OPEN SET TEXT CLASSIFICATION USING NEURAL NETWORKS

by

VINODINI MOLUKUVAN VENKATARAM

B.E., Karnataka University, India, 2000

A thesis submitted to the Graduate Faculty of the

University of Colorado Colorado Springs

in partial fulfillment of the

requirements for the degree of

Master of Science

Department of Computer Science

2018
This thesis for the Master of Science degree by

Vinodini Molukuvan Venkataram

has been approved for the

Department of Computer Science

by

Jugal Kalita, Chair

Jonathan Ventura

Abhijit Bendale

Date: December 12, 2018
ABSTRACT

In a closed set environment, classifiers are trained on examples from a number of known classes and tested with unseen examples belonging to the same set of known classes. However, in most real-world scenarios, a trained classifier is likely to come across novel examples that do not belong to any of the known classes. Such examples should ideally be categorized as belonging to an unknown class. The goal of an open set classifier is to identify and classify test examples of classes unseen during training. The classifier should be able to declare that a test example belongs to a class it does not know when necessary, and possibly, incorporate it into its knowledge as an example of a new class it has encountered.

There is some published research in open set image classification, but open set text classification remains mostly unexplored. In this thesis, we investigate the suitability of various neural networks, viz. Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Hierarchical Neural Networks (HNNs) and Hierarchical Neural Networks with Attention, for open set text classification. We find that CNNs are good feature extractors compared to other neural networks and hence perform better than existing state-of-the-art open set classifiers in smaller domains meaning training the classifier on a low number of classes and testing the classifier on a high number of classes (e.g., train on 2 classes and test on 10 classes), although their open set classification abilities in higher domains (e.g., train on 7 classes and test on 10 classes) in general still need to be explored deeper.
To my husband and two lovely kids
ACKNOWLEDGEMENTS

The author wishes to sincerely thank the following individuals:

- Dr. Kalita for the opportunity, guidance, knowledge, and help,
- Dr. Ventura for the research opportunities,
- My husband, Jayadeva, for all the motivation, support, and help,
- My kids, Harshith and Yeshitha for inspiration, and
- My family and friends for all the encouragement and help.
# TABLE OF CONTENTS

**CHAPTER**

1 **INTRODUCTION**

1.1 Closed Set and Open Set Text Classification ........................................ 2  
1.2 Approach ................................................................................................. 5  
1.3 Organization of the Thesis ....................................................................... 6  

2 **PROBLEM STATEMENT AND SOLUTION** .............................................. 7  
2.1 Datasets and Techniques Used ............................................................... 9  

3 **RELATED WORK** ................................................................................... 11  
3.1 Background ............................................................................................ 11  
3.2 Related Work in Open Set Text Classification ...................................... 14  
3.3 Related Work in Outlier Detection ......................................................... 16  
3.4 Related Work in Outlier Ensemble ......................................................... 21  
3.5 Summary ................................................................................................. 23  

4 **APPROACHES AND ARCHITECTURES** ............................................. 24  
4.1 Recurrent Neural Network (RNN) .......................................................... 26  
4.1.1 LSTM .............................................................................................. 28  
4.2 Bidirectional LSTM (BiLSTM) .............................................................. 31  
4.3 Hierarchical Neural Network (HNN) ..................................................... 32  
4.4 Hierarchical Neural Network with Attention (HNNA) .......................... 34
5.5 Results of Hierarchical Neural Network Architecture . . . . . . . . . . . . 60
5.6 Results of Hierarchical Neural Network With Attention Architecture . . . . 61
5.7 Results of Convolutional Neural Network Architecture . . . . . . . . . . . 61
  5.7.1 Mahalanobis Distance . . . . . . . . . . . . . . . . . . . . . . . . 64
  5.7.2 Local Outlier Factor . . . . . . . . . . . . . . . . . . . . . . . . 68
  5.7.3 Isolation Forest . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
5.8 Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 70
  5.8.1 Preliminary Experiment Results . . . . . . . . . . . . . . . . . . . 70
  5.8.2 Experiment Results . . . . . . . . . . . . . . . . . . . . . . . . . . 70

6 CONCLUSION AND FUTURE WORK 76
  6.1 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76
  6.2 Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 77

REFERENCES 80
# LIST OF FIGURES

1.1 Text Classification .................................................. 1  
1.2 Closed Set Text Classification ................................. 3  
1.3 Open Set Text Classification ........................................ 4  
2.1 Block Diagram of Open Set Text Classifier ............... 9  
4.1 Block Diagram of Open Set Text Classifier ............... 25  
4.2 RNN Cell ................................................................. 27  
4.3 Text Classification Using RNN ............................... 28  
4.4 LSTM Cell (Adapted from [84]) ............................. 29  
4.5 Text Classification Using BiLSTMs ....................... 32  
4.6 Hierarchical Neural Network for Classification .......... 33  
4.7 Hierarchical Neural Network with Attention for Classification (Adapted from [86]) .............................. 35  
4.8 Stages of CNN Architecture in Text Classification (Adapted from [79]) .................................................. 39  
4.9 Convolutional Neural Network Architecture for Classification (Adapted from [33]) ........................................ 42  
4.10 Weibull Distribution for various values of \( \lambda \) and \( k \) .................................................. 46  
4.11 Distribution of Data Points ........................................ 47  
4.12 \( \text{reach-dist}(p_1,o) \) and \( \text{reach-dist}(p_2,o) \), for \( k=4 \) .................................................. 49  
4.13 Outlier Detection with Isolation Forest .......................... 52
5.1 Variation of Openness .......................... 57
5.2 k-Class Activation Vectors ...................... 66
5.3 Weibull Distribution of Two Classes ........... 67
5.4 Activation Vectors of Two Classes .......... 73
LIST OF TABLES

5.1 Dataset Example Reviews .................................................. 55
5.2 Open Set Experiments on 20 Newsgroups Dataset (Tested on 10 classes) using HNN Model .......................................................... 61
5.3 CNN Baseline Configuration ................................................. 62
5.4 Closed Set Evaluation of All Models ...................................... 62
5.5 Preliminary Experiments on Amazon Product Reviews dataset (Tested on 10 and 20 Classes) .................................................. 70
5.6 Preliminary Experiments on 20 Newsgroups Dataset (Tested on 10 and 20 Classes) ............................................................ 71
5.7 Experiments on Amazon Product Reviews Dataset (Tested on 10, 20 and 50 Classes) ........................................................... 74
5.8 Experiments on 20 Newsgroups Dataset (10 and 20 Classes) .......... 75
CHAPTER 1
INTRODUCTION

Text classification is the process of categorizing text into organized groups based on their content. In the context of machine learning, text classification is a supervised learning task that categorizes a text document into one of a predefined set of classes. This categorization may be based primarily on keywords present in the text. Supervised learning discovers a relation between input and output pairs through the use of a labeled dataset. A labeled dataset consists of a set of examples with associated class labels. Figure 1.1 illustrates product reviews categorized into three groups with labels namely shoes, toys and tents based on their content.

Another common type of text classification is sentiment analysis, whose goal is to identify either positive polarity or negative polarity of the text content.
Text classification is a commonly used Natural Language Processing (NLP) task in applications such as automatic news article categorization into well-defined topics and, sentiment analysis. The rapid growth of online information has made text classifiers one of the most important tools for people to track and organize information. As new online text content is continuously being generated by social media, blogs and news articles, it is possible that new online documents may not belong to any of the previously defined or known groups but instead belong to a new or unknown group.

1.1 CLOSED SET AND OPEN SET TEXT CLASSIFICATION

A labeled text corpus or dataset is a group of samples or examples that have been tagged with a “label” or a “class”. We use a labeled dataset for training a classifier. After the classifier is trained with labeled training data, it is ready for testing. Based on the technique adopted for text classification during testing, we categorize it into two categories: closed set text classification and open set text classification.

In closed set text classification, the classifier is tested on unseen examples (during training) that belong to one of the trained classes. For example, let us consider a product review dataset with three classes or labels, namely, *Shoes*, *Toys* and *Tents*. We split our dataset into 70% for the train set and 30% for the test set. Then, we train our classifier with test examples. During testing, we show examples from the test set (unseen during training) and measure how well the classifier is able to identify these test examples. This scenario is called closed set text classification as shown in Figure 1.2.

On the other hand, in open set text classification, the classifier is tested on new examples that do not belong to any of the trained classes. Considering the above example, the classifier has been trained on three product review examples. But during testing, if the classifier
is tested on a say “camera” review example, this new test example does not belong to any of the three classes. This situation of the classifier where new examples from new classes are given during testing to perform text classification is called \textit{open set} text classification. The classifier should be able to create a new class called \textit{unknown} and classify the camera review test example to the \textit{unknown} class. Figure 1.3 shows how open set text classification works.

Most research in text classification is made under the closed set assumption. In real world applications, closed set text classifiers are not ideal to deploy, as new diversified text is added continuously through social media or blogs, which may not necessarily belong to any of the previously known classes. The drawbacks of such a text classifier is that it fails to identify examples of unknown or new classes that were not presented during training and may mis-classify such examples as belonging to one of the known classes. In real world scenarios, a resilient, trained classifier should be able to recognize examples of unknown classes and accordingly update its learned model. This is known as the \textit{open set} approach to text classification. Thus, the goal of an open set classifier is to anticipate and be ready to
handle test examples of unknown classes.

Open set classification is gaining popularity these days and is a part of new machine learning paradigm called Lifelong Machine Learning (LML) [1]. The learning process in humans accumulates and maintains the knowledge learned from previous tasks and uses it seamlessly in learning new tasks and solving new problems. Over time humans learn more and become more knowledgeable, and more effective at learning. Lifelong Machine Learning aims to mimic this human learning process and capability.

Abundant information on the web and extensive sharing of concepts across various domains provide great learning opportunities for LML. According to Chen and Liu [1], LML is defined as a continuous learning process. At any time point, the learner is assumed to have learned to perform a set of N tasks, \( \{T_1, T_2, \ldots, T_N\} \). The performed tasks can be sentiment analysis, text classification into predefined sets, identifying the author of a text and so on. These tasks, which are also called the previous tasks, have their corresponding
datasets \( \{D_1, D_2, \ldots, D_N\} \) on which they were trained. The tasks can be of different types and from different domains. When faced with the \((N + 1)\)th task \( T_{N+1} \) (which is called the new or current task) with its training dataset \( D_{N+1} \), the learner can leverage the past knowledge to help learn \( T_{N+1} \). The objective of LML is usually to optimize the learning of the new task \( T_{N+1} \) based on the past knowledge.

1.2 APPROACH

In this thesis, we create a classification model which, once trained, can identify new examples that do not belong to any known classes and can assign those examples to a new class named \textit{unknown} class. We initiate our research in open set text classification by building a Recurrent Neural Network (RNN) model. The results obtained were not satisfactory. We added additional layers to the existing RNN model and renamed the model Hierarchical Neural Network (HNN). Due to poor results, we added attention mechanism to the existing model and retried all experiments. The model was too slow to train as well as ineffective to produce satisfactory closed set text classification accuracy. In order to achieve state-of-the-art results in identifying the unknown class, we need high closed set text classification accuracy. So, we continued our research by using a Convolutional Neural Network (CNN). They are good at content based classification task and performed better than RNNs, HNNs and HNN with attention networks. We used an ensemble approach to identify new examples during testing. Our open set text classifier with an ensemble approach performed better than the existing classifiers, beating the state-of-the-art in some domains.
1.3 ORGANIZATION OF THE THESIS

The thesis is organized as follows. In Chapter 2, we discuss problems with closed set text classifiers, the need for open set text classifier and the solutions adopted to resolve the problems of close set text classifier. In Chapter 3, we discuss related work in open set classification in both text and image domains. In Chapter 4, we introduce the approaches and techniques we followed to develop open set text classifier. In Chapter 5, we perform extensive experimental evaluation. In Chapter 6, we conclude the thesis with future work.
CHAPTER 2
PROBLEM STATEMENT AND SOLUTION

Classification is the task of learning to select the correct class label for a given input. In open set text classification, we want to be able to classify new examples as belonging to existing classes as well as an unknown class, as appropriate.

Let us represent the training text corpus or dataset \(D\) having \(t\) examples belonging to \(n\) classes or labels as

\[
D = \{(x_1, l_1), (x_2, l_2), \ldots, (x_t, l_t)\}
\]

where \(x_i\) is the \(i\)th document, \(l_i\) is the class label for \(x_i\) and \(i = 1, \ldots, n\).

A closed set classifier model \(C_s(x)\) is trained on examples from \(n\) classes. These \(n\) classes are also referred to as known classes. Then it is tested with previously unseen test examples \(x_{\text{test}}\) that belong to one of the \(n\) known classes. This classifier, classifies \(x_{\text{test}}\) to one of the \(n\) known classes:

\[
C_s(x) = (x_{\text{test}}, l_{\text{pred}})
\]

where \(l_{\text{pred}}\) is one of the \(n\) labels \(\{l_1, l_2, \ldots, l_n\}\)

However, this is not the ideal way to design a classifier targeting real world applications. With the proliferation of the web, there are more and more unique texts added online on a daily basis and they may not belong to any of the previous known classes. If such an \(x_{\text{test}}\) is given to a trained closed set classifier, it will mis-classify \(x_{\text{test}}\) to one of the \(n\) known
classes. This reduces the performance of the model due to poor accuracy.

The solution to avoid mis-classification is to incorporate rejection capability in the closed set classifier. This results in an open set classifier model $O_s(x)$, which can handle test examples from an unknown class appropriately. $O_s(x)$ can classify $x_{test}$ to one of the $n$ classes or reject to indicate that it does not belong to any of those known classes. To accommodate rejection, we need to add an additional class named unknown to the existing list of $n$ classes. Thus we have an open set classifier $O_s(x)$ which is trained with $n$ class labels, but tested with $n+1$ class labels.

$$O_s(x) = (x_{test}, l_{pred})$$ (2.3)

where $l_{pred}$ could be one of the $n+1$ labels $\{l_1, l_2, \ldots, l_n, l_{unknown}\}$

To reduce the amount of mis-classification, we propose a model that is a combination of neural networks for classification and ensemble outlier detection for unknown classes.

We first train the neural network model on a set of examples. Once the model is trained, we introduce new test examples during testing, that do not belong to any of the trained classes. These new test examples behave as outliers. Over the last couple of decades, ensemble systems have gained popularity in the machine learning community. The use of an ensemble of classifiers is a machine learning technique where classification decisions are combined in some way to improve the performance of the overall system. The simplest way to combine decisions is by voting or weighted voting. We use an ensemble of outlier detectors and combine the outlier decisions by voting to improve overall system performance.

Figure 2.1 shows an overview of the model proposed.
2.1 DATASETS AND TECHNIQUES USED

The datasets that we use for our model are 20 Newsgroups [2] that consists of 18,828 documents partitioned (nearly) evenly across 20 mutually exclusive classes and Amazon Product Reviews [3] which consists of 50 classes of products, each with 1,000 review documents each.

We build various neural network models for text classification. The model with highest closed set text classification result is selected as the base model. The base model is trained with examples of 2 classes belonging to the 20 Newsgroups dataset. The test set is prepared by combining examples from 2 trained classes (that are not seen during training) and examples from 8 untrained classes. So the test set contains many examples of classes that are unknown to the classifier. These unknown examples are referred to as outliers. The test set is given to the trained classifier and the output vectors called activation vectors are obtained from the penultimate layer. The activation vectors are further analyzed by three outlier detection models to identify the examples that belong to the unknown class. An ensemble approach with majority voting scheme is proposed to classify the test examples.
Similar experiments are conducted by varying the number of classes in the training set and test set for both datasets. The next chapter discusses related work in open set classification in Computer Vision and Natural Language Processing (NLP) domains.
CHAPTER 3

RELATED WORK

A key assumption made by basic classification algorithms is that examples that appear in the test data must belong to one of the classes in the training, called the closed set assumption [4]. To allow for the possibility that the set of classes is open or expandable during testing or deployment, classification algorithms need to be adaptive. There are some prior approaches for open set classification.

3.1 BACKGROUND

In Computer vision, Scheirer et al. [5] studied the problem of recognizing unseen images that the system was not trained for by reducing open space risk and empirical risk. The basic idea is that a classifier should not cover too much open space where there are few or no training data examples due to the existence of a space in which classification probabilities are not currently known. Empirical risk comes from actual examples being mis-classified by a trained classifier, and the open space risk recognizes the fact that the presence of unknown classes is likely to introduce errors into classification decisions. Their model reduces the risk by introducing parallel hyperplanes, one near the class boundary and another far from it to introduce slabs of subspaces for the classes, and then develops a greedy optimization algorithm that modifies a linear SVM and moves the planes incrementally. This work was extended to multi-class open set classification by introducing what is called a Compact Abating Probability (CAP) model [6]. Scheirer et al. build a classifier called
W-SVM using properties of Extreme Value Theory for calibration of scores produced by 1-class and binary SVMs. Extreme Value Theory (EVT) [7], [8] and [9] is usually used to deal with and predict rare events or values that occur at the tails of distributions. The unnormalized probability of inclusion for each class is estimated by fitting a Weibull distribution [10] over the positive class scores from SVM classifiers. The assumption here is when a trained classifier cannot classify an example as belonging to any of the known classes, it is a case of “failure” of the classifier and is deemed unusual. Jain et al. [11] also use EVT to formulate the open set classification problem as one of modeling positive training data at the decision boundary. They introduce a new algorithm called the $P_i$-SVM for estimating the unnormalized posterior probability of class inclusion. Their approach is different from the one introduced by Platt et al. [12], who took SVM outputs and converting them to probabilities by fitting a sigmoid function to the SVM scores.

Bendale and Boult [13] present an approach to minimize the weighted sum of empirical risk and open set risk using thresholding sums of monotonically decreasing recognition functions, and use their approach to extend the Nearest Centroid Classifier (NCM)[14]. This classifier represents classes by the mean feature vector of its elements. An unseen example is assigned a class with the closest mean. The Nearest Non-Outlier (NNO) algorithm [13] adapts NCM for open set classification, taking into account open space risk and metric learning. The nearest class mean metric learning (NCML) [15] approach extends the NCM technique by replacing the Euclidean distance with a learned low-rank Mahalanobis distance. This gives better results than the former as the algorithm is able to learn features inherent in the training data.

In the context of deep learning, Bendale and Boult [16] adapt a CNN [17] to perform open set classification in the vision domain. In closed set classification, the final softmax
layer of the CNN essentially chooses the output class with the highest probability with respect to all other output labels. Bendale and Boult propose OpenMax, which is a new model layer that estimates the probability of an input belonging to an unknown class instead of softmax. Ge et al. [18] adapt OpenMax to generative adversarial networks (GANs) for open set vision problems.

Recently, in Computer vision, Shu et al. [19] presented a model for unseen class discovery in open world classification that is a combination of two main neural networks: an Open Classification Network (OCN) for seen class classification and unseen class rejection, and a Pairwise Classification Network (PCN) that learns a binary classifier to predict whether two given examples come from the same class or different classes. Their OCN module uses Deep Open Classification (DOC) [20], which was for open text classification, but their experiments showed that it also works well for images and outperforms OpenMax [16].

In the field of Computer vision, Dhamija et al. [21] worked towards improving the ability to classify correct classes while reducing the impact of unknown inputs by training the networks using negative samples from some unknown classes. They derive a novel loss function called the Entropic Open-Set loss, to increase the entropy of the softmax scores for background training samples and to improve the handling of background and unknown inputs. They extend entropic open-set loss to new loss metric called objectosphere loss, to further increase softmax entropy and performance by minimizing the Euclidean length of deep representations of unknown samples. They also proposed a new evaluation metric for comparing the performance of different approaches under the presence of unknown samples, and they showed that the new loss functions advance the state-of-the-art for open set image classification.
3.2 RELATED WORK IN OPEN SET TEXT CLASSIFICATION

All the work mentioned so far have been in the context of computer vision. Work in open set classification in natural language processing for textual data is limited, but important as well. An example of an open set text classification scenario is authorship attribution, where each author happens to be a class. An open set text classifier must recognize the author of a document and classify it to one of the known classes appropriately. Additionally, the classifier should explicitly recognize an unseen document as written by none of the known authors, if it is the case. Open set classification is likely to be helpful in identifying authors for any type of articles or documents, including historical or fictional works from the past or present, or emails, social media posts or leaked political documents. The number of authors and hence the number of classes is potentially very large.

Much research has contributed to the area of statistical authorship attribution with the key notion that measuring textual features enables distinction between texts written by different authors [22]. These features range from indicators of content divergence between authors using bag-of-words to stylometric features that reflect an author’s unique writing patterns, e.g., use of punctuation marks, emoticons, whitespace, etc. [23], and character and word n-grams [24].

One-class SVM [25] and Support Vector Data Description (SVDD) [26] are the earliest approaches for open set text classification. One-class SVM considers the origin in the feature space as the only member of the negative class, and maximizes the margin with respect to it. SVDD tries to place a hypersphere with the minimum radius around almost all the positive training points. It has been shown that the use of Gaussian kernel makes
SVDD and One-class SVM equivalent, and the results reported in Khan et al. [27] demonstrate that SVDD and One-class SVM are comparable when the Gaussian kernel is applied. However, as no negative training data is used, one-class classifiers have trouble producing good separations.

More powerful binary classification models have been proposed specifically for open set visual recognition tasks. We discuss various open set classifiers in computer vision as we use these as baseline for comparing our model results. Scheirer et al. [5, 6] offered a formalization of the risk of the unknown in open set recognition that is used to develop the 1-vs-Set Machine [5] and W-SVM (a calibrated non-linear classifier) [6] algorithms. Jain et al., [11] proposed an algorithm called the $P_i$-SVM to achieve good generalization through some measure of discrimination with known negative classes and to estimate the unnormalized posterior probability of class inclusion.

Dalvi et al. [28] proposed Exploratory Learning in the multiclass semi-supervised learning (SSL) setting. In their work, an “exploratory” version of the expectation-maximization (EM) algorithm is proposed to extend traditional multiclass SSL methods, which deal with the scenario when the algorithm is given examples from only some of the classes in the data. It automatically explores different numbers of new classes in the EM iterations. Fei et al. [4] use Center-Based Similarity (CBS) learning [29] where a document is represented as a vector of similarities from centers of spheres that correspond to individual classes. Around the sphere that represents positive examples of a class, they draw a slightly bigger sphere to provide additional space for a class to accommodate unseen examples. They also use SVM hyperplanes to bound the bigger spheres. The unbounded regions correspond to unknown classes.

The Nearest Centroid Class (NCC) algorithm proposed by Doan et al. [30] builds upon
the NCM, but uses a density-based method following the approach of the clustering algorithm called DBSCAN [31]. Doan et al. represent a class not by a sphere but a set of density-connected regions and also consider the centroids of these regions and not the means. Nearest Centroid Class (NCC) algorithm is based on incremental learning and able to detect unknown class during testing. Prakhya et al. [32] investigated the suitability of widely-acclaimed Convolutional Neural Networks (CNNs) for open set text classification and find that CNNs are good feature-extractors, and hence perform better than existing state-of-the-art open set classifiers in domains where the number of trained classes is small and number of untrained classes is large. Prakhya et al. implemented probabilistic and distance based outlier detection techniques to identify examples from untrained classes. This thesis is the extension of Prakhya et al. work in open set text classification.

Deep Open Classification (DOC) by Shu et al. [20] uses CNN [33] as its base and adds a 1-vs rest final sigmoid layer and Gaussian fitting for classification. DOC adopts the OpenMax layer [13] for open text classification. Their 1-vs-rest layer contains $m$ sigmoid functions for $m$ seen classes. For the $i$-th sigmoid function corresponding to class with label $l_i$, DOC takes all examples with label $l_i$ as positive examples and all the rest examples with label not equal to $l_i$ as negative examples. Shu et al. [20] perform evaluation of their model using 20 Newsgroups and Amazon reviews of 50 classes of products.

We compare our results with Shu et al.[20] as it is the latest work been done in open set classification. The details of our approach is discussed in Chapter 4.

### 3.3 RELATED WORK IN OUTLIER DETECTION

An outlier is an observation point that is distant from other observations. Outlier detection is the process of finding the data points that are considerably dissimilar or inconsistent
Chapter 3. RELATED WORK

with rest of the given data points. These inconsistent data points are referred to as outliers. Classification or clustering methods can produce outliers as a side product of their main task. For example, in classification, mislabeled data can be considered outliers, while in clustering, data that do not belong to any clusters can be considered outliers.

In many contexts, outliers are treated as noise that must be eliminated. But in some applications such as credit card fraud detection, in medical analysis to identify malignant and benign cells, and in many other fields, outliers are of great interest. Similarly, in our open set text classification, outliers themselves are very important in identifying if a new test example belongs to known class or unknown class.

Approaches to outlier detection largely treat it as an unsupervised problem in which examples of outliers are not available during training to learn from. This aspect of outlier detection is more challenging than many other supervised or semi-supervised methods. The unsupervised outlier detection method looks for outliers in an unlabeled data set by assigning a score to each object which reflects its degree of abnormality. These scores are usually computed by comparing each object with objects belonging to its neighborhood. Most of the unsupervised approaches proposed in the data mining literature can be classified as deviation-based [34], distance-based [35], density-based [36] and Multi Granularity Deviation Factor (MDEF)-based [37].

There is significant research conducted in outlier detection for large datasets. A deviation-based outlier detection method identifies outliers by examining the main characteristics of objects in a group. Objects that deviate considerably from the general characteristics of the groups are considered outliers. This approach has been pursued in [34, 38] where the groups are formed in random and the result depends on the selected groups.

Distance based method was originally proposed by Knorr et al. [39]. Variants of the
distance based outlier methods were developed by [40, 41, 42]. Ramaswamy et al. [40] extended distance-based outlier detection algorithm by proposing a method that is based on the distance of a point \( p \) from its \( k \)th nearest neighbor. The distance of the \( k \)th nearest neighbor of point \( p \) is a measure of how much of an outlier point \( p \) is. For example, points with larger values for \( D^k(p) \) have more sparse neighborhoods and are thus typically stronger outliers than points belonging to dense clusters which tends to have lower values of \( D^k(p) \). Here, \( D^k(p) \) denotes the distance of the \( k \)th nearest neighbor of point \( p \). Later, a partition-based algorithm is used to effectively mine outliers. The partition-based algorithm first partitions the input points using a clustering algorithm, and computes lower and upper bounds on \( D^k(p) \) for points in each partition. It then uses this information to identify the partitions that cannot possibly contain the outliers and prunes them. The outliers are then computed from the remaining points, belonging to unpruned partitions in a final phase.

A density based outlier detection method was proposed by Breuning et al. [36]. The density based approach relies on the local outlier factor (LOF) of each object, which depends on the local density of its neighborhood. The neighborhood is defined by the distance of \( k \) nearest neighbor. The LOF compares the density of each object \( o \) of a dataset \( D \) with the density of the \( k \) nearest neighbors of \( o \). A LOF value of approximately 1 indicates that the corresponding object is located within a region of homogeneous density (i.e. a cluster).

Several extensions and refinements of the basic LOF model have been proposed, e.g., a connectivity-based outlier factor (COF) [43] or a spatial local outlier measure (SLOM) [44]. Papadimitriou [45] proposed another local outlier detection method named Local Outlier Integral (LOCI) based on a metric called Multi Granularity Deviation Factor (MDEF). For any given value \( p \), MDEF is a measure of how the neighborhood count of \( p \) compares with that of the values in its sampling neighborhood. A value is considered an outlier, if its
MDEF deviates significantly (more than three standard deviations) from the local averages. When dealing with high-dimensional data, the performance of earlier discussed approaches degrades due to the well known “curse of dimensionality” [46]. When using distance-based algorithms to score outliers, we can see the effect of weakly correlated and irrelevant attributes in the concentration of distances. In high-dimensional space, the data becomes sparse, and all pairs of data points are almost equidistant from one another [47, 48]. Therefore, the outliers no longer appear like an outlier, as they become less distinguishable from one another. To handle this situation, Aggarwal et al., [49] proposed subspace outlier detection to emphasize outliers in a lower-dimensional local subspace of relevant attributes. Isolation forests [50] is a subspace outlier detection.

Clustering is an unsupervised machine learning task of grouping a set of given data points into clusters so that points within each cluster are similar to each other and points from different clusters are dissimilar. Clustering based approaches detect outliers by examining the relationship between data points and the clusters. A data point is considered an outlier if it belongs to a small and remote cluster, or does not belong to any cluster.

There are clustering algorithms that can detect outliers as a by-product of the clustering process [51]. The popularly used clustering algorithms are K-means [52], Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [53], and Expectation-Maximization (EM) Clustering using Gaussian Mixture Models (GMM) [54]. Most of the clustering based algorithms work on the principle that data points that are far away from the centroid are treated as outliers.

The K-means [52] clustering algorithm partitions the data points into \( K \) clusters, where each data point belongs to the cluster with the nearest mean serving as a prototype of the cluster. If the data point lies outside all the clusters, it is an outlier. K-means requires the
user to specify in advance the value of $K$, which is the number of clusters. Determining
the optimum number of clusters is a hard problem and thus requires running the K-means
algorithm a number of times with different $K$ values and selecting the best results for the
particular data set. K-means also fails in cases where the clusters are not circular, again as
a result of using the mean as cluster center.

Unlike K-means, DBSCAN [53] does not require the number of clusters in advance as
a parameter. Rather, it infers the number of clusters based on the data, and it can discover
clusters of arbitrary shape in spatial databases with noise. DBSCAN groups together data
points that are closely packed together, i.e., the points with many nearby neighbors and
marks as outliers the points that lie alone in low-density regions whose nearest neighbors
are too far away.

The above two approaches do not perform well if the clusters overlap or if the data
points are spread out. Gaussian Mixture Models (GMM) with EM algorithm can [54] han-
dle this situation. With GMMs, the data points are assumed to have Gaussian distribution
with two parameters to describe the shape of the clusters: the mean and the standard de-

viation. In order to find the parameters of the Gaussian for each cluster, an optimization
algorithm called Expectation-Maximization (EM) is used. The EM algorithm starts by
selecting the number of clusters (like K-means) and randomly initializing the Gaussian dis-

tribution parameters for each cluster. There are two steps that are repeated iteratively until
convergence namely, E-step or Expectation-step and M-step or Maximization step. In the
E-step, given the Gaussian distributions for each cluster, the probability that each data point
belongs to a particular cluster is computed. The closer a point is to the Gaussians center,
the more likely it belongs to that cluster. In M-step, based on the probabilities, a new set
of parameters for the Gaussian distributions are computed such that the probabilities of
data points within the clusters are maximized. These new parameters are computed using a weighted sum of the data point positions, where the weights are the probabilities of the data point belonging in that particular cluster.

In general, the clustering approach for outlier detection is suitable for datasets with compact clusters that are well separated. In our dataset, the examples from trained and untrained classes do not form well separated clusters and hence, may not be an appropriate approach for outlier detection.

3.4 RELATED WORK IN OUTLIER ENSEMBLE

We can combine the outputs of multiple models in order to improve the robustness of our classifier. This phenomenon is referred to as creating an ensemble. An appropriate combination of multiple outlier detection algorithms can increase outlier detection performance [55] which has recently triggered a wide interest in outlier ensembles [56, 57, 58]. In data mining problems, for example, common ensemble methods in classification include bagging, boosting and stacking [59, 60, 61]. Similarly, to improve the quality of the clustering, ensemble methods are often used [60]. In recent years, there is significant theoretical advancements been made in the field of outlier ensembles [57].

Aggarwat et al. [56] proposed that outlier ensemble can be categorized into two types.

- Sequential ensembles, and
- Independent ensembles.

In sequential ensembles, a set of outlier detection algorithms or models are applied sequentially. The final result is either a weighted combination of, or the final result of
the last application of an outlier detection algorithm. In the context of the classification problem, boosting [60] is an example of sequential ensembles.

In independent ensembles, different outlier detection algorithms or models are applied to the entire dataset or portions of the dataset. The output from the different models are combined together in order to achieve high outliers. The main idea of independent ensembles is that every model is specialized for certain characteristics of observations and therefore, might be a good idea to integrate different outlier detection results. The final result of the independent ensembles is the combination of individual outlier detection results through majority voting system. One or another result about a test sample might be wrong, but the majority might still be right.

The first approach to improve outlier detection by ensemble techniques was feature bagging [62], combining different results of the same algorithm, namely LOF [36] applied to different feature subsets. In the feature bagging approach, every outlier detection algorithm uses a small subset of features that are randomly selected from the original feature set. As a result, each outlier detector identifies different outliers, and then assigns outlier scores (i.e., probability of being outliers) to all the data points. The outlier scores computed by the individual outlier detection algorithms are then combined in order to find the better quality outliers.

He et al. [63] proposed an algorithm called SOE1 (Subspace Outlier Ensemble using 1-dimensional subspaces) and built an unified framework for outlier detection in high dimensional spaces from an ensemble learning viewpoint. Subsequent research on outlier detection by ensemble was proposed by Zimek et al. [58] that focused on the fundamental aspects for success in building ensembles for unsupervised outlier detection. Their studies
highlight three important aspects for outlier ensembles: assessment of diversity, normalization of scores, and combination procedures. Zimek et al. [64] proposed data perturbation (process of introducing noise in dataset) as a means to induce diversity among individual outlier models and to build an ensemble with suitable combinations of the resulting outlier scores.

Recently, Kirner et al. [65] focused on different approximation techniques such as space-filling curves, Locality Sensitive Hashing to identify low density regions [66] and NN-Descent for iterative refinement of neighborhoods [67] in building outlier ensembles. They used LOF [36] as the basic outlier detection technique. Outlier scores computed on various approximations are then combined with standard procedures [58], using score normalization [68] and ranking of average scores.

3.5 SUMMARY

In this chapter, we discussed about various open set classification techniques implemented in both Computer vision and NLP. In this thesis we are implementing neural network based text classification as neural networks are gaining popularity and achieving state-of-art-results. This thesis adopts an ensemble approach of outlier detection. We use the independent ensemble technique to detect whether a new test example belongs to a known class or unknown class. We use three different methods for diversity, namely density based, distance based and high dimensionality based methods to identify outliers and finally combine the results using majority voting. The details of our approach is discussed in Chapter 4.
CHAPTER 4
APPROACHES AND ARCHITECTURES

There are various machine learning models that can be used to implement text classification. Some of the most popular traditional machine learning models for text classification are Support Vector Machine (SVM) [69], Naive Bayes [70, 71], k-Nearest Neighbors [72], Bagging and Boosting models [73]. We need a high performance text classification model for our open set text classifier. Neural Networks are proving to be remarkable at text classification, achieving state-of-the-art results on a wide range of NLP tasks [20, 33, 74, 75, 76, 77, 78, 79].

The approach adopted in this thesis for open set text classification is a model that is a combination of neural networks for classification and ensemble outlier detection for unknown classes. Fig 4.1 shows a block diagram of the open set text classifier. This chapter discusses the various neural networks used for text classification and different outlier detection models put together in to an ensemble model to classify the test examples as belonging to known class or unknown class. Good performance of open set text classifier relies on its closed set validation. In machine learning, model validation is the process where a trained model is evaluated with a testing dataset. The testing dataset in closed set validation contains examples belonging to known classes.

We build various neural network models for closed set text classification and analyze their performance after training. Yin et al. [80] claim that RNNs and their variants perform better in text classification. We tried using variants of RNN for closed set text classification. After finding the best closed set text classifier, we make it our base model for open set text
classification. We train our classifier with text corpora. During testing, we introduce new samples in our test set that do not belong to any of the trained classes. We obtain activation vectors (AV) from the penultimate layer, and feed to outlier detection models. The AVs represent the actual magnitude of activation for the various output or class nodes. Many neural network models use a sigmoid layer on top of this layer to convert the class activation magnitudes to pseudo-probabilities. We forgo the use of this sigmoid layer, and replace it with an ensemble of outlier detection approaches. The outlier detection models that we use are Mahalanobis Weibull, Local Outlier Factor (LOF) and, Isolation Forest discussed in sections 4.6.3, 4.6.4 and 4.6.5 respectively. We combine the predictions of each model using a voting scheme. Finally, we predict if the test sample belongs to known class or the unknown class.

The following section gives a detailed description of each neural network model we use.
4.1 RECURRENT NEURAL NETWORK (RNN)

Recurrent Neural Networks (RNNs) are a class of Artificial Neural Networks that are useful in learning sequential data and have shown promising results in many NLP tasks. A Recurrent Neural Network (RNN) [81] has recurrent connections, which behaves like memory. This makes them applicable for sequential prediction tasks. In RNNs, the hidden layer $h_t$ at time step $t$ is computed from a nonlinear transformation of the current input layer $x_t$ and the previous hidden layer $h_{t-1}$. The final output $y_t$ is computed using the hidden layer $h_t$. One can interpret $h_t$ as an intermediate representation summarizing the past, which is used to make a final decision on the current input. Thus, $h_t$ is a representation of a sequence seen up to time $t$. RNNs are called recurrent because they perform the same task on every word of the input sequence, generating an output that is dependent on previous computations. At time $t = 1$, i.e., when first word $x_1$ is encountered, the initial hidden state $h_0$ can be considered as a zero vector indicating no prior knowledge.

The activation of the hidden state $h_t$ at the time-step $t$ is evaluated as a function $f$ of the previous hidden state $h_{t-1}$ and the current input symbol $x_t$. Here, $f$ is a nonlinear function such as the sigmoid function.

$$h_t = \begin{cases} 
0 & \text{if } t = 0 \\
 f(h_{t-1}, x_t) & \text{otherwise} \end{cases} \quad (4.1)$$

Figure 4.2 shows a single RNN cell.

At each time step $t$, the input word $x_t$ represented by its word embeddings is multiplied by input weight $w_x$ to produce the input to the hidden state. The hidden state $h_t$ is calculated
based on the previous hidden state $h_{t-1}$ and the input at the current step as

$$h_t = \sigma(w_x x_t + w_h h_{t-1}).$$  \hspace{1cm} (4.2)

where $\sigma$ is the sigmoid function and $w_h$ is the weight of the hidden state. The sigmoid function outputs a number between 0 and 1, given an input. Depending on the application requirements, we can calculate the output at each time step or at the end of the last word in the sentence.

The final output $y_t$ is computed as

$$y_t = \text{Softmax}(w_y h_t).$$  \hspace{1cm} (4.3)

Recurrent neural networks, when unfolded, are equivalent to feed forward neural networks with as many hidden layers as the number of tokens in the input sequence, with
shared connections across multiple layers of time. In text classification, an RNN maps the input sequence to a fixed-sized vector and then feeds the vector to a softmax layer to obtain vector of probabilities as shown in Figure 4.3.

The backpropagation algorithm applied to an RNN during training is called back propagation through time (BPTT) [82]. In this technique, the gradient at each output depends on the calculations of the current time step and previous time steps. RNNs trained with BPTT have difficulty learning long-term dependencies, which occur when the sequence is very long and can result in vanishing gradients [83]. To address the vanishing gradients problem of long term dependencies, Long Short-term Memory (LSTM) was proposed by Hochreiter et al. [83].

4.1.1 LSTM

LSTMs are an improvement to Recurrent Neural Networks and are very effective at capturing long-term dependencies. An LSTM has, inside it, a memory cell that takes as input the previous state \( h_{t-1} \) and current input \( x_t \).
As shown in Figure 4.4, an LSTM block has a memory cell $C_t$, an input gate $I_t$, an output gate $O_t$ and a forget gate $F_t$. All these gates have their own set of weight values. The LSTM forget gate controls how much previous information should be erased from the cell, the input gate controls how much information should be added to the cell, and the output gate controls the exposure of the internal memory state.

An LSTM calculates hidden state $h_t$ through the gating mechanism, following 4 steps.

- The Forget gate ($F_t$) takes $x_t$ and $h_{t-1}$ as inputs, and applies the sigmoid activation function as per equation:

$$F_t = \sigma(w_{xf} x_t + w_{hf} h_{t-1} + b_f).$$  \hspace{1cm} (4.4)

- The Input gate ($I_t$) decides what new information to store in the cell state, a decision
which is vital to an LSTM. It takes $x_t$ and $h_{t-1}$ as inputs and has two calculations viz., deciding what value to update through the sigmoid function, and generating new candidate cell value $\tilde{c}$ through $tanh$ function as per equations:

$$I_t = \sigma(w_{xi}x_t + w_{hi}h_{t-1} + b_i), \quad (4.5)$$

$$\tilde{c}_t = tanh(w_{xc}x_t + w_{hc}h_{t-1} + b_c). \quad (4.6)$$

We combine these two parts to finally update the previous cell state $c_{t-1}$ to the new cell state $c_t$ as per equation:

$$C_t = f_t * c_{t-1} + i_t * \tilde{c}_t. \quad (4.7)$$

• The Output gate ($o_t$) decides how much of the cell state $C_t$ contributes to the output as the new hidden state $h_t$, using the sigmoid function as per the equation:

$$O_t = \sigma(w_{xo}x_t + w_{ho}h_{t-1} + b_0), \quad (4.8)$$

and finally the new hidden state $h_t$ is obtained by

$$h_t = O_t * tanh(c_t). \quad (4.9)$$

In all the above equations, $b_f$, $b_i$ and $b_o$ are the biases.

A problem with using a single LSTM layer is that it learns representations only from previous time steps. In some applications, to understand the context of a sentence better,
we need to look ahead to learn representations from future time steps also. So, we use Bidirectional LSTM layers which are two independent LSTM layers stacked together.

4.2 BIDIRECTIONAL LSTM (BILSTM)

In a BiLSTM, as the name suggests, there are two types of connections, one going forward in time to learn from previous representations, and another going backwards in time to learn from future representations. Thus, the input to the first LSTM layer is fed in normal time order, and the input to the second LSTM layer is a reversed copy of the input sequence. This provides additional context to the network and results in faster and even fuller learning. The outputs of the two networks are usually concatenated at each time step.

We build one of our text classification models using bidirectional LSTMs as shown in Figure 4.5. We use Keras [85] to build this text classification model. Keras provides a Bidirectional layer wrapper to implement Bidirectional LSTMs. Given an input sequence $S = x_1, x_2, x_3, ..., x_n$, where $n$ is the length of input text, the embedding layer of the model embeds words or tokens into vectors. The next layer, the Bidirectional LSTM processes these embedded words into hidden vectors by processing the data in both directions with two separate hidden layers, which are then fed forward to the same output layer. A BiLSTM computes the forward hidden sequence $A$, the backward hidden sequence $A'$ and the output sequence $y$ by iterating the backward layer from $t = n$ to 1, the forward layer from $t = 1$ to $n$, and then generating the output $y_0, y_1, y_2 ..., y_n$ by concatenating forward and backward outputs. The output from BiLSTM is fed to a dense output layer for classification.
The Hierarchical Neural Network (HNN) model for text classification is an extension to the previous RNN model. The HNN model has two sub-models: an encoder model and a decoder model. The encoder model reads the entire input sequence and encodes it into an internal representation, often a fixed-length vector. The encoder model, which is built using an embedding layer and Bidirectional LSTM layers. The encoder output data is then passed to a Keras layer called TimeDistributed, which is a part of the decoder model.

The decoder model is built using a TimeDistributed layer, a BiLSTM layer and a final Dense output layer. The TimeDistributed wrapper layer allows us to apply a layer to every temporal slice of an input. The output from the TimeDistributed layer is fed to a BiLSTM
layer. Finally, the output layer has a softmax activation applied to it. Keras computes error by comparing the output with the training input data for each batch and backpropagates gradients. Both the encoder and the decoder sub-models are trained jointly, meaning at the same time. Figure 4.6 shows the HNN model.

**Figure 4.6**: Hierarchical Neural Network for Classification
4.4 HIERARCHICAL NEURAL NETWORK WITH ATTENTION (HNNA)

Attention mechanisms in neural networks [86] are gaining in popularity these days, and we tried our HNN model with attention. As we know, not every word in a sentence contributes equally to the representation of the sentence’s meaning. For example, consider a sentence, “The movie we saw yesterday was amazing”. The single word *amazing* in the sentence contributes for positive sentiment of the movie. The attention network aims to recognize such source words that are important to the meaning of the sentence, giving high attention weights to these words in computing the sentence vector.

Once sentence vectors are obtained, a document vector can be obtained in a similar way. In this architecture, Bidirectional GRUs (Gated Recurrent Unit) are used to integrate meanings of words by summarizing information from both directions. This incorporates contextual information from both sides in the encoded representation. The outputs from the bidirectional GRU layers are fed into a learnable function (the attention layer) to produce a probability vector $\alpha$. A sentence vector $s$ is computed as a weighted average of $h_T$, with weighting given by $\alpha$. The sentence vectors become input to the next layer and finally we obtain the document vector.

A GRU [87] is similar to an LSTM cell as they both use gating mechanisms to resolve the long term dependency problem. The gating units of a GRU modulate the flow of information inside the unit, without having a separate memory cell. The GRU was designed to adaptively reset or update its memory content. Accordingly, a GRU has two types of gates as opposed to three gates in LSTMs: the reset gate $r_t$ and the update gate $z_t$. These two gates work together to control the information to be updated to the state. GRUs are simpler and train faster than LSTMs.
The overall architecture of the Hierarchical Neural Network with an associated Attention Network (HNNA) is shown in Figure 4.7.

It consists of four parts: a word sequence encoder, a word-level attention layer, a sentence encoder and a sentence-level attention layer.
4.4.1 WORD SEQUENCE ENCODER

The words \( w_{it} \) represents \( ith \) word at time slice \( t \) in a sentence are embedded to vectors through an embedding matrix. The embedded words are given to a bidirectional GRU layer which reads the words in forward and backward directions. The forward GRU reads the input word sequence as it is ordered (from \( w_1 \) to \( w_T \)) and calculates a sequence of forward hidden states. The backward GRU reads the words in reverse order (from \( w_T \) to \( w_1 \)) generating a sequence of backward hidden states.

The output from the bidirectional GRU represents encoded words obtained by concatenating forward hidden state and backward hidden state outputs. Thus, the word \( w_{it} \) in a sentence at time slice \( t \) contains a robust encoding of the entire sentence.

4.4.2 WORD-LEVEL ATTENTION LAYER

After we obtain the contextualized word encoding, we introduce a word level attention mechanism to extract such words that contribute most to the sentence meaning. We perform summation on these informative words to get a sentence encoding or sentence vector. The output \( u_{it} \) from the bidirectional GRU is evaluated using the equation

\[
u_{it} = \tanh(W_wh_{it} + b_w).
\] (4.10)

In the attention mechanism, we first feed the contextualized word representations \( h_{it} \) through a one-layer MLP to get a hidden representation for each word \( u_{it} \). Then, we measure the importance of this word with a word level context vector \( u_w \) (\( u_w \) is initialized randomly and collectively learned during the training process) and apply a softmax function to get a normalized weight \( \alpha_{it} \) as per equation
\[
\alpha_{it} = \frac{\exp(u_{it}u_w)}{\sum_{t=1}^{T} \exp(u_{it}u_w)}. \tag{4.11}
\]

After that, we compute the sentence vector \( s_i \) as a weighted sum of the word annotations based on the weights as per equation

\[
s_i = \sum_{t=1}^{T} \exp(\alpha_{it}h_{it}). \tag{4.12}
\]

### 4.4.3 SENTENCE ENCODER

Once the sentence vectors \( s_i \) are obtained, we implement a similar procedure to obtain a document vector \( v \) by adding a sentence encoder on top of the word encoder. We use a bidirectional GRU to encode the sentences in both directions, and concatenate forward and backward hidden state outputs to get a final summarized encoding of a sentence.

### 4.4.4 SENTENCE-LEVEL ATTENTION LAYER

Similar to the word level attention mechanism, we introduce a sentence level attention mechanism. We introduce a sentence level context vector \( u_s \) (\( u_s \) is initialized randomly and collectively learned during the training process), to measure the importance of a sentence. Finally, we compute a document vector \( (v) \) that represents a summary of the document by performing following computations

\[
u_i = \tanh(w_sh_i + b_s), \tag{4.13}\]
\[ \alpha_i = \frac{\exp(u_i u_s)}{\sum_{t=1}^{n} \exp(u_i u_s)}, \tag{4.14} \]

\[ v = \sum_{t=1}^{T} \exp(\alpha_i h_i). \tag{4.15} \]

The accuracies obtained for closed set classification using variants of the RNN model are low and hence not suitable for open set classification. RNN variants are good at modeling units in sequence such as question-answering, and translation and a CNN is supposed to be good at extracting position invariant features for used in tasks such as searching for named entities, and terms representing anger or sadness.

The model presented by Kim et al. [33] achieves strong classification performance across a range of text classification tasks like sentiment analysis. The next section discusses the CNN model for closed set classification.

### 4.5 CONVOLUTIONAL NEURAL NETWORK (CNN)

CNNs represent a class of deep, feed-forward artificial neural networks that are in essence feature-extracting architectures. Since Prakhya et al. [32] and Bendale et al. [16] have shown that CNNs work well in open set classification, we use their approach as the basis and extend the work as necessary.

The input to most NLP tasks are tokenized sentences or documents, which are then represented as a matrix of vectors. Each row of the matrix corresponds to one token, typically a word, but it could be a character. Each token is then represented by a vector using word embeddings like Word2Vec [88] or GloVe [89], but the tokens could also be one-hot vectors that index the word into a vocabulary. If the length of a given sentence is \( s \), and the
Figure 4.8: Stages of CNN Architecture in Text Classification (Adapted from [79]).
dimensionality of the word vectors is $d$ then the dimensionality of the sentence matrix is $s \times d$.

One of the coveted properties of CNNs in computer vision is that they preserve 2D spatial orientation. Like images, texts too have an orientation. Instead of 2-dimension, texts have a one-dimensional structure where the sequence of the words matters. The tokenized sentences are each replaced by a $d$-dimensional word vector. We perform the convolution operation using a filter and a window of $n$ words to extract features from the input. A filter matrix $w$ is applied to a region size $h$ (i.e., height) to produce a new feature $c_i$ given by the equation

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

(4.16)

where $f$ is a non-linear activation function, $b$ is a bias and $x_{i:i+h-1}$ represents window of words between $i : i + h - 1$.

This filter is repeatedly applied to sub matrices of the sentence matrix to produce a feature map

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

(4.17)

where $n$ represents the length of the sentence. We can use multiple filters with varying window sizes or multiple filters with the same region size to obtain complementary features. We then commonly apply 1-max pooling [90] over each feature map $c$ to extract the maximum value. 1-max Pooling is a down sampling strategy in Convolutional Neural Networks, which selects the maximum value of the feature map:

$$\hat{c} = \max\{c\}.$$  

(4.18)
The main idea behind max pooling is to capture the most important feature for each feature map. The outputs generated from each filter map is concatenated into a fixed-length feature vector. These features form the penultimate layer which is then fed to a fully connected softmax layer to generate the final classification. While training the model, we can choose hyperparameters like different optimizers, activation functions, learning rate, and L2 regularization to improve accuracy of the model.

Figure 4.8, taken from Zhang et al. [79], illustrates the model just described by taking a 7 word input sentence with 5-dimensional embedding as an example. The exclamation mark in the input sentence is considered a word, but some authors deal with it differently. We now have a sentence matrix of the shape 7 x 5. For convolutions, two filters each for three region sizes: 2, 3 and, 4 is used. The CNN performs convolutions to get two feature maps for each region size. Then, max pooling extracts the maximum value from each feature map, which is then concatenated to obtain one fixed length vector. This fixed length vector is passed through the fully connected dense layer with the final softmax layer to get the probability distribution over 2 class labels.

Kim et al. [33] help us arrive at a simple but effective neural network architecture for text classification. Thus, we perform experiments with a single-layer CNN, using a modified final layer instead of softmax. Figure 4.9 shows the CNN architecture we implemented in this thesis.

4.6 ENSEMBLE OUTLIER DETECTION MODELS

Over the last couple of decades, ensemble systems have gained popularity in the machine learning community. The use of an ensemble of classifiers is a machine learning technique where classification decisions are combined in some way to improve the performance of the
overall system. The simplest way to combine decisions is by voting or weighted voting. As discussed in chapter 3, section 3.4, we adopt the ensemble approach for outlier detection. We assume that new examples that do not belong to any of the known classes behave like outliers considering the known classes.

### 4.6.1 OUTLIERS

Usually, outliers are extreme values that deviate from other observations, and they may indicate experimental errors or novelty. In other words, an outlier is an observation point that is distant from other observations. Outlier detection is the process of finding the data points that are considerably dissimilar or inconsistent with rest of the given data points. These inconsistent data points are referred to as outliers. In open set classification, the test set consists of examples from trained and untrained classes. The examples from untrained classes are considered as outliers. Hence, the outlier detection techniques are implemented to identify these outliers. Outliers can be of two types, namely univariate and multivariate.
Univariate outliers can be found in a distribution of values in a single feature space. Multivariate outliers can be found in a $n$-dimensional space of $n$ features. It is very difficult for humans to look at distributions in $n$-dimensional space. Hence, we need to train a model to identify multivariate outliers.

### 4.6.2 APPROACHES FOR OUTLIER DETECTION

There are various approaches to find outliers, namely based on the effort to separate an outlier from non-outlier, the lack of a dense region of neighbors, etc.

#### 4.6.2.1 DISTANCE BASED APPROACH

Knorr and Ng [39] were the first to formalize a distance-based outlier detection scheme. The distance-based approach provides particular advantage when the data does not fit into any standard distribution model [91], and it can still discover outliers effectively in the multi-dimensional case. The most popular distance-based methods is k-Nearest Neighbors [92]. We can use either Euclidean [93] or Mahalanobis distance [94] to calculate the distance between a data point and its $k$th nearest neighbor. We use Mahalanobis distance in this thesis which is discussed in Section 4.6.3.

#### 4.6.2.2 DENSITY BASED APPROACH

A density-based approach [91] introduces a new outlier representation called a local outlier, to investigate the degree of an object to be an outlier considering the local neighborhood density. This degree is also identified as a local outlier factor (LOF) [36] associated with each data object. Since distance-based methods may be difficult in dealing with data points
in different density areas, density-based method could be applied to handle such kind of data points. Three common density-based methods are LOF, Influenced Outlierness (INFLO) [95] and Local outlier correlation integral (LOCI) [45]. We use LOF for outlier detection in this thesis which is discussed in Section 4.6.4.

4.6.2.3 HIGH DIMENSIONAL APPROACH

When dealing with high-dimensional data, the performance of earlier discussed approaches degrades due to the well known “curse of dimensionality” [46]. When using distance-based algorithms to score outliers, we can see the effect of weakly correlated and irrelevant attributes in the concentration of distances. In high-dimensional space, the data becomes sparse, and all pairs of data points are almost equidistant from one another [47, 48]. Therefore, the outliers no longer appear like outliers, as they become less distinguishable from one another. To handle this situation, Aggarwal et al. [49] proposed subspace outlier detection to emphasize outliers in a lower-dimensional local subspace of relevant attributes. Isolation forests [50] is a subspace outlier detection. We use Isolation Forest for outlier detection, which is discussed in Section 4.6.5.

4.6.3 MAHALANOBIS DISTANCE (DISTANCE BASED APPROACH)

The Mahalanobis Distance (MD) [94] is a way to compute distance between two points when the features or dimensions used to represent the points may be correlated. MD is a measure of the distance between a point \( x \) and a distribution \( D \). It measures the number of standard deviations between point \( x \) and the mean of \( D \).

The commonly used Euclidean distance between two points assumes that the features used are at right angles to each other, or that the features or dimensions are uncorrelated.
However, if the features are correlated, they are no longer at right angles, and in such situations MD measure can be used to compute distances between points. The Mahalanobis method is robust to increasing dimensionality, because it uses the covariance matrix in order to summarize the high dimensional deviations in a statistically effective way. A common use for the Mahalanobis distance is for identifying outliers.

We use Mahalanobis Distance as a distance metric to compute how much an example deviates from the class mean. The Mahalanobis distance precisely does this by giving us a multi-dimensional generalization of the number of standard deviations a point is from the distribution’s mean. The closer an example is to the distribution mean, the lower is the Mahalanobis distance. Mahalanobis distance can be used to compute the distance between two points also. The Mahalanobis distance between point (object) $\vec{x}$ and point (object) $\vec{y}$ is given by:

$$d(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T C^{-1} (\vec{x} - \vec{y})}$$

(4.19)

where $C$ is the covariance matrix, among the features used to represent $\vec{x}$ and $\vec{y}$ calculated a priori.

### 4.6.3.1 WEIBULL MODEL

The Weibull Distribution [96, 97] is a continuous probability distribution that is widely used in life data analysis. In Reliability Theory, Weibull models are used to describe various types of observed failures of components and phenomena. The formula for the two parameter Weibull Distribution is often used in failure analysis and is given by:
Chapter 4. APPROACHES AND ARCHITECTURES

\[ f(x; \lambda, k) = \frac{k}{\lambda} \left( \frac{x}{\lambda} \right)^{k-1} e^{-\left( \frac{x}{\lambda} \right)^{k}} \text{ for } x \geq 0 \] (4.20)

where \( k \) is the shape parameter that determines the failure rate, and \( \lambda \), also called the characteristic life parameter, is the scale parameter of the distribution. We use 2-parameter Weibull distribution in our experiments.

In this thesis, we compute the Mahalanobis distance of each data point with respect to its class mean. The probabilities are calculated using the Weibull model for all the distances obtained and then plotted to obtain the Weibull Distribution. The data points that lie close to the class mean have low Mahalanobis distance and high probability values. Similarly, the data points that are far away from the class mean have high Mahalanobis distance and low probability values. These probabilities are used to infer whether a data point belongs to a class or fails to belong to a class. As a data point drifts away from the class mean, it fails to belong to that particular class and behaves like an outlier. Figure 4.10 shows the Weibull Distribution for different values of \( \lambda \) and \( k \). We choose Weibull Distribution over other distributions, as this distribution best suits to identify the data points that fail to belong to a particular class.

![Weibull Distribution](image)
Distance based approaches have problems finding an outlier like point $O_1$ as shown in Figure 4.11, because points in cluster $C_2$ are less dense compared to points in cluster $C_1$. If we select a large threshold to capture an outlier like $O_1$, many of the points in $C_2$ will be counted as outliers. Cluster based approaches have similar problems, because they only consider the distance between a point and the centroid of a cluster to calculate the outlier score. The density based approaches, especially the Local Outlier Factor approach discussed in Section 4.6.4 are sensitive to densities and more appropriate for detecting outliers in such situations.

![Distribution of Data Points](image)

**Figure 4.11: Distribution of Data Points**

### 4.6.4 LOCAL OUTLIER FACTOR (DENSITY BASED APPROACH)

The local outlier factor (LOF) is a score computed by an unsupervised density based algorithm proposed by Breunig et al. [36] that relies on nearest neighbor search. It computes the local density deviation of a given data point in a multidimensional dataset with respect
to its neighbors. It considers an unseen sample as outlier if it has a substantially lower density than its neighbors.

4.6.4.1 LOF ALGORITHM

The algorithm for computing the LOF score of a data point \( p \) in a dataset \( D \) has several steps. Let \( D \) be a dataset; \( p \) and \( o \) be some data points in \( D \); and \( k \) be any positive integer.

- Computing \( k \)-distance of \( p \): For each data point \( p \), we compute the distance to its \( k \)-th nearest neighbor. We name it \( k \)-distance\((p)\). The data points that lie within \( k \)-distance\((p)\) to data point \( p \) are said to be in its \( k \)-distance neighborhood. \( k \)-distance\((p)\), provides a measure of the density around the data point \( p \). When \( k \)-distance\((p)\) is small, the area around \( p \) is dense and when \( k \)-distance\((p)\) is large, the area around \( p \) is sparse.

- Finding \( k \)-distance neighborhood of \( p \): The \( k \)-distance neighborhood of \( p \) contains every data point where the distance for \( p \) is not greater than the \( k \)-distance. In other words, we compute the maximum distance up to the \( k \)-th nearest point. For example, for \( k = 3 \), if we find the 3 nearest neighbors to have distances of 1.5, 2.3 and 4.7, the \( k \)-distance for this point will be 4.7.

- Computing the reachability distance of \( p \) with respect to the data point \( o \): The reachability distance of data point \( p \) with respect to data point \( o \) is defined as:

\[
reach-dist_k(p, o) = \max\{k\text{-distance}(o), d(p, o)\}. 
\] (4.21)
We choose a parameter $MinPts$ specifying the minimum number of data points $(N_{MinPts})$ to define the notion of density, and use the values $reach-dist_{MinPts}(p,o)$, for $o \in N_{MinPts}(p)$, as a measure to determine the density in the neighborhood of a data point $p$.

- Computing local reachability density of a data point $p$:

\[
\text{lrd}_{MinPts}(p) = \left[ \frac{\sum_{o \in N_{MinPts}(p)} \text{reach} - \text{dist}_{MinPts}(p,o)}{|N_{MinPts}(p)|} \right]^{-1}.
\]  

(4.22)

Figure 4.12 illustrates the idea of reachability distance with $k = 4$. If the data point $p$ is far away from data point $o$ (point $p_2$ in the Figure 4.12), the reachability distance between the two points is simply their actual distance. However, if they are close (point $p_1$ in the Figure 4.12), the actual distance is replaced by the $k$-distance of $o$. This is done to reduce the statistical fluctuations of $d(p,o)$ for all the $p$’s close to $o$.

The local reachability density of a data point $p$ is the inverse of the average reachability distance from the $MinPts$-nearest neighbors of $p$:
Essentially, the local reachability density of a data point \( p \) is an estimate of the density at point \( p \) by analyzing the \( \text{MinPts} \)-distance of the data points in \( N_{\text{MinPts}}(p) \). Based on local reachability density, the local outlier factor score can be computed as shown in the next step.

- Computing the local outlier factor score of \( p \): The local outlier factor is a ratio that determines whether or not a data point is an outlier with respect to its neighborhood. \( \text{LOF}(p) \) is the average of the ratios of the local reachability density of \( p \) and that of \( p \)'s \( \text{MinPts} \)-nearest neighbors.

\[
\text{LOF}_{\text{MinPts}}(p) = \frac{\sum_{o \in N_{\text{MinPts}}(p)} \frac{lrd_{\text{MinPts}}(o)}{lrd_{\text{MinPts}}(p)}}{|N_{\text{MinPts}}(p)|} \tag{4.23}
\]

The LOF score generated for regular points are close to 1 indicating that the data point is in a comparably dense area just like its neighbors, and the score for outliers will be far from 1.

In this way, LOF is able to capture both outliers points (\( o_1 \) and \( o_2 \) in Figure 4.11) due to the fact that it considers the density around the points.

### 4.6.5 ISOLATION FOREST (HIGH DIMENSIONAL APPROACH)

Generally, anomalous examples are few in number and different from non-anomalous or normal examples. This makes anomalies more susceptible to isolation from other examples in space than normal points. Liu et al. [50] showed that a tree structure can be constructed effectively to isolate every single instance in a dataset. Because of their susceptibility to isolation, anomalies are isolated closer to the root of the tree, whereas normal points are isolated at the deeper end of the tree. Liu et al. [50] called this tree Isolation Tree or iTree.
Chapter 4. APPROACHES AND ARCHITECTURES

The Isolation Forest or iForest algorithm, builds an ensemble of iTrees for a given data set. Anomalies are those instances which have short average path lengths on the iTrees.

The Isolation Forest algorithm detects outliers using a set of isolation trees. The Isolation Forest algorithm recursively partitions the data at random partition points with randomly chosen features. Doing so isolates instances into nodes containing only one instance. The heights of branches containing outliers are comparatively low compared to the heights for other data points. The height of the branch is used as the outlier score. The final combination step is performed by averaging the path lengths of the data points in the different trees of the isolation forest.

One important challenge in anomaly detection comes up when dealing with high dimensional data. For distance-based methods, every point is equally sparse in a high dimensional space, thus possibly making distance a useless metric [47, 48]. The Isolation Forest algorithm achieves high anomaly detection performance quickly with high dimensional data.

4.6.5.1 ISOLATION FOREST ALGORITHM

Let us consider a dataset $D$ containing $N$ points. The initial step is to create a root node of the Isolation tree $T$ with $N$ points. A candidate list $C$ of nodes is initialized as a single list containing the root node. Then, the following steps are repeated to create the isolation tree $T$ until the candidate list $C$ is empty:

- Randomly select a node $S$ from $C$ and remove from $C$ list.

- Select a random attribute $i$ and split the data in $S$ into two sets $S_1$ and $S_2$ at a random value $v$ chosen along that attribute. All data points in $S_1$ satisfy $x_i \leq v$ and all data points in $S_2$ satisfy $x_i > v$. The value $v$ is chosen uniformly at random between the
maximum and minimum values of the $ith$ attribute among the data points in node $S$.
The nodes $S_1$ and $S_2$ are children of $S$ in $T$.

- If $S_1$ and $S_2$ contain more than one point, add it to $C$ and repeat previous step. Otherwise, assign the node as an Isolation tree leaf.

The outlier nodes tend to be isolated more quickly than non-outlier nodes. The distance of the path from the root to the leaf is averaged and normalized among all the trees to calculate the anomaly or outlier score. An anomaly score of 1 is considered an outlier, and a score close to 0 is considered normal. Figure 4.13 shows an outlier $X_0$ detected, using the Isolation Forest algorithm.
4.7 SUMMARY

This chapter has discussed in detail neural network models for text classification and various approaches for outlier detection. We use a neural network for classification, followed by an ensemble approach in our combined model for robust outlier detection. We implement the models presented in this chapter for closed set and open set text classification. The experimental procedure and result analysis are discussed in detail in Chapter 5.
As discussed in Chapter 4, the approach adopted in this thesis for open set text classification is a model that is a combination of neural networks for classification and ensemble outlier detection for unknown classes. We used the following datasets for our experiments.

5.1 DATASETS

To demonstrate the effectiveness of open set classification, we must choose a dataset with a large number of classes. This allows us to hide classes during training. These hidden classes can later be used during testing to gauge open set classification accuracy. We use the following two freely available data sets.

- **20 Newsgroups** [2] - Consists of 18,828 documents partitioned (nearly) evenly across 20 mutually exclusive classes.

- **Amazon Product Reviews** [3] - Consists of 50 classes of products, each with 1,000 review documents each. We perform topic-based classification. That is, given a review, the system decides what class of product the review is about.

Table 5.1 shows one example in each dataset.
Amazon Camera

I have read most of the review about this camera and agree with the 5-star review. This be my first dslr camera so most camera’s in this price range would have been great to I. I currently own a canon g12. I be amaze how easy it be to take picture due to the greatly reduce shutter lag. The control be user friendly Wish I could add more.

20 Newsgroups comp.windows

I’m asking for help on a sticky problem involving unreasonably low apparent precision in Z-buffering, that I’ve encountered in 2 different PEX implementations. I can’t find any discussion of this problem in any resources I can lay hands on. I am posting this article by itself on comp.graphics, and virtually the same article with a test program demonstrating the problem on comp.windows.

Table 5.1: Dataset Example Reviews
5.2 EVALUATION PROCEDURE

Closed set evaluation assesses a trained classifier with unseen data that belong to the classes that were shown during training. The function to separate these classes was learned during training. The classes presented during testing are exactly the same classes the model was trained on, although the examples presented are different. In open set evaluation, the classifier is tested with examples from classes never seen during training along with examples from the trained classes. Thus, we say the classifier is able to acquire incomplete knowledge during the training phase as it is compelled to make decisions on new examples during the testing phase.

During the training phase, we train the classifiers on a limited number of classes. During the testing phase, examples of unknown classes that were not learned during training, are shown to the classifier. We shuffle examples of the known classes and the unknown classes to create a test set, and feed them to our model while testing. We evaluate the performance of the classifier based on how well it identifies examples from unknown classes. In some sense, we amalgamate all unknown classes into an undifferentiated single unknown class. We can estimate the open set range of a classifier by using a measure called openness, proposed by [5, 6]. This measure is concerned only with the number of new classes introduced rather than the open space itself. openness is given by equation,

$$openness = 1 - \sqrt{\frac{2 \times |training\ classes|}{|testing\ classes| + |target\ classes|}}. \quad (5.1)$$

The above equation yields openness in terms of percentage (values between 0 and 100 percent), where 0 percent represents a completely closed set situation and values greater than 0 represent open set situation. Figure 5.1 shows variation of openness in the 10 class...
As shown in Figure 5.1, when we train our model on 10 classes and test on 10 classes, openness is 0. When the number of training classes is low, openness is high as there are many untrained classes. When the number of training classes increases, openness decreases since there are fewer untrained classes.

Accuracy, precision, recall, and F-score are used to measure the closed set performance of our model. The $F_1$-Measure is defined in terms of precision $p$ and recall $r$ as:

\[
p = \frac{tp}{tp + fp} \quad (5.2)
\]

\[
r = \frac{tp}{tp + fn} \quad (5.3)
\]

\[
F_1(p, r) = \frac{2rp}{r + p} \quad (5.4)
\]
where $tp$ is the number of true positives, $fp$ is the number of false positives and $fn$ is the number of false negatives [98].

These metrics are expanded to the open set scenario by grouping all unknown classes into one undifferentiated and combined unknown class. A true positive is when an example of a known class is correctly classified, and a true negative is when an example of an unknown class is correctly predicted as unknown. False positives (an unknown class predicted as known) and false negatives (a known class predicted as unknown) are the two types of incorrect class assignments.

### 5.2.1 OPEN SET EVALUATION

After training, the weights of the model are well learnt to make accurate predictions. For open set evaluation, we reserve some classes (as unseen) in training and mix them back during testing. We use 25%, 50%, 75%, or 100% classes for training in both 10 classes or 20 classes. Using 100% classes for training is the same as traditional closed set classification.

Consider the 20 Newsgroups dataset. Suppose we pick 10 random classes first which we call the “10 classes” scenario/situation. Next, we train the classifier on 2 random classes out of these 10 picked. Finally, we test the trained classifier on unseen examples of all 10 classes. This is the scenario in which the classifier was trained on only 25% of the classes on which it was eventually tested. Similarly in the “20 classes” scenario, for the 25% case, we use 5 classes for training and all 20 classes for testing (15 unseen classes in training).
5.3 PRELIMINARY EXPERIMENTS

The initial work in open set text classification was motivated by Bendale et al.’s [16] method of implementing CNNs for open set image classification. Prakhya et al. [32] adapted Bendale’s architecture for open set text classification and extended the work as required. The dataset was preprocessed before performing experiments such as splitting the texts of the document into words; then removing stop words, special characters, punctuation’s, headers and footers to prevent the model from overfitting on data; and finally, conversion to lower case. The headers and footers contain the products names, and it becomes easier for the classifier to know what the class is. The classifier will not be generalised as it is not deciding based on the review content. For the classification of new examples, an ensemble approach using Euclidean-Cosine, Mahalanobis and, Isolation forest was used. The Euclidean-cosine distance is a weighted combination of Euclidean and cosine distances.

Similar to [4, 30, 20], we conduct our experiments by introducing “unseen” classes during testing. For open set evaluation, we reserve some classes (as unseen) in training and mix them back during testing. We use 25%, 50%, 75%, or 100% classes (closed set) for training in all 10, 20 or 50 domains. By \( n \) domains, we mean that the testing is on examples from \( n \) classes. \( x\% \) means \( x \) percent of the classes were used for training although the testing was with \( n \) classes. Tables 5.5 and 5.6 in Section 5.8.1 shows the results obtained in different classes (25%, 50%, 75%, 100%) for two datasets.

\(^1\)I was the second author of the paper
5.4 RESULTS OF BIDIRECTIONAL LSTM ARCHITECTURE

The Bidirectional LSTM model as shown in Figure 4.5 with Input layer, Embedding layer, BiLSTM layer and, Output layer is trained for 100 epochs to learn features and develop the ability to classify examples on which they have not been trained. While testing for closed set text classification, the accuracy of the model that we obtained is 0.623, which is very low. To implement open set text classification, we need a base model that can produce high closed set text classification accuracy. We came to this conclusion by analyzing the previous results from all the other authors in open set text classification. Hence, we extend this model by adding another layer of BiLSTMs with the Time Distributed layer available in Keras [85]. We call this model as Hierarchical Neural Network.

5.5 RESULTS OF HIERARCHICAL NEURAL NETWORK ARCHITECTURE

We stack two layers of Bidirectional LSTMs with the TimeDistributed layer of Keras [85] in between and a final dense layer as shown in Figure 4.6. This model is trained for 100 epochs to develop the ability to classify examples on which they have not been trained with high accuracy. When tested for closed set text classification, the accuracy of the model obtained is 0.692, which is better than the previous model. We used this model for open set text classification, but got disappointing results as shown in Table 5.2.

We add attention mechanism to the HNN architecture to improve its performance in closed set text classification.
<table>
<thead>
<tr>
<th>20 Newsgroups</th>
<th>Number of tested classes = 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>% trained classes</td>
<td>25%</td>
</tr>
<tr>
<td>HNN</td>
<td>0.487</td>
</tr>
</tbody>
</table>

TABLE 5.2: Open Set Experiments on 20 Newsgroups Dataset (Tested on 10 classes) using HNN Model

5.6 RESULTS OF HIERARCHICAL NEURAL NETWORK WITH ATTENTION ARCHITECTURE

We introduced attention mechanism in both stacked layers. This model is also trained on 10 classes for 100 epochs to develop the ability to classify examples on which they have not been trained. When tested for closed set text classification, the accuracy of the model dropped to 0.528.

5.7 RESULTS OF CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE

The input to the CNN is a sentence that is tokenized and each token is represented as a word2vec \([77]\) word embedding. This is followed by a convolutional layer with multiple filters, then a max-pooling layer, then a fully connected layer with dropout, and finally a modified final layer. We obtain Activation Vectors (AV) from the penultimate layer, which are fed to an ensemble outlier detection model for final prediction as discussed in Chapter 4. The hyperparameters that we chose for our CNN model for all our experiments are summarized in Table Table 5.3. We pad each sentence to the maximum sentence length (reviews) in each class of the dataset which would range from 800 to 1400. Padding sentences to the same length is useful because it allows us to efficiently batch our data since
each example in a batch must be of the same length. We build a vocabulary index and map
each word to an integer between 0 and the vocabulary size (20,000 approx). Each sentence
becomes a vector of integers.

<table>
<thead>
<tr>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>word embedding</td>
<td>word2vec</td>
</tr>
<tr>
<td>filter sizes</td>
<td>(3,4,5)</td>
</tr>
<tr>
<td>feature maps</td>
<td>100 for each filter size</td>
</tr>
<tr>
<td>activation function</td>
<td>ReLU</td>
</tr>
<tr>
<td>pooling</td>
<td>1-max pooling</td>
</tr>
<tr>
<td>dropout rate</td>
<td>0.5</td>
</tr>
<tr>
<td>$L2$ Regularization</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**TABLE 5.3: CNN Baseline Configuration**

We trained our CNN model on 10 classes and recorded the closed set test accuracy. We
got a closed set accuracy of 0.86. Table 5.4 summarizes all the results of training the models
on 10 classes of the 20 NewsGroup dataset and tested on unseen examples belonging to the
same 10 trained classes. This scenario represents closed set text classification. From the
results, it is evident that the CNN model outperformed other models.

Additionally, we observed that the variants of the RNN models took more time to train
(more than 2 days for each training case) as compared to the CNN model (less than a day)
and also performed poorly in feature extraction. They could not produce high closed set

<table>
<thead>
<tr>
<th>Models</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>BiLSTM</td>
<td>0.623</td>
</tr>
<tr>
<td>HNN</td>
<td>0.692</td>
</tr>
<tr>
<td>HNN with Attention</td>
<td>0.528</td>
</tr>
<tr>
<td>CNN</td>
<td><strong>0.861</strong></td>
</tr>
</tbody>
</table>

**TABLE 5.4: Closed Set Evaluation of All Models**
classification results. Thus, the conclusion that we draw from the variants of the RNN models is that they are not suitable for open set classification.

For all our experiments, CNN is the base model. The penultimate layer is the main focus of our network because this is the layer that contains the real activations for nodes corresponding to the various classes of the train and test samples of our dataset. We normalize these real activations during training and testing using softmax equation.

\[
softmax(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}
\]  

(5.5)

However in our case during testing, there is an unknown class to be considered as well and we do not know the activations or probabilities associated with such an unknown class. Therefore, we need a modified softmax layer to take care of this situation. Bendale et al. [16] had replaced the layer that computes softmax with the so-called OpenMax layer, which computes the probability of an example belonging to a known or unknown class. Prakhya et al. [32] used the same approach to identify whether a test example belonged to a known class or unknown class.

Our new model also uses an ensemble approach with three outlier detection models to make decisions with the activations in the penultimate layer. Our model need not be retrained after the introduction of a new unknown class as open set determination happens after training. In our experiments discussed here, we compare the performance of our ensemble-based open set text classifier with other open set classifiers that have been previously used with open set image classification and with the methods of Prakhya et al. [32], Fei et al. [4] and, Doan et al. [30], which were used for open set text classification.
As discussed in Chapter 4, we collect activation vectors for test samples from the penultimate layer. We use an ensemble outlier detection model to determine whether a test example is from a known class or not. This ensemble includes the following three sub-models:

1. Mahalanobis Distance (Distance based outlier detection),
2. Local Outlier Factor (Density based novelty detection), and
3. Isolation Forest (High dimensional outlier detection).

The high level description of steps followed in ensemble approach is given below.

- Give the activation vectors of each test example to the Mahalanobis Distance model. Obtain the predictions for each test example.
- Feed the activation vectors of each test example to the LOF model. Get the predictions for each test example.
- Give the activation vectors of each test example to the Isolation Forest model and obtain the predictions for each test example.
- Take majority voting among each test example’s prediction to get the final result.

We give a detailed description how each model is able to predict a test example.

5.7.1 MAHALANOBIS DISTANCE

We find the mean or centroid for all the activation vectors of every trained class obtained from the penultimate layer. Then, we deviate from the original OpenMax [16] and adapt Prakhya et al.’s [32] method of finding the $k$-nearest examples to the centroid of every training class. We refer to these examples as $k$-Class Activation Vectors ($k$-CAV). For every
example in a training class, we calculate the Mahalanobis Distance between the respective AV and the $k$-CAVs. Doing so results in $k$ distances per AV. We take the average of these $k$ calculated distances. As the number of classes in our dataset is far less than those used in image classification, the $k$-CAVs of a class are used to represent a class more accurately than a single mean activation vector. This also mitigates the effect of outlier AVs in a class.

We observe that when $k$ is around 10, the trade-off between performance and computation time is optimized. Therefore, for all experiments, we fix the value of $k = 10$. Figure 5.2 shows $k$-Class Activation Vectors.

---

**Algorithm 1 Weibull Model**

**Require:** Activation vectors in the penultimate layer $v(x) = v_1(x) \ldots v_N(x)$

**Require:** FitHigh function from libMR library

```plaintext
for i = 1...N do
    For each class, compute class mean
end for

for i = 1...N do
    compute Mahalanobis distance between $v_i$ and class mean
end for

Find k-nearest vectors to the class centroid (k-CAV)

for i = 1...N do
    for j = 1...K do
        compute Mahalanobis distance between $v_i$ and k-CAV
    end for
    $v_i = \text{mean(k-CAV)}$ libMR models = FitHigh($v_i$, tailsize)
end for

Returns libMR models with w.scores, Weibull shape and scale parameters.
```

The distances obtained are used to generate a Weibull model for every training class. We follow Bendale et al. [13] methodology to use the libMR$^2$ [99] *FitHigh* function that returns Weibull shape and scale parameters (Weibull fitting parameters) and w.scores. The

---

$^2$https://github.com/Vastlab/libMR
Figure 5.2: k-Class Activation Vectors

FitHigh function takes in activation vectors and tailsize as input parameters. Tailsize is the largest distance to the mean activation vector. The output from the FitHigh is libMR model or Weibull model with w_scores which is a weibull score. We compute probability of inclusion for each activation vector of a class by subtracting w_scores from 1.

Figure 5.3 shows the probabilities of inclusion obtained from the generated Weibull model for 2 training classes belonging to the 20 Newsgroups dataset. From the Figure 5.3, it is evident that as an example deviates more from the class center (k-CAVs), the probability of inclusion of that example decreases.

The sum of all inclusion probabilities is taken as the total probability that a test example belongs to a known class. It is also called the known class probability $Known_p$ for the test example $t$:

$$Known_p(t) = \sum_{i=1}^{n} P(c_i | t)$$  \hspace{1cm} (5.6)

where $n$ is the total number of known classes. However, in open set text classification, there
are unknown classes that will occur at test time and, hence, it is not appropriate to require the probabilities to sum to 1 (as we usually do in standard softmax layer for classification). We need to find unknown class probability $Unknown_p$ which is computed by subtracting 1 from known class probability.

$$Unknown_p = 1 - Known_p(t) = 1 - \sum_{i=1}^{n} P(c_i|t)$$  \hspace{1cm} (5.7)

where $C_i$, is a known class for $i = 1....n$

We compare the maximum known class probability $Known_p$ and unknown class probability $Unknown_p$. If the unknown class probability is greater than the known class probability, we label the example as belonging to the unknown class; otherwise, the example is assigned to the class with the highest known class probability.

Let us consider an example. Assume the test example has an inclusion probability of 0.6 and 0.2 with respect to $c_1$ and $c_2$. The total inclusion probability is 0.8, which represents known class probability. The unknown class probability is $1 - 0.8 = 0.2$. We compare 0.2 with $\max(0.6, 0.2)$. The unknown class probability is lesser than 0.6, thus the test example
belongs to \( c_1 \). On the other hand, assume the test example has an inclusion probability of 0.2 and 0.1 with respect to \( c_1 \) and \( c_2 \). The known class probability is 0.3 and the unknown class probability is \( 1 - 0.3 = 0.7 \). After comparing 0.7 with \( \max(0.2, 0.1) \), we decide that the test example belongs to the unknown class.

For every example in the test set, we follow all the above procedure to compute whether it belongs to a known class or the unknown class. The results are stored for final voting.

### 5.7.2 LOCAL OUTLIER FACTOR

We fit the LOF algorithm \(^3\) (discussed in Chapter 4, Section 4.6.4) on the activation vectors for the training samples, and the algorithm provides a learned frontier or boundary delimiting the contour of the distribution obtained from the training samples. If any test sample lies within the learned boundary, it is assumed to belong to the same class. Otherwise, if it lies outside the boundary, it is considered as belonging to unknown class.

During testing, we predict the LOF model on the activation vectors obtained from the penultimate layer. The LOF model generates outlier scores for each test example based on a density based approach. If the model is trained for 2 classes, for each test example two scores are computed with respect to two classes. If the two scores are negative, the test example is considered to belong to the unknown class. If the scores are positive, the test example is assigned to the class with highest positive score. The procedure is followed for all test examples and the results are stored for final voting.

\(^3\)https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html
5.7.3 ISOLATION FOREST

We fit the Isolation Forest algorithm \(^4\) (discussed in Chapter 4, Section 4.6.5) on the activation vectors for the training samples obtained from the penultimate layer. During testing, we apply the Isolation Forest model on the activation vectors of the test examples. The scores obtained from the Isolation Forest algorithm are min-max normalized. The lower the scores of test example, the more abnormal the test example is. If the model is trained on 2 classes, for each test example there are two scores with respect to two classes. If both the scores are below the threshold values (given by the algorithm), the test example is considered an outlier. If the score of the test example is above the threshold value, the test example represents an inlier and belongs to the class with the higher score. The procedure is followed for all test examples and the results are stored for final voting.

After we obtain the predictions from all the three outlier detection models, we use a voting scheme to pick the final prediction. In most of the test cases, Mahalanobis model performs very well. Therefore, the predictions from the Mahalanobis model is used as a tie-breaker in case of differing predictions.

We experimented by adding 2 more outlier detection models along with above 3 ensemble models, based on distance, namely Euclidean distance model and Cosine distance model. We followed the same procedures as Mahalanobis model and generated predictions for test samples. We took voting scheme for all 5 ensemble outlier detection models. The results obtained remained almost the same as that of 3 ensemble detection models. On analysis, we found that Euclidean and Cosine distance model performed poorly as compared to other 3 models and hence did not contribute much towards the end result of ensemble

Chapter 5. EXPERIMENTAL PROCEDURE AND EVALUATION

5.8 RESULTS

5.8.1 PRELIMINARY EXPERIMENT RESULTS

Tables 5.5 and 5.6 show the preliminary results obtained in different scenarios (25%, 50%, 75%, 100%) for two datasets. This thesis aims to improve the results by trying different architectures. The following sub section gives details of results obtained from each architecture. The highest scores are bolded.

<table>
<thead>
<tr>
<th>% trained classes</th>
<th>Number of tested classes = 10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>25%</td>
</tr>
<tr>
<td>CNN([32])</td>
<td>0.797</td>
</tr>
<tr>
<td>NCC §</td>
<td>0.61</td>
</tr>
<tr>
<td>cbsSVM*</td>
<td>0.450</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>% trained classes</th>
<th>Number of tested classes = 20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>25%</td>
</tr>
<tr>
<td>CNN([32])</td>
<td>0.648</td>
</tr>
<tr>
<td>NCC §</td>
<td>0.606</td>
</tr>
<tr>
<td>cbsSVM*</td>
<td>0.566</td>
</tr>
</tbody>
</table>

TABLE 5.5: Preliminary Experiments on Amazon Product Reviews dataset (Tested on 10 and 20 Classes)

5.8.2 EXPERIMENT RESULTS

The results of the CNN network, along with the outlier ensemble model for the Amazon review dataset (10, 20 and 50 Classes) are shown in Table 5.7 and the results of 20 News-Group dataset (10 and 20 Classes) are shown in Table 5.8. For every scenario, we report

---

5cbsSVM* [29] performed better in all cases, hence previous model results are not shown here.
Chapter 5. EXPERIMENTAL PROCEDURE AND EVALUATION

<table>
<thead>
<tr>
<th>20 Newsgroups</th>
<th>Number of tested classes = 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>% trained classes</td>
<td>25%</td>
</tr>
<tr>
<td>CNN([32])</td>
<td>0.719</td>
</tr>
<tr>
<td>NCC (\frac{5}{3})</td>
<td>0.652</td>
</tr>
<tr>
<td>cbsSVM*</td>
<td>0.417</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>20 Newsgroups</th>
<th>Number of tested classes = 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>% trained classes</td>
<td>25%</td>
</tr>
<tr>
<td>CNN([32])</td>
<td>0.668</td>
</tr>
<tr>
<td>NCC (\frac{5}{3})</td>
<td>0.635</td>
</tr>
<tr>
<td>cbsSVM*</td>
<td>0.593</td>
</tr>
</tbody>
</table>

Table 5.6: Preliminary Experiments on 20 Newsgroups Dataset (Tested on 10 and 20 Classes)

our results using 5 random train-test partitions for each dataset. For example, we train a model on random 2 classes and test it on 10 classes (2 known + 8 unknown) for 25% scenario in the 10 class scenario. The 8 unknown classes are selected randomly. We create 5 such models that are trained on 2 different classes. The final result reported is the average of all five models. The same procedure is followed for all other scenarios (50%, 75% and 100%). The results obtained by using diversified outlier detection methods improved significantly from the results of the preliminary model named CNN([32]). We report F-scores as all other models report F-scores and the classifiers used as baselines for comparison are briefly described below (in detail we covered in Chapter 3, Section 3.2).

- **1-vs-rest-SVM** - Standard 1-vs-rest multiclass SVM with Platt Probability Estimation [12].
- **1-vs-set-linear** - A generalization of the 1-class Support Vector Machine to better support generalization and specialization consistent with the open set problem definition [5].
Chapter 5. EXPERIMENTAL PROCEDURE AND EVALUATION


- **P_i-SVM** - SVM model that estimates the unnormalized posterior probability of class inclusion [11]

- **ExploratoryEM** - “Exploratory” version of Expectation-Maximization algorithm (EM) [28]

- **cbsSVM** - Center-Based Similarity Space SVM [4]. It uses SVM to build 1-vs-rest CBS classifiers for multiclass text classification with rejection option.

- **NCC** - Nearest Centroid Class model [30] uses a density-based method following the approach of the clustering algorithm called DBSCAN [31].


F-score performance numbers of 1-vs-rest-SVM, 1-vs-set SVM, W-SVM, P_i-SVM, and cbsSVM are from the study by Fei et al. [4], marked as *. Results pertaining to the Nearest Centroid Class model (NCC) are from the study by Doan et al. [30], marked as §. The results from Deep Open Classification (DOC) are from Shu et al. [20]. Our model performs better than cbsSVM and NCC classifiers in smaller number of classes. DOC performs better than all other models considering results reported by the Shu et al. [20]. However, they used an experimental protocol that is different from that is used by us and several other papers. They mention that they split the training set into 60% training, 10% validation and 30% test set, and perform their evaluation on this test set. They do not perform any data preprocessing. On the other hand, we clean our dataset before performing experiments. We remove headers and footers to create a more generalized model. Our model is generalised as it learns to classify based on the review contents. We test our model
with the separate test set instead of splitting the training set into 30% test set. The evaluation protocol we follow is completely different from the way DOC conducts evaluation. We believe the CNN approach effectively isolates documents in smaller number of classes compared to other SVM-based approaches. Unlike cbsSVM, our model do not have to retrain when new unknown classes are introduced and hence, our models are more viable in real world scenarios.

Figure 5.4 shows the activation vectors obtained from models trained on 2 classes plotted in 2-dimensional space. The plots show distinct clusters of activation vectors. As seen from the plots, the activation vectors obtained from 2 classes of the 20 Newsgroups dataset are very close to each other and the activation vectors obtained from 2 classes of Amazon dataset are well separated. Most of the examples in the 20 Newsgroups are overlapping, making it harder for the model to correctly classify them. Thus, the results of the Amazon review dataset are higher compared to that of the 20 Newsgroups dataset. We believe that the CNN approach effectively isolates documents in smaller number of classes compared to other SVM-based approaches.

**Figure 5.4**: Activation Vectors of Two Classes.
<table>
<thead>
<tr>
<th>Number of tested classes</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Amazon Product Reviews</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>% trained classes</td>
<td>25%</td>
<td>50%</td>
<td>75%</td>
<td>100%</td>
</tr>
<tr>
<td><strong>CNN with outlier ensemble</strong></td>
<td>0.841</td>
<td>0.789</td>
<td>0.755</td>
<td>0.877</td>
</tr>
<tr>
<td>DOC</td>
<td>0.590</td>
<td>0.741</td>
<td>0.791</td>
<td>0.854</td>
</tr>
<tr>
<td>CNN ([32])</td>
<td>0.797</td>
<td>0.753</td>
<td>0.727</td>
<td>0.821</td>
</tr>
<tr>
<td>NCC §</td>
<td>0.61</td>
<td>0.714</td>
<td>0.781</td>
<td>0.854</td>
</tr>
<tr>
<td>cbsSVM*</td>
<td>0.450</td>
<td>0.715</td>
<td>0.775</td>
<td>0.873</td>
</tr>
<tr>
<td>1-vs-rest-SVM*</td>
<td>0.219</td>
<td>0.658</td>
<td>0.715</td>
<td>0.817</td>
</tr>
<tr>
<td>ExploratoryEM*</td>
<td>0.386</td>
<td>0.647</td>
<td>0.704</td>
<td>0.854</td>
</tr>
<tr>
<td>1-vs-set-linear*</td>
<td>0.592</td>
<td>0.698</td>
<td>0.700</td>
<td>0.697</td>
</tr>
<tr>
<td>wsvm-linear*</td>
<td>0.603</td>
<td>0.694</td>
<td>0.698</td>
<td>0.702</td>
</tr>
<tr>
<td>$P_1$-svm-linear*</td>
<td>0.600</td>
<td>0.695</td>
<td>0.701</td>
<td>0.705</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Amazon Product Reviews</strong></th>
<th>Number of tested classes = 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>% trained classes</td>
<td>25%</td>
</tr>
<tr>
<td><strong>CNN with outlier ensemble</strong></td>
<td>0.663</td>
</tr>
<tr>
<td>DOC</td>
<td>0.639</td>
</tr>
<tr>
<td>CNN ([32])</td>
<td>0.648</td>
</tr>
<tr>
<td>NCC §</td>
<td>0.606</td>
</tr>
<tr>
<td>cbsSVM*</td>
<td>0.566</td>
</tr>
<tr>
<td>1-vs-rest-SVM*</td>
<td>0.466</td>
</tr>
<tr>
<td>ExploratoryEM*</td>
<td>0.571</td>
</tr>
<tr>
<td>1-vs-set-linear*</td>
<td>0.506</td>
</tr>
<tr>
<td>wsvm-linear*</td>
<td>0.553</td>
</tr>
<tr>
<td>$P_1$-svm-linear*</td>
<td>0.547</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Amazon Product Reviews</strong></th>
<th>Number of tested classes = 50</th>
</tr>
</thead>
<tbody>
<tr>
<td>% trained classes</td>
<td>25%</td>
</tr>
<tr>
<td><strong>CNN with outlier ensemble</strong></td>
<td>0.623</td>
</tr>
<tr>
<td>DOC</td>
<td>0.612</td>
</tr>
<tr>
<td>NCC §</td>
<td>0.55</td>
</tr>
<tr>
<td>cbsSVM*</td>
<td>0.557</td>
</tr>
<tr>
<td>ExploratoryEM*</td>
<td>0.348</td>
</tr>
<tr>
<td>1-vs-rest-SVM*</td>
<td>0.460</td>
</tr>
<tr>
<td>1-vs-set-linear*</td>
<td>0.420</td>
</tr>
<tr>
<td>wsvm-linear*</td>
<td>0.488</td>
</tr>
<tr>
<td>$P_1$-svm-linear*</td>
<td>0.487</td>
</tr>
</tbody>
</table>

**Table 5.7:** Experiments on Amazon Product Reviews Dataset (Tested on 10, 20 and 50 Classes)

*a*We report results based on our runs. The results reported in Shu et al. [20] are based on different experimental protocol.
### Table 5.8: Experiments on 20 Newsgroups Dataset (10 and 20 Classes)

We report results based on our runs. The results reported in Shu et al. [20] are based on different experimental protocol.
CHAPTER 6
CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

This thesis addresses the limitation closed set text classification by presenting a model that combines CNN classification with an ensemble for outlier detection. Our open set approach can recognize text documents of classes unseen during training, when the number of trained classes is small, more consistently than existing open set text classification models that are competitive, namely CBS learning [29] and the NCC model [30]. Our model is a generalized model unlike DOC as we train the model by removing headers and footers of the text examples. Headers and footers contain an abundance of clues that distinguish newsgroups, the classifiers barely have to identify topics from text at all, and they all perform at the same high level. By removing headers and footers, our model learns to classify solely only on the text content, without its title or other annotations, and hence, has better capability to recognize examples from unknown classes. This research can prove beneficial when classifying new data and can be used in applications with harder text classification problems like forensic linguistics, such as identifying authorship of threatening emails or malicious code.

We worked with three different outlier detection methods namely a distance based approach and a density based approach and an approach that works well in high dimensional regions. The combination Mahalanobis Distance and Weibull distribution based method for outlier detection performed well with many examples from both the datasets. We can use...
an ensemble of outlier detection methods in real world applications such as fraud detection in credit cards, intrusion detection in cyber security, medical analysis and fault detection in safety critical systems, in addition to outlier detection in text documents.

Our hypothesis about RNNs and its variants was that they are good at learning sequences of data and hence, good at text classification. But after conducting experiments our hypothesis proved wrong. CNNs are best at text classification by learning patterns in the text. We applied an ensemble of outlier detection models following raw classification by a CNN to identify new examples that do not belong to any of the trained classes. By combining the results from different outlier detection techniques we improved the $F_1$-score for open set classification.

A part of this thesis, discussing a similar architecture, but a different outlier detection ensemble [32] was published in ICON 2017 (International Conference on Natural Language Processing). This thesis diversifies the components of the outlier detection ensemble to improve open set classification results.

### 6.2 FUTURE WORK

In the current research model, we obtained activation vectors from the penultimate layer of the CNN and then apply outlier detection algorithms to make a final class prediction for an example. We used supervised learning in which both input and desired output labels are provided for text classification. We can design a semi-supervised learning model which typically has a small amount of labeled data but a large amount of unlabeled data. For example, from the 20 Newsgroups dataset, we can select examples from 8 classes as labeled data and use examples from the remaining 12 classes as unlabeled data. Chapelle et al.[100] found that unlabeled data, when used in conjunction with a small amount of labeled data,
can produce considerable improvement in learning accuracy over unsupervised learning where no data is labeled, but without the time and costs needed for supervised learning where all data is labeled.

Another possible future research work will be to build an end-to-end neural network model for open set text classification similar to Dhamija et al.’s [21] work in open set image classification. We can feed the model with outliers along with training examples. This makes the model learn about outliers during the training phase itself instead of introducing outliers during testing. This approach of introducing outliers during the training phase contradicts the assumption of open set classification as the outliers are known from before.

Another expansion to this research work is to classify examples in the unknown class. Instead of categorizing all new examples as belonging to a single undifferentiated unknown class, we should be able to sub-categorize unknown examples further into additional classes. This will necessitate the development of metrics to split the undifferentiated unknown class to two or more unknown classes, when appropriate. Only if we are able to instantiate a sequence of new classes, as necessary, we can consider the work to be able to instantiate newly discovered classes and hence pave the pathway for incremental class or concept learning. This will be a small step towards lifelong learning [1]. To facilitate incremental learning we should be able to generate artificial outliers. We need to investigate on generating artificial outliers. Possible solutions could be to add outliers to real data by randomization methods or by downsampling a class to desired sparsity to get unknown class.

In authorship attribution, the number of authors is potentially very large, and our approach works well only when the number of target test classes is very small. Our future work in this area is to explore deeper and improvise open set classification abilities when...
trained on large number of classes. In this thesis, we worked with 10 and 20 classes belonging to the Amazon review dataset. A future work would be to handle larger number of classes (30, 40 or 50) belonging to the Amazon review dataset.
REFERENCES


REFERENCES


REFERENCES


REFERENCES


[92] K Ismo. “Outlier detection using k-nearest neighbour graph”. In:


