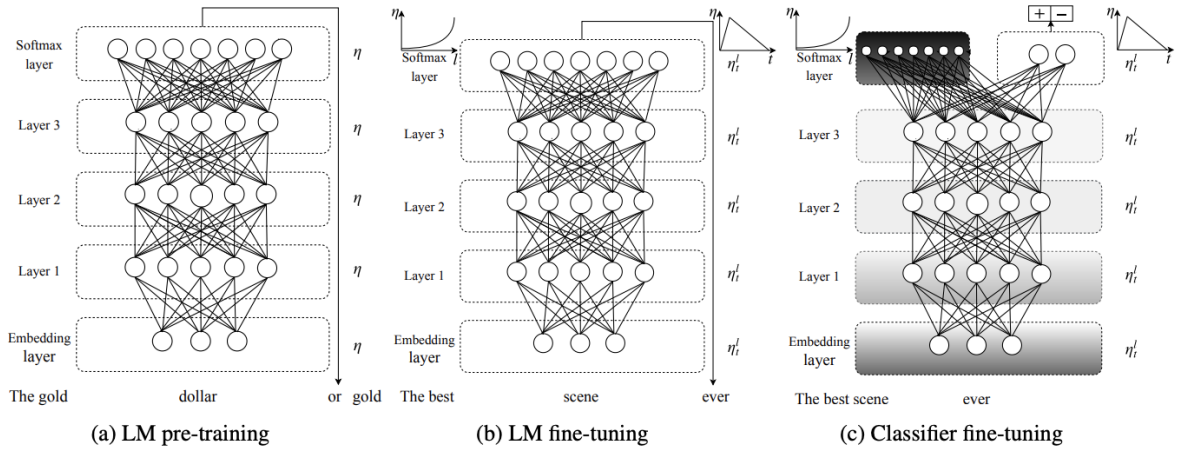


Figure 5.6: The three stages of ULMFiT: pre-training a language model (LM) on a general corpus to capture general language features, fine-tuning the LM on specific task data, and finally fine-tuning the classifier on the target task (Figure 1, [27]).

With regards to transfer learning methods, ULMFiT [27] is particularly promising. They report results specifically on six text classification tasks, although their model can be used for other NLP tasks. Their model has three stages, as shown above. First, a language model (LM) is pre-trained on a general corpus, in their case, processed Wikipedia text. They then fine tune their LM on the data from the specific target task. They use a *discriminative* fine-tuning approach, the intuition being that different layers of the model should have different learning rates, to fine tune them to varying extents. They propose their own formula for determining the learning rate, called the *slanted triangular learning rate*. Finally, they fine-tune their classifier on top of the LM, by adding two linear blocks to the LM. They show that ULMFiT provides comparable classification performance to other models on an IMDB dataset, with 10x less labelled data to train on.

5.3 Text classification in the legal context

In the legal context in particular, the problem of text classification remains an under-explored one [28, 5], providing the motivation for this project. Mencia and Furnkranz [48] evaluate three methods on the multi-label classification task using the original EUR-LEX dataset: the binary relevance approach, multi-label multi-class perceptron approach, and the dual multi-label pairwise perceptron approach. A table of their results was reproduced in Figure 3.1, which shows the latter strategy achieving an average precision of 83.38% on multi-class directory code labels (with over 500 possible labels).

Boella et al. [5] approach the text classification task with a support vector machine (SVM) with TF-IDF representation, which at the time of their publication was the state-of-the-art model for text classification, and in fact was our best model. This approach was also taken by Goncalves and Quaresma [21], who use an SVM on a dataset

of Portuguese legislation as well as a Reuters news dataset. Similarly, de Maat et al. [12], experimenting with binary (a simplified count vectorizer, where a word is represented either present in the sample or not), count and TF-IDF vectorization, show that SVMs perform better than Naive Bayes or decision trees at text classification on a dataset of Dutch laws with 18 categories.

More recently, Sulea et al. [67] examine three tasks: predicting the legal area of a case, predicting the outcome of a case, and estimating the date of the case; the first task is of particular relevance. They propose an ensemble-based SVM, which classifies based on the average output of multiple SVMs; more specifically, it is a mean probability classifier which adds the probability estimates for each class, and chooses as the prediction the class label with the highest average probability. On their dataset of French legislation with 8 possible classes, they show that this method results in an accuracy of 96.8% and F1-score of 96.5%, outperforming their earlier research which uses a single linear SVM [68], which attains an accuracy of 90.2% and an F1-score of 90.3%.

	All Labels			Frequent		Few		Zero	
	RP@5	*nDCG*@5	Micro-*F*1	*RP*@5	*nDCG*@5	*RP*@5	*nDCG*@5	*RP*@5	*nDCG*@5
Exact Match	0.097	0.099	0.120	0.219	0.201	0.111	0.074	0.194	0.186
Logistic Regression	0.710	0.741	0.539	0.767	0.781	0.508	0.470	0.011	0.011
BIGRU-ATT	0.758	0.789	0.689	0.799	0.813	0.631	0.580	0.040	0.027
HAN	0.746	0.778	0.680	0.789	0.805	0.597	0.544	0.051	0.034
CNN-LWAN	0.716	0.746	0.642	0.761	0.772	0.613	0.557	0.036	0.023
BIGRU-LWAN	**0.766**	**0.796**	**0.698**	**0.805**	**0.819**	**0.662**	**0.618**	0.029	0.019
Z-CNN-LWAN	0.684	0.717	0.618	0.730	0.745	0.495	0.454	0.321	0.264
Z-BIGRU-LWAN	0.718	0.752	0.652	0.764	0.780	0.561	0.510	**0.438**	**0.345**
ENSEMBLE-LWAN	**0.766**	**0.796**	**0.698**	**0.805**	**0.819**	**0.662**	**0.618**	**0.438**	**0.345**
MAX-HSS	0.737	0.773	0.671	0.784	0.803	0.463	0.443	0.039	0.028
LW-HAN	0.721	0.761	0.669	0.766	0.790	0.412	0.402	0.039	0.026

Figure 5.7: Extreme multi-label multi-class classification results on EURLEX57K, including logistic regression and HAN (Figure 2, [8]).

Two papers were presented this year at the inaugural Workshop in Natural Legal Language Processing (NLLP). First, Chalkidis et al. [8] update the EURLEX dataset in for the task of 'extreme multi-label text classification' (XMTC), i.e. text classification with many possible classes (over 4,000). They release their novel updated dataset, called EURLEX57K.[31] They first measure the performance of several baseline classifiers, including a naive 'Exact Match' classifier which classifies documents into classes only if their labels appear exactly in the document text, and a logistic regression classifier. They then experiment with eight deep learning methods, including a CNN with an attention mechanism which they termed a 'label-wise attention network' (LWAN) as implemented by Mullenbach et al. [52] and a HAN (which we implemented). They report that that using a 'BIGRU-LWAN' (implementing the LWAN with a bi-directional gated recurrent unit) results in the best performance. These results are reproduced above. Note that they are not comparable to our case, although the subject matter of the dataset is similar and some models implemented are similar, because their task is XMTC.

[31] Available at *http://nlp.cs.aueb.gr/software_and_datasets/EURLEX57K/index.html*

However, some state-of-the-art methods they have not experimented with include using Transformers, which were devised by Vaswani et al. [74] and shown to be more effective and computationally efficient than CNNs and RNNs for other tasks (e.g. machine translation); as well as capitalising on transfer learning and experimenting with pre-trained models such as BERT [13], ELMo [58] and ULMFiT [27], as discussed above.

Howe et al. [28] examine the task they term *legal area classification*, i.e. classifying a judgement into the legal areas it belongs to (e.g. 'property law', 'criminal law' etc.). Each sample can cover more than one area of law, thus it is a multi-label classification task. Each sample is also associated with one or more *topics* within its legal area, e.g. the topics within 'property law' can be 'land', 'purchaser', 'tenant', and so on. They use a dataset they compiled of over 6,000 Singapore judgements. Unfortunately, to our knowledge, this was not publicly released, thus we were unable to run our models on their dataset.

6.1 F1 Score

Subset	10%	50%	100%
$bert_{large}$	45.1 [57.9]	56.7 [63.8]	60.7 [66.3]
$bert_{base}$	43.1 [53.6]	52.0 [57.6]	56.2 [63.9]
$ulmfit$	**45.7** [62.8]	45.9 [63.0]	49.2 [64.3]
$glove_{cnn}$	40.7 [62.2]	58.7 [67.1]	63.1 [70.8]
$glove_{avg}$	36.7 [49.7]	**59.1** [64.3]	61.5 [65.6]
$glove_{max}$	29.2 [47.4]	47.8 [59.9]	52.5 [63.2]
lsa_{250}	37.9 [**63.5**]	55.2 [**70.8**]	**63.2** [**73.3**]
lsa_{100}	30.6 [58.5]	51.8 [68.5]	57.1 [70.8]
$count_{25}$	32.6 [36.1]	31.8 [30.6]	27.7 [28.1]
$base_{pdf}$	5.2 [17.3]	5.5 [16.6]	5.5 [16.6]

Table 2: Macro [Micro] F1 Scores Across Experiments

Figure 5.8: Linear SVM showing the best Micro-F1 in all cases, and best Macro-F1 on their full dataset (Table 2, [28]).

6.2 Precision

Subset	10%	50%	100%
$bert_{large}$	54.7 [65.8]	57.1 [59.7]	63.6 [64.3]
$bert_{base}$	41.4 [45.1]	48.1 [50.0]	61.4 [67.2]
$ulmfit$	49.3 [63.7]	46.6 [61.4]	48.7 [63.2]
$glove_{cnn}$	50.7 [69.8]	63.4 [68.5]	66.7 [72.9]
$glove_{avg}$	**62.5** [68.0]	67.0 [68.1]	64.8 [68.2]
$glove_{max}$	51.3 [65.1]	47.3 [56.6]	59.2 [68.6]
lsa_{250}	56.7 [76.1]	70.0 [81.1]	**83.4** [81.7]
lsa_{100}	52.3 [**77.2**]	**73.8** [**81.9**]	73.9 [**83.7**]
$count_{25}$	30.2 [26.4]	26.4 [19.8]	23.0 [17.8]
$base_{pdf}$	2.9 [10.0]	3.1 [9.5]	3.1 [9.5]

Table 3: Macro [Micro] Precision Across Experiments

Figure 5.9: Linear SVM showing the best precision almost in all cases (Table 3, [28]).

They use a counting strategy as a baseline (similar to the 'exact match' baseline of [8], but the term to be matched must exceed some arbitrarily chosen threshold count), as well as a linear SVM as baseline classifiers. They experiment with pre-trained models, including BERT and ULMFiT, and pre-trained embeddings using GloVe. They held out a test set, and they split their remaining training set in 3 different ways: 10% of the training data, 50% and 100% (all of the training data), and reported performance on their models having trained on each of the sets (as shown above). Interestingly, they note that a one-vs-rest linear SVM with features learned from a subset of data using latent semantic analysis (lsa_{100} and lsa_{250} are linear SVMs trained on 100 and 250 topics extracted by LSA respectively) still had almost consistently the best performance. This was despite the fact that they compared against more sophisticated models such as BERT and ULMFiT. As shown in Figures 5.8 and 5.9 (emphasis on SVMs added with highlighting), the SVMs had almost the best precision and F1 in many cases. However, they do note that the SVMs high F1 was propped up by their high precision, and BERT-based models had the best recall. Still, this was particularly enlightening for us, given that the best performer on our dataset was also a one-vs-rest linear SVM (albeit trained with a different way of selecting the training set).

Finally, we consider some related work specifically on the EURLEX dataset, beyond Mencia and Furnkranz [48] and Chalkidis et al. [8]. Unfortunately, as we have noted, the original EURLEX dataset was for the XMTC task, thus we could not compare our results to other work using the EURLEX dataset. Rubin et al. [63] approach the XMTC task with probabilistic *generative statistical topic models*, such as latent Dirichlet allocation (LDA) which associate individual word tokens with different labels, as opposed to discriminative models such as an SVM. They note that an LDA-based model outperforms the multilabel multiclass perceptron on several multiclass metrics.

Rubin et al. [63] experiment with a neural network with a single hidden layer, and note that using cross entropy as a loss function, as well as a reLU activation function, dropout, and Adagrad for gradient descent optimisation, showed convincing results on multi-label metrics such as rank loss. At the time of their publication, these methods were considered state-of-the-art. Of course, in the fast-moving field of machine learning, these methods have become fairly standard approaches, not only for multi-class classification, but with neural networks in general.

Chapter 6

Conclusions

6.1 Summary of contributions

Text classification is a fairly standard NLP task [36]. For this task, there are several publicly available datasets, which are commonly used: Reuters-21578,[32] Reuters RCV1 (English) and RCV2 (other languages),[33] 20 Newsgroups dataset,[34] and the IMDB movie dataset for sentiment classification.[35]

With reference specifically to legal subject matter, however, it was difficult for us to find a publicly available dataset. We managed to access the EURLEX dataset, but this was for an 'extreme' multi-label, multi-class classification task. Our first step was thus to distill this dataset into one which was appropriate for a single-label, multi-class classification task. We would thus consider our first achievement to be having reshaped the existing EURLEX dataset into one which is appropriate for the standard task of multi-class classification. In fact, we might even argue that with more work, this refined dataset can be published to be used as a standard dataset for legal text classification, which, to our knowledge, is not publicly available. We address this suggestion later, when we discuss future work.

Secondly, we have thoroughly considered both machine learning and deep learning models (per the distinction we drew in our project), through the complete text classification pipeline (as outlined in Section 2.2), from preprocessing to evaluation. Less recent academic work would have only considered machine learning models, with the multilayer perceptron as the most complex model. More recent work has, in our view, tended to consider only more complex models, against a single machine learning model as a baseline, such as a linear support vector machine. We have provided analysis of both these groups of models on the same dataset. We have evaluated and provided visualisations for less popular (from our limited observation) approaches to text classification such as the k-nearest neighbours and decision tree classifiers, and

[32] See *http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html*

[33] See *https://archive.ics.uci.edu/ml/datasets/Reuters+RCV1+RCV2+ Multilingual,+Multiview +Text+Categorization+Test+collection*

[34] See *https://www.kaggle.com/crawford/20-newsgroups*

[35] See *http://ai.stanford.edu/~amaas/data/sentiment/*

more commonly used models like logistic regression and support vector machines. We have also considered models which are more contemporary, such as convolutional and recurrent neural networks, and the hierarchical attention network. In addition, we have experimented with various forms of feature extraction, from the count and TF-IDF vectorizer to pre-trained word embeddings and a trainable embedding layer as part of the hierarchical attention network. Thus, in our view, we have contributed through the breadth of our analysis, particularly in the legal context.

Thirdly, to our knowledge, academic work in the legal context (as discussed in Section 5.3) has generally used pre-trained embeddings from GloVe, word2vec, and fastText. Similarly, tokenization and sentencization has generally been achieved with the Stanford Tokenizer,[36] Natural Language Toolkit,[37] and gensim.[38] We considered a legal domain-specific library, LexNLP, but with our experimentation and to our understanding this simply calls NLTK or the Stanford Tokenizer [32]. Thus, we ran our deep learning experiments with spaCy, and to our knowledge the performance of spaCy with legal texts has not been reported.

Finally, our defined task at the outset was to propose a classification model which, given an English legal document X, can classify that document into one of k predefined classes. We have proposed a model: TF-IDF for feature representation, with a one-vs-rest linear support vector machine. This is by no means a revolutionary proposal. However, we note that this model has long been known to perform especially well on data represented in high-dimensional and sparse vectors [35, 53]. It is also the model which performs best for Goncalves and Quaresma [21], and is the basis for the ensemble model of Sulea et al. [67]. Most significantly, it outperformed BERT and ULMFiT in various measures in Howe et al. [28]. As a result, this choice was not an unexpected outcome.

Thus, although there is certainly very much on which our work has to improve on, as we will discuss below, we consider these to be the academic contributions of our project.

6.2 Practical implications

In legal practice, text classification has many potential uses. Searching legal databases for documents or legislation is an everyday occurrence for most lawyers; common solutions include LexisNexis,[39] Westlaw,[40] and Practical Law.[41] There are also jurisdiction specific databases of legislation, such as EUR-Lex for European Union law, BAILII

[36] See *https://nlp.stanford.edu/software/tokenizer.html*
[37] See *nltk.org*
[38] See *https://radimrehurek.com/gensim/*
[39] See *https://www.lexisnexis.co.uk*
[40] See *https://www.westlaw.co.uk*
[41] See *https://www.practicallaw.co.uk*

for British and Irish law,[42], Congress.gov for US law[43], or SSO for Singapore law,[44] to name only a few. The contents of all these databases are categorised into different classes, usually multi-label, as cases often fall into multiple different areas or subject matter. Such contents are, to our knowledge, often manually hand-labelled.

The databases above generally concern publicly available information, such as judgements and legislation. However, text classification is also important in the private sector. A common task for lawyers is *e-discovery*: this is done in the context of litigation, where lawyers manually go through multiple (possibly thousands of) documents to find documents containing information crucial to a case, and set aside relevant ones. Of course, having an effective information retrieval system would greatly hasten this process, and any such system would require documents to be categorised, which is where text classification is crucial. There are various commercial solutions, such as Exterro.[45]

Similarly, *due diligence* is done often in the context of a merger or an acquisition, where all the relevant documents relating to the company being acquired are hosted in a data room. Often, the data in a data room is not categorised, and it is the lawyers' role to make sense of it, and to ensure that there are no potential loopholes or areas of risk in the documents which could affect the value of the target company. Again, this is often done by manually reviewing documents. Having a scalable, accurate classification system could again be extremely useful. There are various commercial machine learning-based solutions, such as Kira Systems[46] and Luminance.[47]

Thus, a truly effective text classification solution has tremendous potential applications. However, there are two key challenges to this: the first is that such a system must have near complete accuracy. The stakes in commercial contracts are often astronomically high. Thus, if a document were mislabelled by a classifier, the consequences would be far worse than, for example, an image classification system wrongly identifying cat pictures. Hence, a system which has 95.6% classification accuracy, which the linear SVM does on our dataset, may still not be considered successful in commercial terms, although it may alleviate the burden on junior lawyers.

The second challenge is pithily summarised by Zhang, Zhao and LeCun [78]:

> "*There is no free lunch.* Our experiments once again verifies that there is not a single machine learning model that can work for all kinds of datasets."

This is a great obstacle to producing a commercially viable solution. As legal texts come in various lengths, formats, subject matter and even languages, it is our view that finding a model which generalises well remains an open challenge.

[42] See *https://www.bailii.org/*
[43] See *https://www.congress.gov/*
[44] See *https://sso.agc.gov.sg/*
[45] See *https://www.exterro.com/*
[46] See *https://kirasystems.com/*
[47] See *https://www.luminance.com/*

6.3 Challenges

6.3.1 Lack of labelled data

As we have noted, given that text classification is generally a supervised learning task, having sufficient data relevant to the legal domain is vital. As this project was focused on the legal domain, we were unable to obtain a labelled dataset for single-class text classification. We have presented our solution to this challenge, but we have not been able to measure the efficacy of our solutions on another dataset of legal texts, to see if the models we used generalise well.

6.3.2 Hardware limitations

Another challenge we faced is a lack of access to powerful hardware, which resulted in us limiting word embeddings to the first 300 tokens of each sample, and later the first 1,000. Further, the long training times due to hardware limitations forced us to often rely on manual search, rather than a more exhaustive grid search or random search for hyperparameter tuning. As we have noted, this would likely have affected the performance of our deep learning models.

6.4 Possible improvements

On hindsight, there are several ways in which the implementation of this project could have been improved. First, as we have just noted, our approach to hyperparameter tuning more often than not was manual search. This method has been humourously termed *grad student descent* [18], where a graduate student tries hyperparameters to find the best fit. This aptly describes our situation. A grid search may have resulted in marginally better performance in some models, although through the process of manual search, we did not see a drastic change in performance with different hyperparameters.

Secondly, more analysis could have been done during the running of the models. Crucially, an analysis of time and space complexity of models and recording their actual training time should have been done. In practice, a case can perhaps be made for using a model which performs slightly worse in terms of accuracy or related metrics, but is much more efficient in terms of running time. For example, a key benefit of using max pooling with CNNs may be that it reduces the output's dimensions, and thus is effective in terms of training time and memory requirements, but we did not shed light on this.

Finally, we could have done a better theoretical analysis of the models we proposed. From a machine learning perspective, we could have analysed more closely the specific functions and transformations in each layer of our models. From a linguistics perspective, we could have better examined the syntactic structure of our samples. We could have used tools such as spaCy's visualiser[48] to better understand the linguistic

[48] See *https://explosion.ai/demos/displacy*

features of our dataset, and proposed a model based on such an understanding.

6.5 Future work

Aside from the possible improvements cited above, there are ways in which this work can be expanded. First, we can build on what we have done with the original EURLEX dataset to provide a dataset which can be used for single-class English legal text classification. This is very much aided by the fact that the EUR-Lex portal is open for public use, and is constantly updated with labelled samples. Our dataset is very much an imbalanced one, but more labelled samples can be obtained to balance the dataset, increase the number of samples, and publish it for use. Chalkidis et al. [8] updated and released EURLEX57K, but that remains a dataset for XMTC. Several articles cited in related work also use labelled legal datasets, but in French [68] and Portuguese [21]. Howe et al. [28] use an English-language labelled legal dataset, but to our knowledge this was not released for public use.

In addition, we can experiment with many more state-of-the-art models and language models and apply them to the text classification task, including BERT, ELMo, and ULMFiT. We can also use transformers [74], which use only attention mechanisms without CNNs or RNNs. While so far we have applied models designed by others to the data that we have, in future work we could propose our own models which generalise well, although this may prove to be a tall order.

6.6 Legal and ethical considerations

This project relies largely on the EURLEX dataset, which has been made freely available as part of the paper presented by Mencia and Furnkranz [48].[49] The dataset is also freely available to be modified, as we have done in distilling its contents to ensure that it is appropriate for a single-label classification task. In addition, the dataset itself is compiled from data obtained from the EUR-Lex portal. Any commercial or non-commercial use of data from EUR-Lex is authorised, provided its source is cited, as we have done.[50]

With regards to Section 4 of the ethics checklist (see Appendix A), this project does not collect 'personal data', as defined by Article 4(1) of the General Data Protection Regulation (GDPR).[51] This project does not supplement the data provided in the EUR-Lex portal by the European Union in any way. The EUR-Lex portal itself collects personal data, but this is in compliance with personal data protection regula-

[49] For the specific terms and conditions, see
http://www.ke.tu-darmstadt.de/resources/eurlexsection-13

[50] For the European Union's terms and conditions, see
https://eur-lex.europa.eu/content/legal-notice/legal-notice.html2.%20droits

[51] For the full definition, see
https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679

tions, and authorised under Regulation (EU) 2018/1725.[52]

However, where 'processing' is defined under Article 4(2) GDPR,[53] this project may involve 'personal data processing' or 'further processing of previously collected personal data'. The raw text samples in the EURLEX dataset may include information relating to natural persons, such as names. We have performed 'operations' on this data for the purposes of data analysis in our project. However, all such processing has been lawful under Article 6(1) of the GDPR on the lawfulness of processing.[54] Indeed, the data has been used in accordance with the copyright terms of the European Union, which itself is the very source of the GDPR.

With regards to Section 8, we consider our project to have an exclusive focus on civil applications, as defined by the European Commission's guidance.[55] There is no intention for it to be used in military application or serve military purposes.

With regards to Section 10, we have not produced any software for which there are copyright licensing implications. However, a potential copyright implication which arises is that the dataset we have used, and the modified data we will reproduce,[56] is provided under the permissions of Mencia and Furnkranz [48] and the European Union. It follows that any further use of the data is subject to the same copyright rules as we have been subject to.

We do not consider that this project confronts any ethical concerns, other than the fact that machine learning is in general related to artificial intelligence, which itself is a source of much ethical debate for various reasons such as its potential to cause job displacement. However, we consider such matters to be beyond the scope of this project, given that our aim is mainly to evaluate and present models which can classify legal documents accurately. We have considered - and our view is that this project is in line with - the BCS Code of Conduct, IET Rules of Conduct, and Engineering Council Statement of Ethical Principles.

[52] See https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32018R1725
[53] *ibid.*
[54] *ibid.*
[55] For the full guidance note, see
https://ec.europa.eu/research/participants/data/ref/h2020/other/hi/guide_research-civil-apps_en.pdf
[56] Data provided in *data.csv*

Appendix A

Ethics checklist

	Yes	No
Section 1: HUMAN EMBRYOS/FOETUSES		
Does your project involve Human Embryonic Stem Cells?		X
Does your project involve the use of human embryos?		X
Does your project involve the use of human foetal tissues / cells?		X
Section 2: HUMANS		
Does your project involve human participants?		X
Section 3: HUMAN CELLS / TISSUES		
Does your project involve human cells or tissues? (Other than from "Human Embryos/Foetuses" i.e. Section 1)?		X
Section 4: PROTECTION OF PERSONAL DATA		
Does your project involve personal data collection and/or processing?	X	
Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)?		X
Does it involve processing of genetic information?		X
Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc.		X
Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets?	X	
Section 5: ANIMALS		
Does your project involve animals?		X
Section 6: DEVELOPING COUNTRIES		
Does your project involve developing countries?		X
If your project involves low and/or lower-middle income countries, are any benefit-sharing actions planned?		X
Could the situation in the country put the individuals taking part in the project at risk?		X
Section 7: ENVIRONMENTAL PROTECTION AND SAFETY		
Does your project involve the use of elements that may cause harm to the environment, animals or plants?		X
Does your project deal with endangered fauna and/or flora / protected areas?		X
Does your project involve the use of elements that may cause harm to humans, including project staff?		X
Does your project involve other harmful materials or equipment, e.g. high-powered laser systems?		X
Section 8: DUAL USE		
Does your project have the potential for military applications?		X
Does your project have an exclusive civilian application focus?	X	
Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items?		X
Does your project affect current standards in military ethics – e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons?		X
Section 9: MISUSE		

1

Does your project have the potential for malevolent/criminal/ terrorist abuse?		X
Does your project involve information on/or the use of biological-, chemical-, nuclear/radiological-security sensitive materials and explosives, and means of their delivery?		X
Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied?		X
Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cybersecurity related project?		X
SECTION 10: LEGAL ISSUES		
Will your project use or produce software for which there are copyright licensing implications?		X
Will your project use or produce goods or information for which there are data protection, or other legal implications?	X	
SECTION 11: OTHER ETHICS ISSUES		
Are there any other ethics issues that should be taken into consideration?		X

Appendix B

Copy of readme for code repository

Project files

This is a summary of how to run the models and visualisations in this project.

We provide the files for running all the models discussed in the project.

- Download and save the **data.csv** file.
- Due to the file size limitations in CATe, we have hosted the file here.
- It contains 3 columns per line:
    1. "Text": consisting of the raw text samples, extracted from HTML source
    2. "Class": the class label belonging to the text sample
    3. "Cleaned": the preprocessed text samples, for easy access. We discuss the preprocessing steps thoroughly in the project, and also provide the functions for preprocessing in our files.
- Files provided:
    1. *dataset.py*: get word counts, visualise dataset
    2. *machine_learning_models.py*: run machine learning models
    3. *deep_learning_models.py*: run deep learning models (excl. HAN)
    4. *han.py*: run hierarchical attention network (HAN) model
    5. *decision_boundary_visualisation.py*: visualisation for k-NN classifier
    6. *extract.py*: extract text samples from raw html files
    7. *decision_tree.pdf*: image of decision tree visualisation
    8. *random_forest.pdf*: image of random forest visualisation

Libraries

- tensorflow (machine learning library)
- keras (ML library, used with Tensorflow backend)
- pandas (for data handling)
- spaCy (for various NLP tasks)
- mlxtend (for some plotting and additional functions, statistical testing)
- adjustText (to adjust matplotlib plot overlaps)
- seaborn (for visualisation)
- matplotlib (for visualisation)
- beautifulsoup4 (for scraping raw html text files)

Installation

Note that we do not use the latest version of matplotlib (v3.1.1), because of a bug in plotting confusion matrices.

With Python 3.x, using pip:

```
pip install scikit-learn tensorflow keras pandas spacy mlxtend adjustText seaborn matpl
python -m spacy download en_core_web_lg
```

Usage of *dataset.py*

Import all dependencies and function definitions in the file.

We acknowledge that the cleanup_text functions and plot_tokens_clean function are based onthis kernel.

- Store data in a pandas dataframe, 'df'
- Pass in as a string the file location where data.csv has been saved.

```
df = load_data('/.../data.csv')
```

- load the largest nlp model from spaCy

```
nlp = spacy.load('en_core_web_lg')
```

- this function takes loaded pandas dataframe as argument
- prints the counts of each sample in each class
- plots bar chart of distributions of samples in each class
- prints counts of tokens/sentences per sample
- prints bar charts of occurrences of most common words in each class

```
class_distributions(df)
```

- plot a visualisation of spaCy's embeddings of some common words

```
visualise_embeddings()
```

- visualise our dataset with count or TF-IDF vectorizer, with dimensionality reduction to 2D
- takes in the dataframe and two strings as parameters
- the first string is the vectorizer ("count" or "tfidf")
- the second string chooses whether to visualise the full dataset or only the test set ("full" or "test")

```
visualise_data(df, "count", "full")
```

Usage of *machine_learning_models.py*

Import all dependencies and function definitions in the file.

We acknowledge that the plot_confusion_matrix function is based onthis article.

- Store data in a pandas dataframe, 'df'
- Pass in as a string the file location where data.csv has been saved.

```
df = load_data('/.../data.csv')
```

- the run_model function takes three parameters, returns a trained classifier, vectorizer, and confusion matrix; prints classification report from sklearn, accuracy and macro-F1
- parameters:
    1. df - the dataframe we loaded above
    2. vectorizer - the name of the vectorizer (options: "count" or "tfidf")
    3. classifier - the name of the classifier
    - classifier options:
        - "naive_bayes"
        - "decision_tree"
        - "random_forest",
        - "logistic_regression"
        - "linear_svm",
        - "nonlinear_svm"
        - "knn"
        - "mlp"

```
vectorizer, classifier, cm = run_model(df, "count", "logistic_regression")
```

- next, run another classifier if desired

```
vectorizer2, classifier2, cm2 = run_model(df, "tfidf", "linear_svm")
```

- print confusion matrix
- takes confusion matrix object returned by run_model as argument
- if normalize=True, prints a normalised confusion matrix
- change title text string as desired

```
plot_confusion_matrix(cm, normalize=False, target_names=[i for i in range(1,21)],
 title='Confusion Matrix')
```

- run a mcnemar's test if desired (pass in two classifiers returned from run_model)

```
stat_test(df, classifier1, classifier2)
```

- generate plot of logistic regression coefficients
- pass in classifier and vectorizer objects as arguments
- only works if passing in a logistic regression classifier

```
plot_lr_coef(classifier, vectorizer)
```

- generate k-NN performance graph used in our report
- values in this plot are hard-coded from our results

```
knn_plot()
```

Usage of *deep_learning_models.py*

Import all dependencies and function definitions in the file.

- Store data in a pandas dataframe, 'df'
- Pass in as a string the file location where data.csv has been saved.

```
df = load_data('/.../data.csv')
```

- load the largest nlp model from spaCy
- by default, the spacy model takes in inputs of max length 1,000,000 chars
- our dataset has max char length 2,871,868, so we need to configure this

```
nlp = spacy.load('en_core_web_lg')
nlp.max_length = 2871868
```

- choose some options
- if using word embeddings, MAX_LENGTH controls max number of words per sample
- if using sentence embeddings, MAX_LENGTH controls max number of sentences per sample

```
MAX_LENGTH = 300 # the max length per sample (choose wisely)
NB_EPOCHS = 50 # number of epochs over which to run. choose some integer, ideally <100
EARLY_STOPPING = True
PATIENCE = 5 # patience for early stopping, if set to True. choose some integer < NB_EPC
```

- we run the function to add embeddings from spacy to our dataframe
- function takes in 3 parameters, returns our df
    1. df - the pandas dataframe containing our data
    2. embedding - the type of embedding (options: "word_embeddings", "sentence_embeddings")
    3. MAX_LENGTH - options: an integer between 1 to 300,000 but due to memory requirements, ideally <=1,000. Note: setting length=1000 already loads ~75GB of embeddings in memory

```
df = get_embeddings(df, "word_embeddings", MAX_LENGTH)
```

- we run the neural network model, function takes in 6 parameters, including those defined above:
    1. df - the pandas dataframe containing our data
    2. architecture - the type of architecture (options: "mlp", "cnn", "ngram_cnn", "lstm", "bi_lstm")
        - note: "ngram_cnn" is based on the CNN implemented in (Kim, 2014) as discussed in our report
    3. MAX_LENGTH
    4. NB_EPOCHS
    5. EARLY_STOPPING
    6. PATIENCE

```
model, cm = run_model(df, "mlp", MAX_LENGTH, NB_EPOCHS, EARLY_STOPPING, PATIENCE)
```

- plot confusion matrix (normalised or not), if desired

```
plot_confusion_matrix(cm, normalize=False, target_names=[i for i in range(1,21)])
```

Usage of *han.py*

We acknowledge that the implementation of the dot_product function and AttentionWithContext layer, and general design of the network, is based on this and that repository.

However, based on those existing implementations, we configured the network to work in our context and for our purposes.

Import all dependencies and function definitions in the file.

- Store data in a pandas dataframe, 'df'
- Pass in as a string the file location where data.csv has been saved.

```
df = load_data('/.../data.csv')
```

- set some parameters which can be chosen

```
MAX_WORDS = 261737  # number of unique tokens in our training set
MAX_SENTS = 300 # we stick to max. 300 sentences per sample
MAX_SENT_LENGTH = 300 # we stick to max. 300 tokens per sentence
VALIDATION_SPLIT = 0.2 # same training:validation split, 80:20
EMBEDDING_DIM = 300  # we stick to embedding dimensions of 300
```

- run the model

```
model, cm = run_model(df, MAX_WORDS, MAX_SENTS, MAX_SENT_LENGTH, VALIDATION_SPLIT, EMBEI
```

- plot the confusion matrix

```
plot_confusion_matrix(cm, normalize=False, target_names=[i for i in range(1,21)], title=
```

Usage of *decision_boundary_visualisation.py*

Run the entire python script.

We used a list of 20 distinct colours fromhere.

- It generates a plot of our k-NN classifier's decision boundaries.

- The dimensions of our features have been reduced to 2D, so this is not representative of the actual classifiers we trained.
- In the file there is a variable, *vectorizer*, which is set to the *TfidfVectorizer()*.
- Change this to *CountVectorizer()* if desired.

```
# vectorizer = CountVectorizer()
vectorizer = TfidfVectorizer() # choose vectorizer
```

- There are also variables, *k=1* and *weights='distance'*.
- *k* can be reset to any desired value of *k*, and the corresponding plot will be generated.
- *weights* can be set to *'distance'* or *'uniform'*.

```
k = 1 # choose value for k-NN
# weights = 'uniform'
weights = 'distance' # choose distance weighting
classifier = KNeighborsClassifier(n_neighbors=k, weights='distance')
```

- run the entire python script once decided.

Usage of *extract.py*

This file extracts the text samples from the raw HTML files from the original EURLEX dataset. This can be accessed here.

To run this file, first download and unzip all raw HTML files from here. Note the directory in which these files have been saved.

Create a new empty directory where the extracted text files will be stored.

- Import dependencies

```
import os
from bs4 import beautifulsoup
```

- Initialise variables containing the filepath of the directory where all the raw HTML files are, and the directory where the extracted text files will be stored.

```
base_dir = "/..." # the location of raw HTML files
second_dir = "/..." # the location of extracted files
```

Run the rest of the script.

The output of this process is the body text of each file, with a CELEX ID. We then matched the CELEX IDs with the document IDs in this file.

At this stage, we have the body texts, with document IDs. We then matched the document IDs with the top level directory codes, which are the class labels in the file *id2class_eurlex_DC_l1.qrels,* from this zip file.

Note that the output of this process has been saved in a format which is easy to work with, especially with a pandas dataframe, in the file **data.csv**.

Acknowledgements

EURLEX dataset modified with permission.

Access the original dataset here.

See the original paper here.

Access the full EUR-Lex repository here.

Bibliography

[1] M.A. Aizerman, E.M. Braverman, and L.I. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. 1964. https://cs.uwaterloo.ca/ y328yu/classics/kernel.pdf. pages 23

[2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. 1994. http://ai.dinfo.unifi.it/paolo//ps/tnn-94-gradient.pdf/. pages 28

[3] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. 2012. http://jmlr.csail.mit.edu/papers/volume13/bergstra12a/bergstra12a.pdf. pages 8

[4] Christopher Bishop. Pattern recognition and machine learning. 2006. http://users.isr.ist.utl.pt/ wurmd/Livros/school/Bishop%20-%20Pattern%20 Recognition%20And%20Machine%20Learning%20- %20Springer%20%202006.pdf. pages 13

[5] Guido Boella, Luigi Di Caro, and Llio Humphreys. Using classification to support legal knowledge engineers in the eunomos legal document management system. 2011. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.592.5468rep=rep1 type=pdf. pages 3, 87

[6] Guido Boella, Luigi Di Caro, Llio Humphreys, Livio Robaldo, Piercarlo Rossi, and Leendert van der Torre. Eunomos, a legal document and knowledge management system for the web to provide relevant, reliable and up-to-date information on the law. 2012. http://www.semantic-web-journal.net/system/files/swj322_0.pdf. pages 3

[7] Jose Camacho-Collados and Mohammad Taher Pilehvar. On the role of text preprocessing in neural network architectures: An evaluation study on text categorization and sentiment analysis. 2018. https://arxiv.org/pdf/1707.01780.pdf. pages 14

[8] Ilias Chalkidis, Manos Fergadiotis, Prodromos Malakasiotis, Nikolaos Aletras, and Ion Androutsopoulos. Extreme multi-label legal text classification: A case study in eu legislation. 2019. https://arxiv.org/abs/1905.10892. pages 88, 90, 96

[9] Guibin Chen, Deheng Ye, Zhenchang Xing, Jieshan Chen, and Erik Cambria. Ensemble application of convolutional and recurrent neural networks for multi-label text categorization. 2017. https://sentic.net/convolutional-and-recurrent-neural-networks-for-text-categorization.pdf. pages 86

[10] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. 2008. https://ronan.collobert.com/pub/matos/2008_nlp_icml.pdf. pages 25, 26, 83

[11] Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. 2011. http://www.jmlr.org/papers/volume12/collobert11a/collobert11a.pdf. pages 83, 84

[12] Emile de Maat, Kai Krabben, and Radboud Winkels. Machine learning versus knowledge based classification of legal texts. 2010. https://www.researchgate.net/publication/220809905_Machine_Learning_versus_Knowledge_Based_Classification_of_Legal_Texts. pages 88

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018. https://arxiv.org/abs/1810.04805. pages 84, 85, 89

[14] Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. 1997. https://sci2s.ugr.es/keel/pdf/algorithm/articulo/dietterich1998.pdf. pages 13

[15] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. 1997. https://link.springer.com/content/pdf/10.1023%2FA%3A1007413511361.pdf. pages 19

[16] Allen Edwards. Note on the âĂIJcorrection for continuityâĂİ in testing the significance of the difference between correlated proportions. 1948. pages 13

[17] Rong-En Fan and Chih-Jen Lin. A study on threshold selection for multi-label classification. 2007. https://www.csie.ntu.edu.tw/ cjlin/papers/threshold.pdf. pages 12

[18] Oguzhan Gencoglu, Mark van Gils, Esin Guldogan, Chamin Morikawa, Mehmet SÃijzen, Mathias Gruber, Jussi Leinonen, and Heikki Huttunen. Hark side of deep learning - from grad student descent to automated machine learning. 2019. https://arxiv.org/pdf/1904.07633.pdf. pages 95

[19] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. 2011. http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf. pages 25

[20] Yoav Goldberg. A primer on neural network models for natural language processing. 2015. https://arxiv.org/abs/1510.00726. pages 15, 17, 27, 84

[21] Teresa Goncalves and Paulo Quaresma. Evaluating preprocessing techniques in a text classification problem. 2005. http://www.di.uevora.pt/ pq/papers/enia-goncalves-quaresma.pdf. pages 87, 93, 96

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. 2016. https://www.deeplearningbook.org/contents/convnets.html. pages 25

[23] Bhumika Gupta, Aditya Rawat, Akshay Jain, Arpit Arora, and Naresh Dhami. Analysis of various decision tree algorithms for classification in data mining. 2017. https://pdfs.semanticscholar.org/fd39/e1fa85e5b3fd2b0d000230f6f8bc9dc694ae.pdf. pages 20

[24] Isabelle Guyon. A scaling law for the validation-set training-set size ratio. 1997. https://pdfs.semanticscholar.org/452e/6c05d46e061290fefff8b46d0ff161998677.pdf. pages 7

[25] Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. 1997. https://www.bioinf.jku.at/publications/older/2604.pdf/. pages 28

[26] Sepp Hochreiter, Yoshua Bengio, and Paolo Frasconi. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. 2001. https://www.bioinf.jku.at/publications/older/ch7.pdf/. pages 28, 80

[27] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. 2018. https://arxiv.org/abs/1801.06146. pages 87, 89

[28] Jerrold Soh Tsin Howe, Lim How Khang, and Ian Ernst Chai. Legal area classification: A comparative study of text classifiers on singapore supreme court judgments. 2019. https://arxiv.org/abs/1904.06470. pages 3, 87, 89, 90, 93, 96

[29] HSBC. Peer group analysis: Investment trends in legal technology. 2018. https://www.business.hsbc.uk/-/media/library/business-uk/pdfs/financing-investments-in-legal-tech-2018.pdf. pages 3

[30] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification. 2016. https://www.csie.ntu.edu.tw/ cjlin/papers/guide/guide.pdf. pages 23, 69

[31] Jin Huang and Charles X. Ling. Using auc and accuracy in evaluating learning algorithms. 2005. https://dl.acm.org/citation.cfm?id=1048850. pages 15

[32] Michael J Bommarito II, Daniel Martin Katz, and Eric M Detterman. Lexnlp: Natural language processing and information extraction for legal and regulatory texts. 2018. https://pdfs.semanticscholar.org/4814/b59b635292d0ab4812b5b611746ce2eee81f.pdf. pages 93

[33] Alon Jacovi, Oren Sar Shalom, and Yoav Goldberg. Understanding convolutional neural networks for text classification. 2018. https://www.aclweb.org/anthology/W18-5408. pages 25

[34] Gareth James. An introduction to statistical learning: with applications in r. 2013. http://faculty.marshall.usc.edu/gareth-james/ISL/ISLR%20Seventh%20Printing.pdf. pages 7, 8

[35] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. 1998. http://www.cs.cornell.edu/people/tj/publications/joachims_98a.pdf. pages 17, 22, 65, 93

[36] Dan Jurafsky and James H. Martin. *Speech and Language Processing*. 2019. https://web.stanford.edu/ jurafsky/slp3/ed3book.pdf. pages 20, 21, 92

[37] Yoon Kim. Convolutional neural networks for sentence classification. 2014. https://arxiv.org/pdf/1408.5882.pdf. pages 26, 78

[38] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text classification algorithms: A survey. 2019. https://arxiv.org/pdf/1904.08067.pdf. pages 14, 15

[39] Maria Krankel and Hee-Eun Lee. Text classification with hierarchical attention networks. 2019. https://humboldt-wi.github.io/blog/research/information_systems_1819/group5_han/. pages 29

[40] Vrushali Y Kulkarni and Dr Pradeep K Sinha. Pruning of random forest classifiers: A survey and future directions. 2012. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=6282329. pages 21

[41] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. 2015. https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/download/9745/9552. pages 86

[42] Yann LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. 1989. https://www.ics.uci.edu/ welling/teaching/273ASpring09/lecun-89e.pdf. pages 25

[43] Ruey-Hsia Li
and Geneva G. Belford. Instability of decision tree classification algorithms. 2017. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.12.8094rep=rep1type=pdf. pages 49

[44] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Recurrent convolutional neural networks for text classification with multi-task learning. 2016. https://arxiv.org/abs/1605.05101. pages 86

[45] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. 2011. https://ai.stanford.edu/ amaas/papers/wvSent_acl2011.pdf. pages 3

[46] Christopher D. Manning and Hinrich Schutze. Foundations of statistical natural language processing. 1999. https://www.cs.vassar.edu/ cs366/docs/Manning_Schuetze_StatisticalNLP.pdf. pages 11

[47] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. Introduction to information retrieval. 2009. https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html. pages 12, 14

[48] Eneldo Loza Mencia and Johannes Furnkranz. Efficient multilabel classification algorithms for large-scale problems in the legal domain. 2010. www.ke.tu-darmstadt.de/publications/papers/loza10eurlex.pdf. pages 18, 32, 33, 34, 35, 46, 47, 87, 90, 96, 97

[49] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. 2013. https://arxiv.org/abs/1301.3781. pages 17, 18, 72, 84

[50] Arvind Mohan and Datta V. Gaitonde. A deep learning based approach to reduced order modeling for turbulent flow control using lstm neural networks. 2018. https://arxiv.org/abs/1804.09269. pages 29

[51] Tony Mullen and Nigel Collier. Sentiment analysis using support vector machines with diverse information sources. 2004. https://www.ijcaonline.org/archives/volume180/number34/pahwa-2018-ijca-916865.pdf. pages 14

[52] James Mullenbach, Sarah Wiegreffe, Jon Duke, Jimeng Sun, and Jacob Eisenstein. Explainable prediction of medical codes from clinical text. 2018. https://www.aclweb.org/anthology/N18-1100. pages 88

[53] Jinseok Nam, Jungi Kim, Eneldo Loza Mencia, Iryna Gurevych, and Johannes Furnkranz. Large-scale multi-label text classification âĂŤ revisiting neural networks. 2014. http://www.ke.tu-darmstadt.de/publications/papers/NN_MLC_ecml2014_camera_ready.pdf. pages 93

[54] Christopher Olah. Understanding lstm networks. 2015. http://colah.github.io/posts/2015-08-Understanding-LSTMs/. pages 28

[55] Bhumika Pahwa, S. Taruna, and Neeti Kasliwal. Sentiment analysis - strategy for text pre-processing. 2018. https://www.ijcaonline.org/archives/volume180/number34/pahwa-2018-ijca-916865.pdf. pages 14

[56] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825. pages 15, 17, 45

[57] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation (2014). 2014. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.671.1743. pages 18, 72

[58] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. 2018. https://arxiv.org/abs/1802.05365. pages 84, 89

[59] Sebastian Raschka. Why is logistic regression considered a linear model? 2013. https://sebastianraschka.com/faq/docs/logistic_regression_linear.html. pages 21, 55

[60] Sebastian Raschka. Naive bayes and text classification i âĂŞ introduction and theory. 2014. https://arxiv.org/abs/1410.5329. pages 18, 24, 46

[61] Irina Rish. An empirical study of the naive bayes classifier. 2001. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.330.2788rep=rep1 &type=pdf. pages 19

[62] Stephen Robertson. Understanding inverse document frequency: On theoretical arguments for idf. 2004. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.97.7340rep=rep1 &type=pdf. pages 17

[63] Timothy N. Rubin, America Chambers, Padhraic Smyth, and Mark Steyvers. Statistical topic models for multi-label document classification. 2011. https://link.springer.com/content/pdf/10.1007%2Fs10994-011-5272-5.pdf. pages 90, 91

[64] Stuart Russell and Peter Norvig. Artificial intelligence: A modern approach. 2016. http://thuvien.thanglong.edu.vn:8081/dspace/bitstream/DHTL_123456789/4010 /1/CS503-2.pdf. pages 7, 20, 21

[65] Frederick Schauer. Is law a technical language? 2015. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2689788. pages 3

[66] The Law Society. Lawtech adopotion research: Feburary 2019. 2019. https://www.lawsociety.org.uk/support-services/research-trends/documents/law-society-lawtech-adoption-research-2019/. pages 3

[67] Octavia-Maria Sulea, Marcos Zampieri, Shervin Malmasi, Mihaela Vela, Liviu P. Dinu, and Josef van Genabith. Exploring the use of text classification in the legal domain. 2017. https://arxiv.org/pdf/1710.09306.pdf. pages 3, 88, 93

[68] Octavia-Maria Sulea, Marcos Zampieri, Mihaela Vela, and Josef van Genabith. Predicting the law area and decisions of french supreme court cases. 2017. https://arxiv.org/abs/1708.01681. pages 88, 96

[69] Alaa Tharwat. Classification assessment methods. 2018. https://www.sciencedirect.com/science/article/pii/S2210832718301546. pages 9, 36

[70] Michal Toman, Roman Tesar, and Karel Jezek. Influence of word normalization on text classification. 2006. https://www.researchgate.net/publication/250030718_Influence_of_Word_Normalization_on_Text_Classification. pages 45

[71] Michal Tomana, Roman Tesara, and Karel Jezek. Influence of word normalisation on text classification. 2006. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=DE533117C33E2A402501D2FCF32A2AEE?doi=10.1.1.83.6363rep=rep1&type=pdf. pages 14

[72] Bruno Trstenjak, Sasa Mikac, and Dzenana Donko. Knn with tf-idf based framework for text categorization. 2013. https://www.sciencedirect.com/science/article/pii/S1877705814003750. pages 23, 24

[73] Alper Kursat Uysal and Serkan Gunal. The impact of preprocessing on text classification. 2014. https://www.sciencedirect.com/science/article/pii/S0306457313000964. pages 44, 45

[74] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017. https://arxiv.org/abs/1706.03762. pages 89, 96

[75] Linda Wetzel. Types and tokens: Stanford encyclopedia of philosophy. 2006. https://plato.stanford.edu/entries/types-tokens/. pages 14

[76] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. 2016. http://www.cs.cmu.edu/ ./hovy/papers/16HLT-hierarchical-attention-networks.pdf. pages 29, 30

[77] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. 2018. https://arxiv.org/abs/1708.02709. pages 84

[78] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. 2016. https://arxiv.org/abs/1509.01626. pages 85, 86, 94