One Class Text Classification using an Ensemble of Classifiers

Hemanth Kumar Reddy, Mayaluru
Matriculation number: 3063488

January 2020

A thesis submitted in partial fulfillment for the
degree of Master of Science

Institute of Computer Science

Supervisors:
M.Sc. Mohnish Dubey, University of Bonn

Examiners:
Prof. Dr. Jens Lehmann, University of Bonn
Dr. Andreas Hamm, German Aerospace Center (DLR)

INSTITUT FÜR INFORMATIK
RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT BONN

Declaration of Authorship

I, Hemanth Kumar Reddy Mayaluru, declare that this thesis titled, 'One Class Text Classification using an Ensemble of Classifiers' has been written independently and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- Where none other than the specified sources and aids were used.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ______________________________

Date: ______________________________

Acknowledgements

I wish to thank everyone, who has been a part of my journey. First and foremost, I would like to thank my thesis supervisors, M.Sc. Mohnish Dubey and Dr. Andreas Hamm, for their advice, patience and willingness to help me with any thesis related issues. Also, I'm grateful to Prof. Dr. Jens Lehmann and Dr. Andreas Hamm for being my examiners and for the encouragement in my research.

I am thankful to Dr. Mark Azzam, head of the department, Think Tank, German Aerospace(DLR) for providing me an opportunity to work at DLR for my thesis. Furthermore, I would like to thank Mr. Sommer Thorsten for helping me providing access to the infrastructure.

I would like to thank my professors at the Informatik department, the teaching and non-teaching staff of the University of Bonn. I'm grateful to the German Aerospace(DLR), for helping me with the resources required to implement my thesis.

I am profoundly thankful to Ms. Barshana Banerjee for helping me with the thesis report emendations and also for keeping me motivated and being a great support.

I owe my special thanks to my friends especially Mr. Goutham Sai, Mr. Konda Ajith Reddy and Mr. Nikhil Mss who believed in me and being part of my journey.

Last but not least, I would like to take this moment to thank my family especially my father, mother, brother, aunt Mrs. Seetha Lakshmi and uncle Mr. Dhanunjaya Reddy for having faith in me and supporting me all the way. I dedicate this thesis to my grandmother Mrs. Lakshmi Narayanamma who was always there for me.

RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT
BONN

Abstract

Institute of Computer Science

Master of Science

by Hemanth Kumar Reddy, Mayaluru

Traditional classification algorithms work in a closed-world scenario where the training data contains all existing classes. In contrast, open set classifiers can handle new input that does not belong to any of the classes seen during training. Open set classification has been studied intensively in the computer vision domain, primarily in handwriting recognition, face recognition, object classification and computer forensics. Here we are interested in open set classification in natural language processing in one class document classification. We propose a new system based on autoencoder for one class classification of documents leveraging the full text. Extending further, we propose a novel ensemble based classifier model, a combination of several basic classifiers, to detect if an incoming document belongs to the class known from training or an unknown class. We compare and evaluate our methods on existing one class classification datasets for NLP - 20 Newsgroups, reuters and webkb. We also extract and use a new full-text dataset from arxiv.org. Our methods significantly outperforms the current state-of-the-art approaches for one class document classification.

Contents

Chapter 1

Introduction

1.1 Motivation

Classification is the process of grouping set of observations into few sets of classes based on some properties. With lots of advancements in classification tasks, text classification has been studied intensively and addressed many real world applications such as sentiment analysis, movie reviews classification, email spam filtering, biomedical studies and document classification. Based on the methods adopted during training and testing of a model, text classification systems are categorized into two types - closed set classification and open set classification.

In closed set classification, the training dataset and test dataset(the real data) have the same classes. For example, if the training set has data samples that belong to n classes, then each member of the test set will have one of those n class labels(see fig 1.1) that are already seen by the classifier during training.

The scenario in open set classification is different from closed set classification. Here the test set will have class labels that are not seen during the training(see fig 1.2). The classifier should be able to identify the new unseen example as unknown.

Challenges in Text Classification:

- In real-world scenarios, a trained classifier is likely to come across text documents that do not belong to any of the known classes. If such text documents are given to a closed set classifier, it mis-classify the documents into one of the known classes leading to poor model performance.

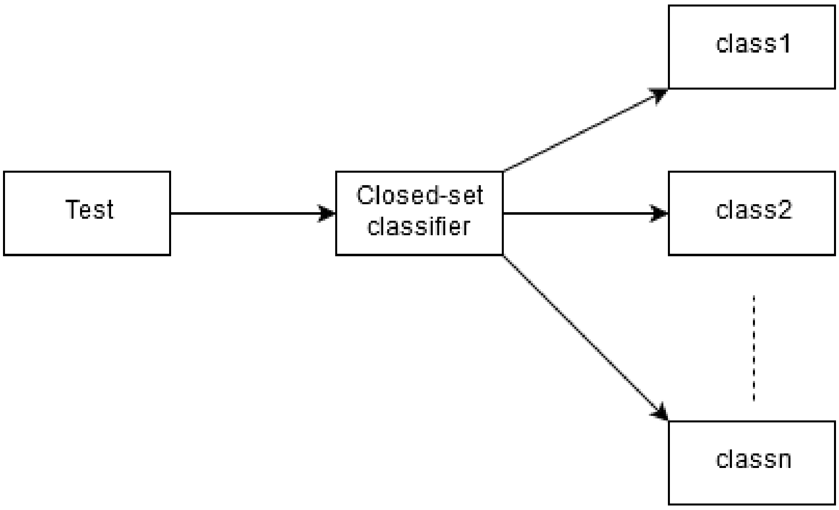


FIGURE 1.1: Closed set classifier

- Considering the rate(volume and variety) at which the text documents are growing across web, it is almost impossible to identify training data from all the classes and train a classifier.
- Under-sampling or no sampling of the data from unknown classes. Thus no prior information about the unknown classes which makes it difficult for the classifier to classify.

1.2 Problem Statement

One class classification is the special case of open set classification where n=1, which means that we need to identify if a given text document belong to the *known* class or an *unknown* class. This problem has been studied under various names such as - anomaly detection, redundancy detection, novelty detection and outlier detection. Recent studies have shown success in open set classification for images. We focus here on classifying texts into one class(known class) or unknown class, which is an important application, considering the exponential growth of text documents every year. In our approach we leverage the information from large set of documents to classify if a new upcoming document(which is not seen during training) belongs to a given class or not.

One class SVM[1], Isolation forest[2] and CNN[3] are the state-of-the-art methods used for open set text classification. Autoenoders[4] achieved promising results for unsupervised learning in the field of computer vision. This motivated us to build a new autoencoder method for one class text classification. Using dimensionality reduction techniques we could achieve better results on all the above models. We observed from our initial experiments that one class SVM is better in finding true positives, while isolation forest is good at predicting true negatives. So if we combine the individual methods as an ensemble, we get a better model. This motivated us to research on the ensemble of the methods and develop a novel ensemble of one class classifiers that outperformed all of the individual methods on standard one class text classification datasets.

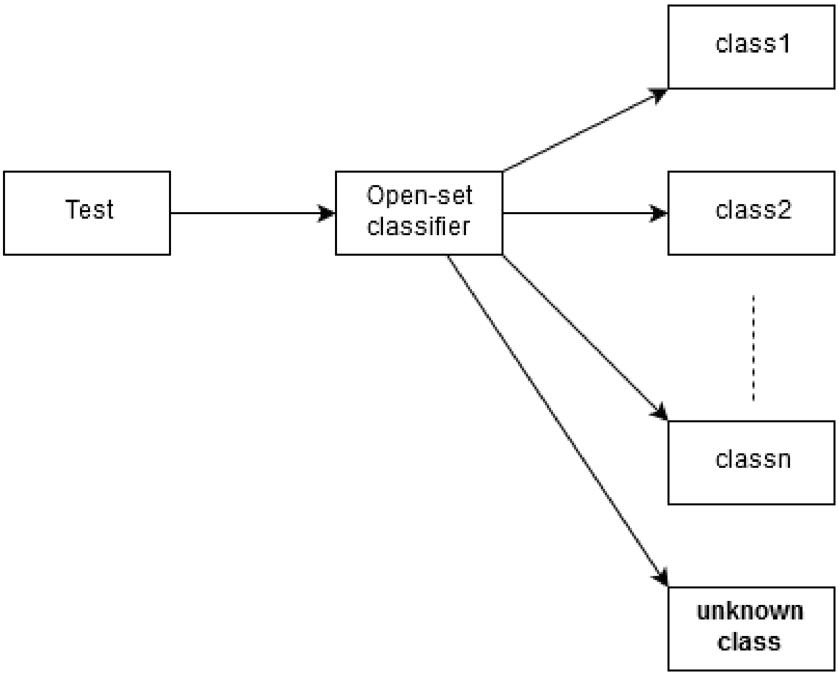


FIGURE 1.2: Open set classifier

Most of the work seen in the text classification is done under closed set assumption. The drawback of this approach is if an unknown new observation is seen, the classifier will wrongly classify it into one of the already seen classes. To avoid this problem open set classifier is considered.

Open set classification is a popular approach in image classification and lot of research has been done in the field of images. But research on open set classification for natural language processing is very limited.

1.3 Contributions

Overall, contributions in this thesis are as follows:

1. An autoencoder based approach for one class text classification. To the best of our knowledge, this is a first approach for one class text classification.
2. An ensemble based open-set document classification model that leverages the power of various machine learning models.

3. An experimentation with various word embeddings(simple frequency-based, high dimensional and transformer-based techniques) described in Sec. 4.3
4. A detailed comparative analysis of the proposed models with state-of-the-art techniques

1.4 How the Thesis is Organised

The thesis is organised into following chapters:

- Introduction and approach(Chapter 1)
- Related Work in one class classification in text and image domains(Chapter 2)
- Chapter 3 describes about the preliminaries and definitions;
- In chapter 4 we cover theoretical background regarding Support Vector Machines, Neural Networks, Isolation forest, Auto encoders for the domain of NLP and ensemble approaches. Finally, the chapter also covers insights of word embeddings
- The methodology we developed is explained in chapter 5
- In chapter 6, we discuss about the datasets and evaluation results in tables and figures;
- Chapter 7 concludes with future work details.

Chapter 2

Related Work

The main assumption for the closed classification algorithms is that the data from the test set should belong to one of the classes in the training set. For open set, these algorithms must be adaptive for the unseen classes during testing.

2.1 Open Set Classification in Computer Vision

The problem of open set classification has been studied intensively in the domain of Computer vision [5] [6] [7] . Often it is impossible to include all classes of images in the training set. So there is a need for a classifier that can identify a class of images not seen during training as unknown. This type of classifier is also used in anomaly detection problems [8].

In the domain of computer vision, Scheirer et al.[9] researched on recognising the images that the system has not seen during training by reducing open space risk and empirical risk. The key idea is that a classifier should not cover too much open space where there are few or no training data examples where the probabilities of classification are unknown. When the classifier mis-classifies the data, empirical risk occurs and the fact that the presence of unknown classes is likely to cause errors into classification decisions is recognised by open space risk. Their model reduces the risk by introducing parallel hyperplanes, one near the class boundary and another away from the boundary, and then develops a greedy optimization algorithm that moves the planes incrementally by modifying a linear SVM.

Later Scheirer et al. [10] explored non-linear kernels that reduce the open space risk by taking the sets with finite measure, labeling them as positive, and formulated a compact abating probability (CAP) model, where probability of class membership abates as points move from known data to open space. A novel technique called Weibullcalibrated SVM (W-SVM) was proposed by Scheirer et al., which combined the statistical extreme value theory (EVT) for score calibration with two separated SVMs. By fitting Weibull distribution over positive class scores from SVM, the probability for each class inclusion is estimated.

Bendale and Boult [11] proposed a distance-based approach known as Nearest Non Outlier (NNO) for open set recognition by extending the Nearest Class Mean (NCM) classifier [12]. The NNO algorithm performs classification based on the distance between the test sample and the means of known classes. It detects outliers for bounding the open space risk, and when all classifiers reject an input sample, it rejects that sample . By replacing the Euclidean distance with a learned low-rank Mahalanobis distance in NCM technique, an extended approach nearest class mean metric learning (NCMML) [12] has been formulated . This gives better results than NCM as the algorithm is able to learn features inherent in the training data.

Neural networks have gained significant popularity for various tasks such as Image recognition, Natural language processing, text classification, etc. They follow a typical SoftMax cross-entropy classification loss, which incurs the normalization problem, making the neural networks inherently have the closed set nature. Bendale and Boult [13] proposed a Convolutional Neural Network(CNN) based open set classification in vision domain as a first solution towards open set Deep Networks by replacing the Softmax layer with an Openmax layer. The openmax layer estimate the probability of an input belonging to the unknown class.

Autoencoders, though originally developed for the dimensionality reduction and feature extraction, are capable to model the training data distribution making them suitable for anomaly detection[14]. Almost all approaches for anomaly detection with autoencoders need the training data to have examples from seen class, but this alone is no guarantee for unseen examples to have large reconstruction errors. Denoising autoencoders are proposed in [15] that learn to reconstruct images from input images(noise corrupted). This regularization makes the latent representation that encodes the most relevant image features. This method achieved significantly better results.

[16] proposed Adversarialy learned generative adversarial networks one-class classification framework that is composed of two main modules: Network R that acts as a prepossessing and Refinement (or Reconstruction) step that works as the novelty detector,and Network D performs the Discrimination (or Detection) supporting nerwork R by enhancing the inlier samples and distorting the outliers. The principle is that instead of deciding on the original samples, the separability of the enhanced inliers and distorted outliers is much better.

2.2 One Class Classification in Natural Language Processing

The earliest approaches for one class text classification in NLP includes one-class SVM [17] and support vector data description (SVDD) [18]. One-class SVM, a traditional ML based technique, separates the training samples from the origin of the feature space with a maximum margin. SVDD encloses the training data with a hyper sphere of minimum volume.

In [17], a method of adapting the SVM methodology to the one-class classification problem was suggested. After transforming the features using a kernel, the origin is treated as the only member of the second class. Then using some relaxation parameters the document of the one class is separated from the origin. Then the standard two-class SVM techniques are employed.

With some assumptions of the above model, [1] proposed an outlier methodology using SVM. The idea is to work in feature space under the assumption that not only is the origin the second class, but also the data points that are close enough to origin are considered as outliers. For this SVM, we need to decide how far a point from origin can be before classified as outlier.

[19] proposed a method for one class document classification using neural networks. In their methodology, they presented that a feed-forward neural network under a bottle neck can be used as a one class classifier by incorporating the restriction of known examples. Under the assumption that the documents are represented in an m-dimensional space, the network can be choosen as a three level network with m inputs, m outputs and k neurons on the hidden level, where $k < m$. The network is then trained, under back propagation to learn the identity function on the examples.

Fei and Liu [20] proposed a novel solution for open set text classification under using centerbased similarity (CBS) space learning.

Prakhya et al. [3] followed the OpenMax approach to explore the open set text classification. Shu et al. [21] replaced the SoftMax layer with a 1-vs-rest final layer of sigmoids and presented Deep Open classifier model. In this approach, the decision boundaries of sigmoid functions are tightened with Gaussian fitting to reduce the open space risk. CNN from [22] is used and the output of the final openmax layer known as document activation vector is passed to an ensemble model for the output prediction.

Deep Open Classification(DOC) proposed by [21] uses CNN by [22] as its base and adds a 1-vs rest final sigmoid layer and Gaussian fitting for classification. DOC uses the

OpenMax layer from [23] for 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(seen) examples and all the rest examples with label not equal to l_i as negative(unseen) examples.

Context Vector Data Description [24], a recent approach, is a self-attentive, multi-context one-class classification method used for unsupervised Anomaly detection on text data. In this approach, sequences of variable-length word embeddings are transformed to a fixed-length representations through a multi-head self-attention mechanism. These representations are trained along with a collection of context vectors. These representations and context vectors are trained together to capture distinct data that still belongs to the known class context.

Chapter 3

Preliminaries

Based on the basic recognition categories of classes [10] [25], we consider the following categories and define as:

Known Class:

The Known Class KC is the class from which the data instances are used to train classifier model. The model see the data from this class and acquire knowledge and learn features about this class of data.

Unknown Class:

The Unknown Class UC is the class which is not present in the training data. The model does not have any knowledge about this class of data.

Based on KC and UC we can further define the closed set classification and open set classification.

Closed Set Classifier

Suppose we have a training dataset with k samples $(x_1, x_2, ..., x_k)$ which belong to n *known* classes $(y_1, y_2, ..., y_n)$

A closed set classifier can be defined as follows:

$$closed(x_{test}) = y_{pred}$$

where $y_{pred} \in \{y_1, y_2, ..., y_n\}$

Open Set Classifier

Similarly an open set classifier can be defined as follows:

$$open(x_{test}) = y_{pred}$$

where $y_{pred} \in \{y_1, y_2, ..., y_n, y_{unknown}\}$, where $y_{unknown}$ is a class not seen during the training. The special case $n = 1$ is called one class classification.

Most of the work we see in the text classification is done under a closed set assumption. The drawback with this approach is, if a document of an unknown class appears during testing, the classifier will wrongly classify it into one of the already seen classes. To avoid this problem, we study open set classification, in particular the one class classifier problem.

Testing one class classifiers with multi-class datasets

For a given set of instances that belong to classes $C_1, C_2, ..., C_n$, in a traditional classification task, the classifier is trained with data from all of the classes. Given a new instance, the classifier tries to classify the new instance into one of the classes $C_1, C_2, ..., C_n$.

In contrast, in **one class classification** scenario, we consider one class C_i from the set of classes $C_1, C_2, ..., C_n$. A classifier K_i treats C_i(training class) as KC and all other classes as *UC*. When a new instance is given to K_i, it identifies if the new instance belongs to C_i or not.

Hypothesis

From the above mentioned definitions and to verify our methodology, we formulate the following hypothesis:

Hypothesis H1: An ensemble approach combines a number m of base classifiers $K_{b,i}$ ($i = 1, ..., n$ and $b = 1, ..., m$). The first hypothesis of this paper is that the classification accuracy of the classifiers $K_{b,i}$ can be surpassed by an ensemble approach.

Hypothesis H2: By relaxing the assumption of one class classifier that the information about unknown class is not available during model building, we formulate the hypothesis that a neural network based ensemble will out perform majority voting using the extra information about the context of the unknown class.

The base classifiers still remain the same but the behaviour of ensemble approach is changed by additional information about the unknown class.

Chapter 4

Theoretical Background

Here we discuss about the concepts starting with natural language processing, neural networks followed by word vector representations(word embeddings) and various techniques to perform embeddings. Then we explain all the theoretical detail about support vector machines, isolation forests and auto encoders along with the dimensionality reduction techniques and ensemble methods that are necessary for building our model.

4.1 Natural Language Processing

Natural language can take any form like text, speech, etc. It is the most common medium of communication and information exchange. Natural language processing (NLP) is a sub-field of computer science that deals with building computational algorithms to automatically analyze and represent human language. NLP-based systems have enabled a wide range of applications such as Google's search engine, Apple's voice assistant Siri and Amazon's Alexa. NLP is also useful to teach machines the ability to perform complex natural language related tasks such as text classification, machine translation, named entity recognition and dialogue generation.

4.2 Artificial Neural Networks

Artificial Neural Networks[26] are inspired by nature, basically from neural networks in the brain. It is an abstraction of biological neural network. Plenty of research occurred in the area of artificial neural networks. Those rapid research works unfolded many different capabilities of these kinds of networks, and gave multiple variants of these networks. A basic neural network architecture can be seen in the figure 4.1. Neural

Networks are developed as an attempt to exploit the architecture of the human brain to perform tasks that conventional algorithms can solve.

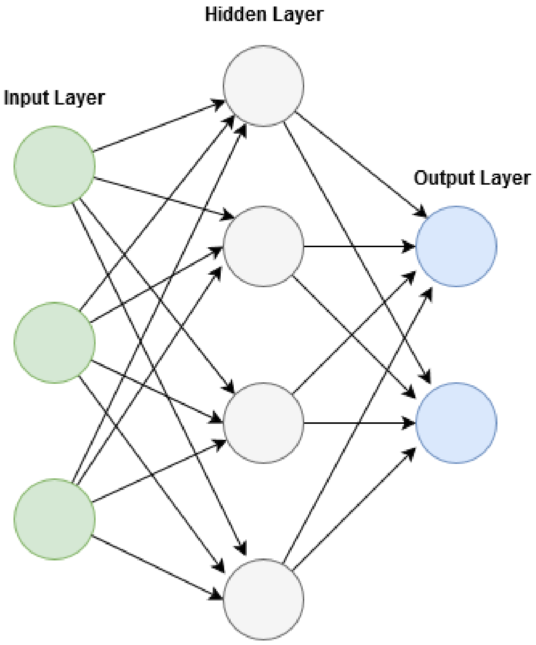


FIGURE 4.1: Artificial Neural Network basic architecture with input, hidden and output neurons

4.2.1 Components

A typical neural networks has the following components:

- **Neurons**: They receive input, combine the input with their internal state (known as activation) and an optional threshold using an activation function, and produce output using an output function. The inputs can be any external data, such as numbers, images and documents.
- **Connections and weights**: The network consists of connections, the connection takes the output of one neuron and provide it as input to another neuron. Each connection is assigned a weight and a neuron can have multiple input and output connections[27].
- **Activation Function and bias**: The activation function computes the input to a neuron from the outputs of previous neurons and their connections as weighted sum and a bias is added to the result. The activation function provides a smooth transition as input values change.

4.2.2 Working

Artificial neural networks use different layers of mathematical processing. An artificial neural network contains neurons anywhere from dozens to millions that are arranged in a sequence of layers. The input layer receives information from various data sources. The information goes through one or more hidden neurons from the input neuron. The input information is transformed into an intermediate state by hidden neuron which can be used by the output neuron. The network starts learning about the information once the data flows through each neuron. The network responds to the information that it was given and process the information at the output neuron.

Neural networks need to be trained with huge amount of information through training for learning. For example, to teach an ANN on how to differentiate a cat from dog, the training set should be thousands of dog images which the network would begin to learn. Once it is trained with the significant amount of data(dog images in this case), the network will try to classify the future data. During the training period, the machine's output is compared to the ground truth data label. If they are the same, then the model learning is finished. If it's incorrect, it uses back propagation to adjust its learning by going back through the layers to tweak the weights.

4.3 Text Embeddings

Word embedding is a feature representation and learning technique in natural language processing (NLP), where individual words are represented as real-valued vectors in a predefined n-dimensional vector space.

There are several methods to generate word embedding which includes neural networks, probabilistic models, dimensionality reduction techniques, etc. Word embedding has proved to be one of the most important discoveries for the performance of deep learning methods on solving natural language processing problems in recent years. It enables machine learning models that rely on vector representation as input instead of text input. These representations preserve semantic and syntactic information on words, and improves performance in almost every NLP task.

Many word representations were proposed using well-known techniques like Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA). We will discuss a few of word embeddings that are prominent over time and relevant for this thesis work.

4.3.1 Word2vec

Word2vec method uses neural networks to generate distributed representation of the words. The Neural Network Language Model (NNML) [28], is a neural architecture developed in 2003, was a very influential work. Over the last two decades, many other models were proposed. But, two very simple log-linear models proposed by Mikolov et al. [29] proved to be a revolutionary work that outperformed all other complex models and, most importantly reduced time complexity.

The main intuition behind Word2Vec [29] proposed by Harris et [30] is words that occur in similar contexts will have similar meaning and training a model with this premise has proven to be surprisingly effective. Models in word2vec are trained using gradient descent and back propagation. Figure 4.2 shows abstract representation of two models.

The two models are described in two sections as follows:

(1) Continuous Bag-of-Words Model

In this method,the context of each word is taken into account and try to predict the word corresponding to the context. One hot encoding is used for the input word and target word. Output error is calculated and compared to the one hot representation of target word. In this process, the vector representation of target word is learned. This model contains three layers: an input layer, a projection layer and an output layer. This model can be compared with the feed-forward neural network where the nonlinear hidden layer is removed and the projection layer is shared for all the words.

(2) Continuous Skip-gram Model

This model is very similar to Continuous Bag-of-Words Model however this model, instead of predicting the current word based on the context, in a sentence the model tries to maximize classification of a word based on another word. The goal of skip-gram is to learn probability distribution of words which are present within a given distance (context) of the input word. Then we choose output layer to represent that probability distribution. The performance and quality of the model depends on the context range and can be increased by increasing the range. But, range increase increases time complexity as distant words are less likely to be related to the center word. Simple vector addition of words show an interesting and meaningful results. For example, vector addition result of ”Germany” and ”capital” is close to ”Berlin”. This interesting property is depicted in [31].

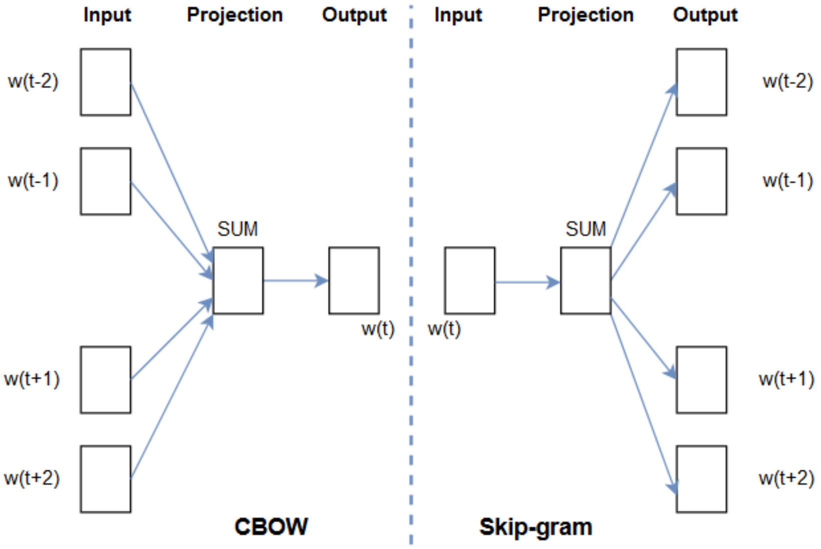


FIGURE 4.2: The CBOW predicts the current word based on the context words. The Skip-gram model predicts context words given the current center word [29]

4.3.2 GloVe

GloVe, Global vectors for word representation, is an unsupervised learning approach developed at Stanford. The key idea is to generate embeddings by aggregating global word-word co-occurrence matrix from a corpus. Every unsupervised word representation learning method uses statistical data about word occurrence but the actual hurdle is to generate meaning from these statistics and generate simple vector forms which can represent the meaning.

Pennington et al. [32] presented a new global log bi-linear regression model that combines these following two model families: global matrix factorization and local context window methods. Global matrix factorization is the process where we use matrix factorization technique to perform rank reduction on a large term-frequency matrix. Local context window is the process of learning semantics by passing a window over corpus line-by-line and try to predict surrounding of a word(as previously discussed as Skip-gram model) or a word given its surrounding (continuous bag-of-words model).

For example, let's say we want to establish one concept that distinguishes man from woman, i.e. sex or gender, which may also be equivalently specified by various other word pairs, such as brother and sister or king and queen. The vector differences of the pairs man - woman, king - queen, and brother - sister might all be expected to be roughly equal.This property can be explained by the visualization in Figure 4.3. The model is capable of fast training, it is scalable to huge corpora as well as perform well with small corpus. It utilizes the ability to capture global statistics while simultaneously capturing the meaningful linear substructures. As a result, GloVe performs well on various tasks, for example word analogy, word similarity, and named entity recognition.

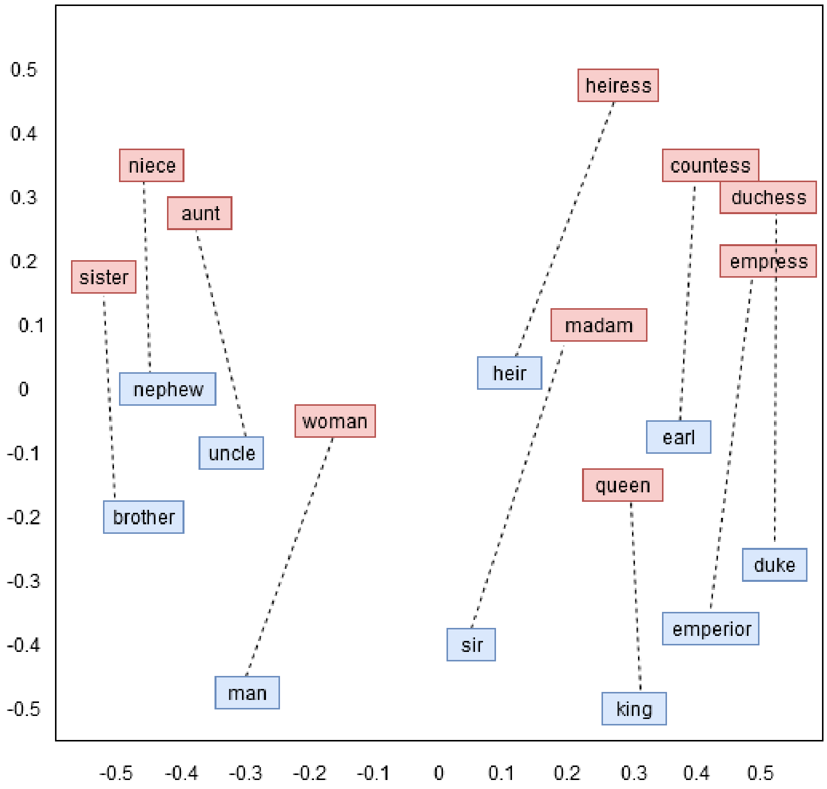


FIGURE 4.3: Glove model [32]

4.3.3 FastText

FastText [33] is a library built by Facebook to learn word representations. It breaks the words into several n-grams. An n-gram is a sub-string formed by grouping n characters at a time from the word itself. For example, the word *here* has the tri-grams: her, and ere. The final word embedding vector will be the sum of all these n-grams for *here*. Given the training dataset, we will have word embeddings for all the n-grams after training the neural network.

4.3.4 ELMo

ELMo[34] is a deep contextualized word representation that models both (1) the syntax and semantics of word use, and (2) how these differ across linguistic contexts. ELMo can significantly improve the performance of the state of the art models for challenging NLP problems, including question answering, textual entailment and sentiment analysis.

ELMo word vectors are computed using a two-layer bidirectional language model (biLM). This biLM model has two layers stacked together. Each layer has 2 passes: forward pass and backward pass(fig 4.4)

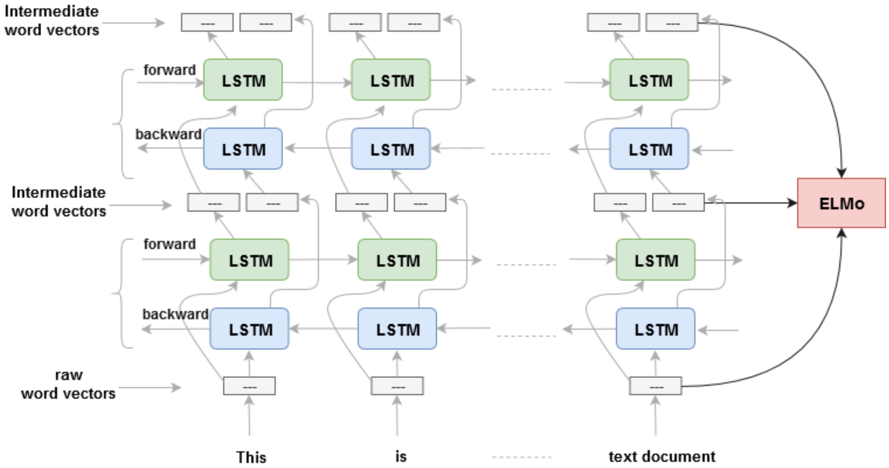


FIGURE 4.4: ELMo Architecture

The information about a word and the context before it is present in forward pass. In backward pass, the information about the word and the context after it is present. The the intermediate word vectors are formed by combining the forward and backward pass information. These intermediate word vectors are fed into the next layer. The final representation is the weighted sum of the word vectors and the 2 intermediate word vectors.

4.3.5 Universal Sentence Encoder

Universal sentence encoder [35] is one of the latest embedding technique to capture the context. Universal Sentence Encoder encodes text into high dimensional vectors that can be used for text classification, semantic similarity, clustering, and other natural language tasks. It comes with two variations - one trained with Transformer encoder and other with Deep Averaging Network. Figure 4.5 shows the architecture of both the variations. The former has higher accuracy but computationally more intensive while the latter is computationally less expensive with little lower accuracy.

The encoder is trained with unsupervised data from Wikipedia, web news,web question-answer pages and discussion forums. This is then augmented with the supervised Stanford Natural Language Inference(SNLI) corpus[37].

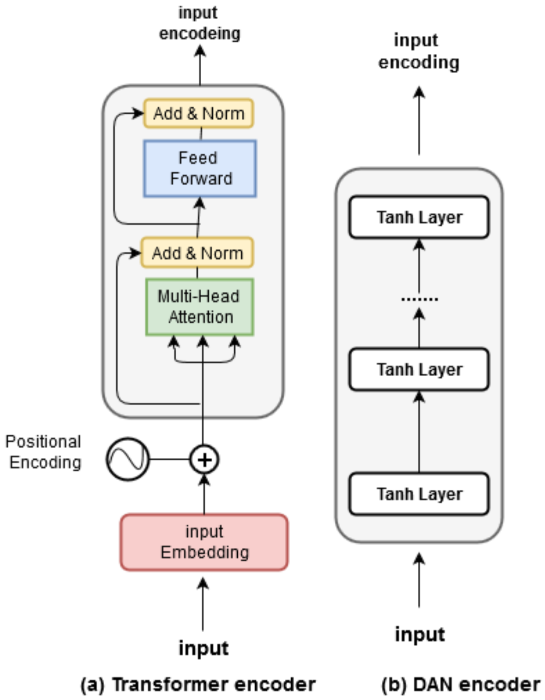


FIGURE 4.5: Variants of Universal Sentence Encoder [36]

4.3.5.1 Transformer

The transformer based model constructs the sentence embeddings using the encoding sub-graph of the transformer architecture. The sub-graph uses attention to compute context aware representations of words in a sentence. These are converted into a fixed length sentence encoding vector by computing the element-wise sum of the representations at each word position. The encoder takes lowercase tokenized string as input and output a 512-dimensional vector as sentence embedding. The transformer based encoder achieves the best overall transfer task performance, but comes with computational cost(time and memory).

4.3.5.2 Deep Averaging Network

This model uses deep averaging network [38] where the input embeddings for words and bi-grams are averaged together and then sent through a feed forward deep neural network to produce the sentence embeddings. This network also takes lowercase tokenized string as input and output a 512-dimensional vector as sentence embedding.

4.3.6 BERT

BERT is a neural network-based technique for NLP that stands for Bidirectional Encoder Representations from Transformers. It learns the context of the word based on it's surrounding words and not just the immediate surrounding word. By just one additional output layer, the pre-trained BERT model can be fine-tuned to create state-of-the-art models for a variety of NLP tasks. During the training phase, BERT learns from both the left and the right side of a token's context which is the reason for calling it bi-directional.

BERT works in following 2 steps: pre-training and fine-tuning. During the pre-training step, the model is trained on unlabeled data over different tasks. In the latter step, the fine-tuning step, the model is initialized with the pre-trained parameters, and these parameters are fine-tuned using labeled data from various tasks. It has multi-layer bidirectional Transformer encoder architecture.

BERT is pre-trained on the text from entire Wikipedia that consists of 2,500 million words and Book Corpus with 800 million words. Tts input representation is constructed by adding the corresponding token, segment, and position embeddings for a given token. With these combinations, without making any major change in the model's architecture, we can easily train it on various of NLP tasks. The computational cost(time and memory) is high with bert compared to the above models discussed.

4.3.7 Infersent

Facebook released a pre-trained sentence encoder called infersent[39] in 2018 that was trained on a natural language inference task and transfers well to other tasks.The architecture consists of 2 parts:

- **The sentence encoder:** takes word vectors and encodes sentences into vectors
- An **NLI classifier** takes the encoded vectors as input and outputs a class among entailment, contradiction and neutral.

It's a bi-directional LSTM[40] network which computes n-vectors for n-words and each vector is a combination of output from a forward LSTM and a backward LSTM that read the sentence in opposite direction. Then a max/mean pool is applied to each of the concatenated vectors to form the fixed-length final vector.

4.3.8 Flair

Flair embeddings [41] are the contextual string embeddings that capture latent syntactic-semantic information. Flair passes sentences as sequences of characters into a character-level language model to form word-level embeddings. These are contextualized by surrounding text. LSTM [40] is used as language modeling architecture.

4.4 Support Vector Machine(SVM)

SVM is a supervised learning model associated with learning algorithms that are used for classification. Given a set of training examples, belonging to one of the two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other. It constructs a hyperplane in a high dimensional space for classification or regression tasks.

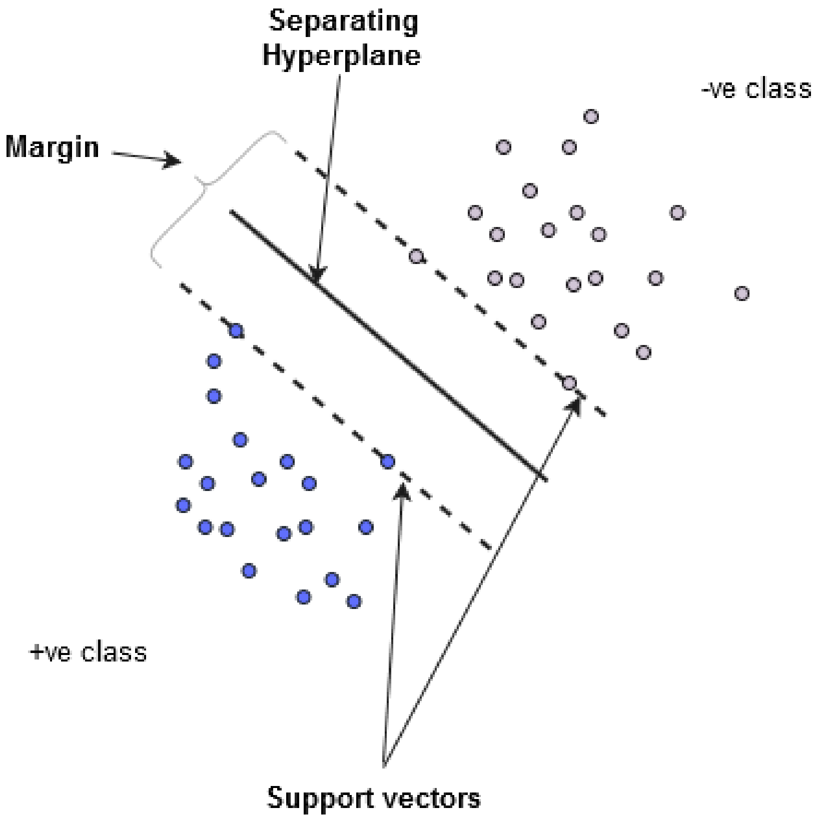


FIGURE 4.6: Representation of SVM for linearly separable dataset [42]

SVMs can create a non-linear decision boundary by projecting the data through a non-linear function to higher dimension space. This means that data points which can't be separated by a straight line in their original space I are projected to a feature space

F where there can be a straight hyperplane that separates the data points of one class from other.

The hyperplane is represented with the equation $w^T x + b = 0$, with $w \in F$ and $b \in R$. The hyperplane determines the margin between the classes. All the data points for the class 1 are on one side, and all the data points for class 1 on the other side. The distance from the closest point from each class to the hyperplane is equal, thus the constructed hyperplane searches for the maximal margin between the classes.

To prevent the SVM classifier from over-fitting with noisy data, slack variables ξi are introduced to allow some data points to lie within the margin, and the constant C¿0 determines the trade-off between maximizing the margin and the training data points in that margin. The objective function of the SVM classifier is to minimize the following:

$$\min_{w,b,\xi_i} \frac{\| w \|^2}{2} + C \sum_{i=1}^{n} \xi_i$$

subject to:

$y_i(w^T(x_i) + b) \geq (1 - \xi_i)$ and $\xi_i \geq 0$ for all i=1,. . . ,n

This minimization problem is solved using Lagarange multipliers(α_i) and the decision function for a data point x becomes:

$$f(x) = sgn(\sum_{i=1}^{n} \alpha_i y_i K(x, x_i) + b)$$

The function K is known as kernel function:

$$K(x, x_i) = \phi(x)^T \phi(x_i)$$

This function gives the same result as the dot product of the vectors in feature space. Popular choices for the kernel functions are linear, polynomial, sigmoidal and Gaussian Radial Base Function.

Figure 4.6 shows traditional binary svm that seperates the given data into 2 classes with a separating hyperplane in the center and the two support vectors. SVM tries to maximize the margin to make minimal errors during classification.

4.5 One Class SVM

According to [43], one class svm basically separates all the data points from the origin (in feature space) and maximizes the distance from this hyperplane to the origin. This results in a binary function that identifies the regions in the input space in which the probability of data occurrence is high. Thus the function returns +1 in a "small" region for input data and -1 elsewhere. The minimization function is slightly different from the general SVM classifier as below:

$$\min_{w,\xi_i,\rho} \frac{\| w \|^2}{2} + \frac{1}{\nu n} \sum_{i=1}^{n} \xi_i - \rho$$

subject to:

$(w.\phi(x_i)) \geq (\rho - \xi_i)$ and $\xi_i \geq 0$ for all i=1,...,n

Here parameter ν sets an upper bound on the fraction of outliers and, it is a lower bound on the number of training examples used as Support Vector.

The decision function for one class svm is given by

$$f(x) = sgn(\sum_{i=1}^{n} \alpha_i K(x, x_i) - \rho)$$

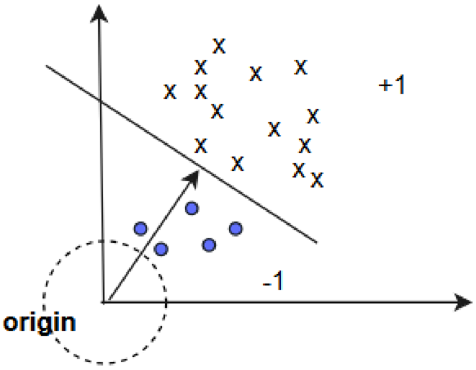


FIGURE 4.7: One-Class Support Vector Machines[44]

Figure 4.7 explains how one class svm works. It treats the origin as the only member of the second class. Then using relaxation parameters, it separates the members of the one class from the origin. Then the standard SVM techniques are employed.

4.6 Isolation Forest

Isolation Forest [2] builds an ensemble of iTrees for a given data set. iTrees are constructed by recursively partitioning the given training set until instances are isolated or a specific tree height is reached whose limit is automatically set by the sub-sampling size. This recursive partitioning of data 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 and thus the height of the branch is used as the outlier score. The path lengths of the data points in the different trees of the isolation forest is averaged as a final step.

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. The Isolation Forest algorithm achieves high anomaly detection performance quickly with high dimensional data.

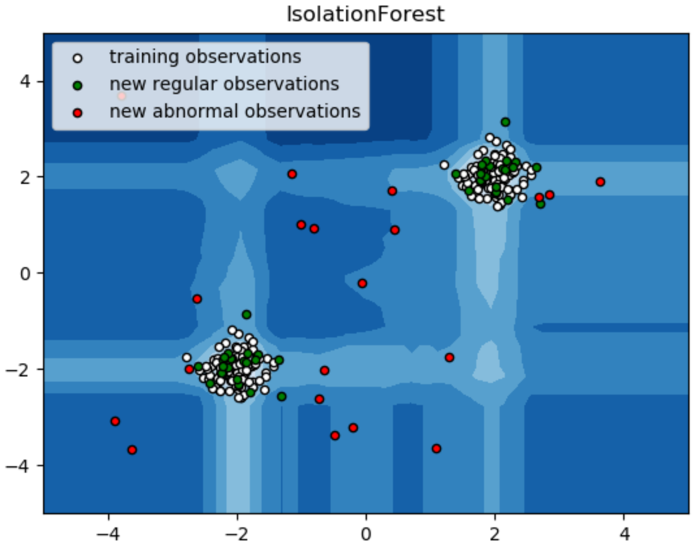


FIGURE 4.8: Isolation Forest from scikit learn [45]

4.7 Dimensionality Reduction

In Natural Language Processing information is often represented in very high-dimensions. This results in problems for practical applications such as storage and computational requirements. When no suitable visualization techniques are available, building an understanding of such data source is also a drawback. Dimensionality reduction techniques can overcome these problems by reducing the number of dimensions while retaining the original information.

Principal component analysis (PCA) is the best known dimensionality reduction technique. PCA aims to detect the correlation between variables. If there is a strong correlation between variables, then the attempt to reduce the dimensionality makes sense. For each of the principal components, the attributed information is explained by the variance. By projecting the information in the feature space onto linear subspaces, the information is transferred into a low-dimensional system retaining as much amount of data as possible.

Following are the steps followed by PCA approach:

- Standardize the data.
- Obtain the Eigenvectors and Eigenvalues from the correlation matrix
- Sort eigenvalues in descending order and choose the k eigenvectors that correspond to the k largest eigenvalues where k is the number of dimensions of the new feature subspace ($k \leq d$).
- Construct the projection matrix W from the selected k eigenvectors.
- Transform the original dataset X using projection matrix W to obtain a k-dimensional feature subspace Y.

4.8 Autoencoder

Autoencoder is a neural network that does not require target class for learning. In simple words, autoencoder is a feed forward neural network that attempts to reconstruct its input and it is used for unsupervised learning. The aim of autoencoder is to reconstruct the features by learn representations. It is used typically for the task of denoising and dimensionality reduction. The basic form of an autoencoder is similar to the multilayer perceptron. Deep networks can be created by stacking Autoencoders. The features generated by an autoencoder can be used for classification, clustering, and anomaly detection.

4.8.1 Mathematical definition of Autoencoder

A traditional autoencoder takes a vector $x \in [0,1]^d$, and maps it to a hidden representation $y \in [0,1]^{d'}$ using a mapping function $f_\theta(x) = s(Wx + b)$ with $\theta = \{W, b\}$.

W is a weight matrix with shape $d' \times d$ and b, bias vector. The hidden latent representation y is then mapped back to a *reconstructed vector* $z \in [0,1]^d$ in input space $g_{\theta'}(y) = s(W'y + b')$ with $\theta' = \{W', b'\}$. The weight matrix W of the reverse mapping may optionally be constrained by $W = W^T$, in which case the autoencoder is said to have tied weights. Each training $x^{(i)}$ is thus mapped to a corresponding $y^{(i)}$ and a reconstruction $z^{(i)}$. The parameters of this model are optimized to minimize the average reconstruction error as follows:

$$\theta^*, \theta'^* = \arg\min_{\theta,\theta'} \frac{1}{n} \sum_{i=1}^{n} L(x^{(i)}, z^{(i)})$$

The loss function used for minimizing the reconstruction error is squared error defined as $L(x,z) = || x - z ||^2$

4.8.2 Architecture

A typical autoencoder consists of 2 networks (Fig. 4.9): encoder and decoder both of which contain an input layer, one or more hidden layers and an output layer. The main difference between autoencoder and multiplayer perceptron is that the output of autoencoder has the same number of neurons as the input. The purpose is to reconstruct its own inputs rather than predicting the target value from given input.

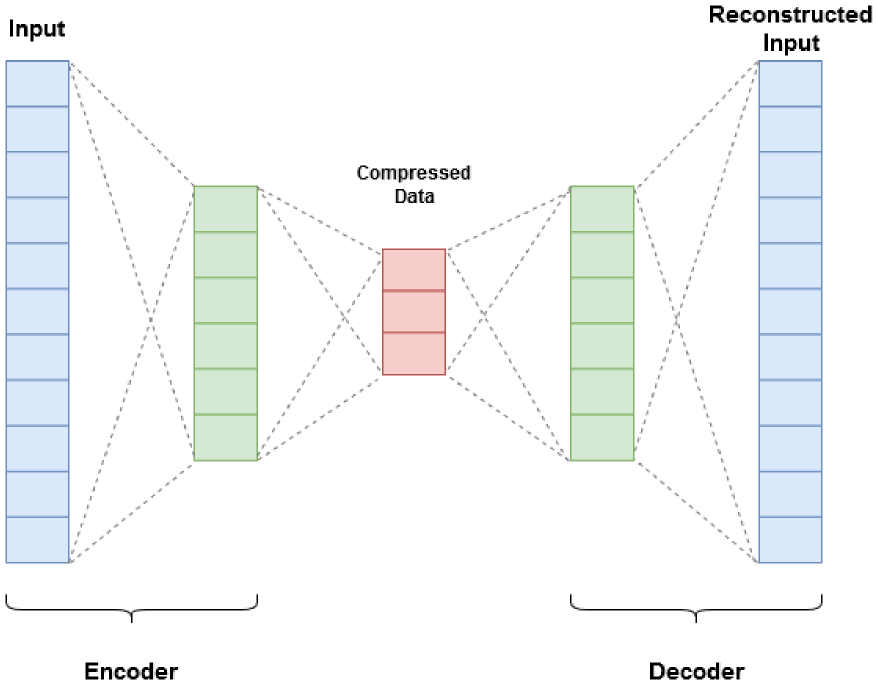


FIGURE 4.9: Autoencoder-architecture

The network structure has connections between layers, but has no connection inside each layer. The training of autoencoder neural network is to optimize reconstruction error using the given samples.

4.9 Ensemble Models

In machine learning, ensemble modeling is a process where multiple models are combined to obtain better model performance, either by using many different algorithms or using different training data sets. The ensemble model combines the prediction of each base model and results in final prediction for the unseen data. The ensemble models reduce the generalization error of the prediction. As long as the base models are diverse, the prediction error of the model decreases when the ensemble approach is used.

Ensemble methods help to minimize noise, bias and variance.

4.9.1 Various Ensemble approaches

The various ensemble techniques are:

- **Max Voting:** The max voting method is generally used for classification problems. In this technique, multiple models are used to make predictions for each data point. The predictions by each model are considered as a ‘vote’. The majority vote is used as the final prediction.
- **Averaging:** In this method, the average of predictions from all the models is taken and use it to make the final prediction.
- **Weighted Average:** This is an extension of the averaging method. Each model is assigned with different weights by defining the importance of each model for prediction.
- **Stacking:** Stacking uses predictions from multiple models to build a new model. This final model is used to predict the outputs of the test data set.
- **Blending:** Blending follows the same approach as stacking but uses only a validation set from the train set to make predictions. The validation set and the predictions are used to build a model.
- **Bagging** Bagging combines the results of multiple models (for instance, all decision trees) to get a generalized result. Bagging uses the subsets of observations

from the original dataset to get a fair idea of the distribution (complete set). The size of subsets can be less than the original set.

- **Boosting** In Boosting, each subsequent model attempts to correct the errors of the previous model. The subsequent models are dependent on the previous model. Boosting decreases the bias error and builds strong predictive models. Boosting has shown better predictive accuracy than bagging, but it also over-fit the training data sometimes.

4.9.2 Advantages of using ensemble methods:

- **More accurate prediction results** The ensemble of models will give better performance on the test data as compared to the individual models in most of the cases. This is because, it extracts the best features from each model and combines them.
- **Stable and more robust model** The aggregate result of multiple models is always less noisy than the individual models. This leads to model stability and robustness.
- Ensemble models can be used to **capture the linear as well as the non-linear relationships** in the data.

Chapter 5

Methodology

5.1 Data Preprocessing

We used various datasets for the model training and evaluation. The data is a group of documents(sequence of sentences). Data pre-processing is the first step before model building. There are many data preparing techniques available. For our experiments we used the following pre-processing techniques as shown in Figure 5.1

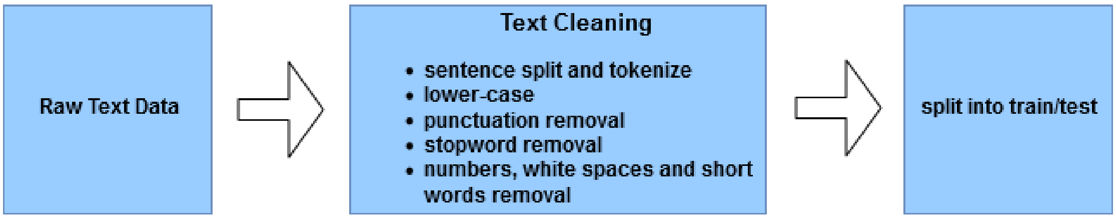


FIGURE 5.1: Flow diagram of basic data preprocessing

5.1.1 Text Cleaning

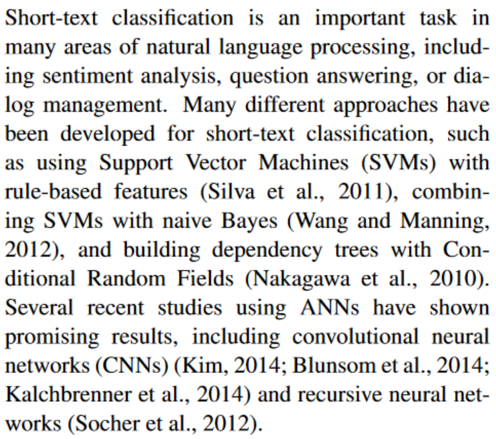
Raw data is pre-processed to obtain a cleaner dataset for the model building. We used nltk library for the pre-processing. The following tasks are performed as part of pre-processing:

i. The text is tokenized into sentences and words.

ii. All the words are converted to lower-case and punctuations are removed.

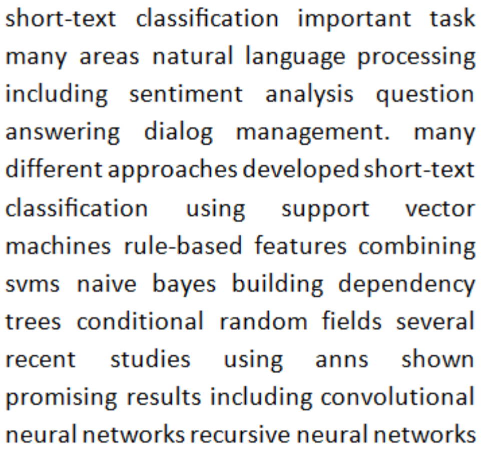
iii. Numbers, white spaces and special characters are removed using regular expressions.

iv. The stopwords, which are the most common words(for e.g. the, is, at, which, and on), are removed. NLTK stopwords list is used for implementation.



Short-text classification is an important task in many areas of natural language processing, including sentiment analysis, question answering, or dialog management. Many different approaches have been developed for short-text classification, such as using Support Vector Machines (SVMs) with rule-based features (Silva et al., 2011), combining SVMs with naive Bayes (Wang and Manning, 2012), and building dependency trees with Conditional Random Fields (Nakagawa et al., 2010). Several recent studies using ANNs have shown promising results, including convolutional neural networks (CNNs) (Kim, 2014; Blunsom et al., 2014; Kalchbrenner et al., 2014) and recursive neural networks (Socher et al., 2012).

FIGURE 5.2: raw text



short-text classification important task many areas natural language processing including sentiment analysis question answering dialog management. many different approaches developed short-text classification using support vector machines rule-based features combining svms naive bayes building dependency trees conditional random fields several recent studies using anns shown promising results including convolutional neural networks recursive neural networks

FIGURE 5.3: clean text

This gives us a representation of the raw text without any noise. Once the text cleaning is done, train and test data are stored as pandas dataframes with respective labels. Sample text before pre-processing and after pre-processing is shown above.

5.2 Text Embedding

In Natural language processing, text embeddings play a major role to build deep learning models. The text embedding converts text into numerical vectors.

5.2.1 Need for converting text into vectors

Lot of research happened in machine learning to convert data into vectors. Once the data is converted into vector, we can tell similarity between the data points by calculating their distance. Techniques like word2vec, GloVe etc., discussed in section 4.3 do that by converting words to vectors. For example, the vector form of 'cat' is more similar to 'dog' than 'eagle'. When it comes to sentences, it is not only important to just convert the words to vectors but also to capture the context of the whole sentence in the vector.

5.2.2 Embeddings used for our models

As there are various models used in building our ensemble model, we experimented with different embeddings and chosen the best embeddings for each model. Below is the list of our embeddings.

- GloVe
- Fasttext
- Universal Sentence Encoder
- Bert
- Infersent
- ELMo
- Ensemble of GloVe + Flair Embeddings

5.3 One Class SVM

This is the first method in our ensemble of classifiers. One class support vector machines is one of the state-of-the-art techniques in one class classification.

The processed text is converted into vector embeddings using various embedding techniques mentioned in 5.2.2. Once the embeddings are created, the features are standardize by scaling to unit variance. As the vector embeddings are high dimensional we used dimensionality reduction technique of principal component analysis. Later at this point one class SVM is applied using 'rbf' kernel. With the above setting the model is trained using train data where the data with one class is fed to the model. The model is tested with the samples from the already seen class during training as well as the unseen class data.

5.4 Isolation Forest

The setting for the model is similar to that of one class SVM. We use embeddings followed by standardization and dimensionality reduction using PCA. After this step isolation forest algorithm is applied and trained on the train data. Model is tested on the test data to find if a new document belongs to the trained class.

5.5 Autoencoders

The autoencoder is trained using the documents that belong to the known class. It tries to learn the representation by minimizing the reconstruction error. If for a test sample the reconstruction error is small, then it belongs to the known class, otherwise it does not belong to the known class.

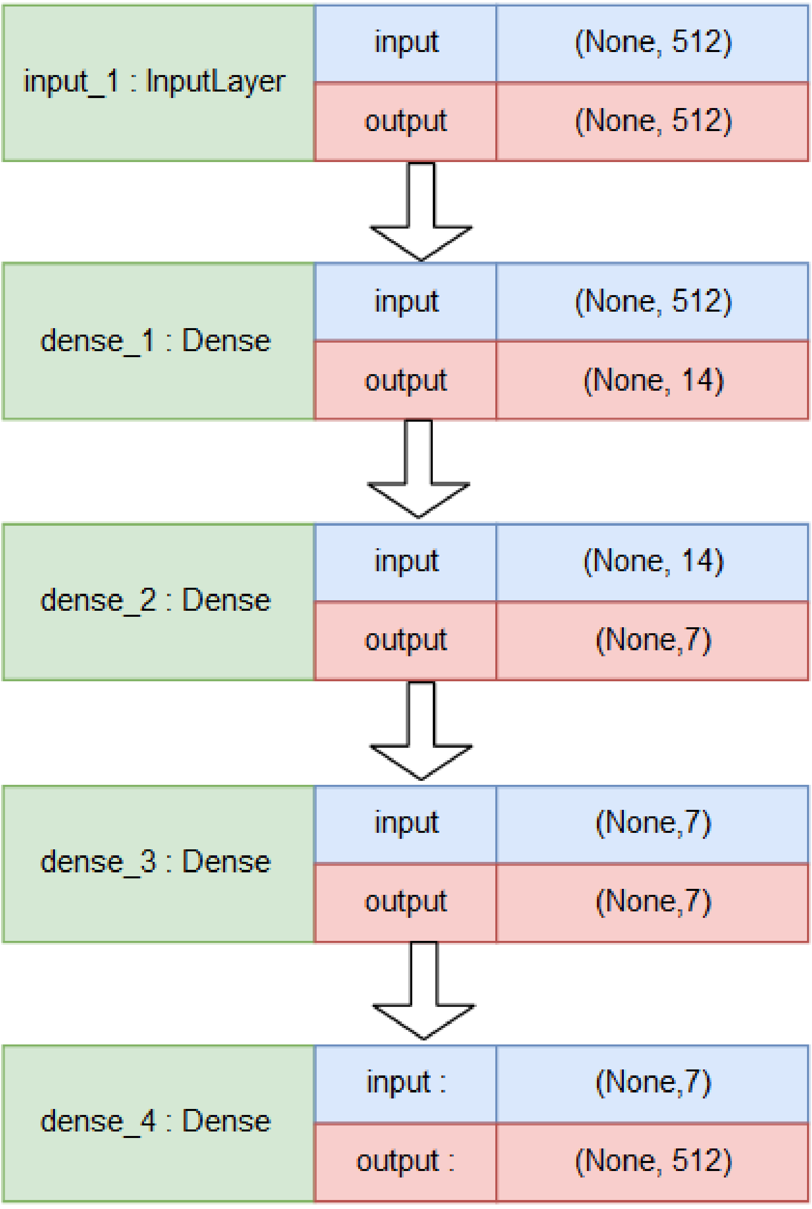


FIGURE 5.4: Autoencoder Layers - Input, Encoder(dense_1, dense_2) and Decoder(dense_3, dense_4)

Once the model is trained, it learns to reproduce feature vectors that represent the known class onto the output layer. Now when a new document is considered during model testing, the autoencoder will predict if the incoming document belongs to the known class or an unknown class through following steps:

i. The new document x_i is sent to autoencoder. At the output layer, the document is reproduced.

ii. Then a reconstruction error ϵ_i is calculated as the distance between the original document vector and the reproduced document vector.

iii. The document is classified as *known* if the reconstruction error $\epsilon_i \leq th$ and *unknown* if the reconstruction error $\epsilon_i > th$ where th is the threshold. The reconstruction error with maximum f1 score is considered as threshold.

In figure 5.4, we describe the number of layers present in both encoder and decoder part of the model and also the type of layers that are present in the model.

5.6 Ensemble Model

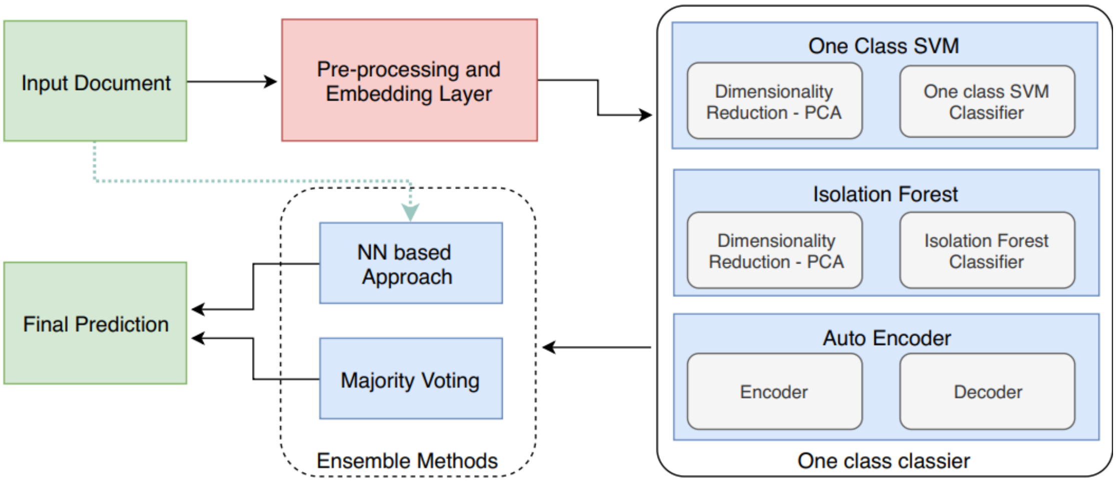


FIGURE 5.5: Architecture - Ensemble Model

Ensemble models provide a technique for generating a better model by combining multiple models. The idea is a single model can be locally optimal or overfit a particular training set, combining multiple models can improve the accuracy. Ensemble models have a set of models as base which accept the same input and predict the output independently. These outputs are combined together to form an ensemble output. In order for ensemble methods to be more accurate than any of its individual members, the base learners have to be as accurate as possible and as diverse as possible. There are various ensemble learning techniques - bagging, boosting, majority voting etc. Here we introduce a novel majority voting based ensemble method for classification. We also build a neural network based ensemble model with relaxed assumptions of one class classifier. Our base models include one class SVM, isolation forest and autoencoder as shown in the architecture(fig. 5.5).

Chapter 6

Experiments

In this section we describe the experiments conducted on the various models.We started with state of the art methods for comparison purposes followed by our methods - autoencoders and ensemble models. We used an ensemble based on majority voting (E_m), which needs to see only the known class, as well as a neural network based ensemble (E_n) with enhanced performance using unknown class instances during ensemble tuning.

6.1 Motivation

We aim to evaluate the performance of our models(autoencoder and ensemble model) with other state-of-the-art systems on the one class text classification task. This evaluates our hypotheses *H1* and *H2* mentioned in chapter 3.

Further, we intend to find the best combination of the classifiers for classifying whether a given test example belongs to the training class or not. We approached by experimenting with the state-of-the-art models for classification. We found that some models perform well in predicting true positives and some models work well in identifying true negatives, so in order to achieve a good prediction, we combined these models to create an ensemble of classifiers that gave good results.

For each experiment, we consider data from one class(out of all the available classes) during model training and data from all the classes(including the training class) during testing. We also compare the results with different embedding techniques discussed in section 4.2. We present our experimental results in a series of tables for the datasets discussed in 6.2.

We represented the models in the tables as M_s for one class svm, M_i for isolation forest, M_a for autoencoder based model, E_m for ensemble model with majority voting and E_n for ensemble model with neural network.

We discuss the dataset in the section 6.2 followed by evaluation metrics in 6.4 and section 6.5 explains about experimental setup, results and evaluation.

6.2 Datasets

We perform one-class classification experiments on the following classification datasets which allow us to quantitatively evaluate the model performance.

- **20 Newsgroups** [46] This data set is a collection of 20,000 newsgroup documents, distributed evenly across 20 different newsgroups. It was originally collected by Ken Lang [47], that become a popular data set for experiments in natural language processing, such as text classification and text clustering. The data set for our experiments consists of 7889 documents of which 590 documents(belongs to same class) are used for training and 7299 for testing. The documents are sorted by date, duplicates and some headers removed.
- **Reuters** [48] The documents in the Reuters-21578 collection of 21578 documents appeared on the Reuters newswire in 1987. The documents were assembled and indexed with categories. We chose a subset of 5162 documents based on the variety of documents from the classes earn, acq, crude, interest, money-fx, trade and ship. Training dataset has 2843 documents and 2319 test documents.
- **Arxiv** [49] This data contains all paper related to the categories AI, ML, CL, NER, CV and quantitative biology published between 2018 Jan to 2018 March. It has approximately 5947 full-text papers of which 4000 papers are used for training and 1947 are considered for testing.
- **WebKB** [50] WebKB is extracted from the World Wide Knowledge Base project of CMU text learning group. These web pages were collected from various computer science universities and manually classified into seven different classes: student, faculty, staff, department, course, project, and other. 2491 documents were considered for our experiments. 1096 documents belongs to the training class and 1395 documents for testing.

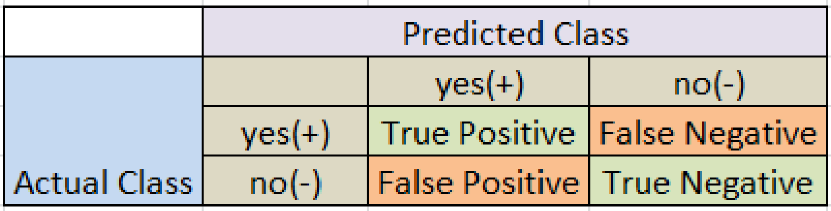
6.3 Implementation details

We used the packages nltk[51], numpy[52] and pandas[53] for data preprocessing. For loading embeddings we used pytorch[54] and flair[41] packages. Built-in functions from sklearn[45] used for model building includes one class SVM, isolation forest and local outlier factory. Standardscaler and PCA are used from sklearn for standardizing and dimensionality reduction. In addition, we implement the autoencoder model with keras[55]. The hardware specifications for the implementation are:

- CPUs: 36
- RAM: 270GB
- OS: Debian GNU/Linux 9

6.4 Evaluation Metrics

Before we proceed with evaluation metrics we define the confusion matrix(see fig. 6.1) followed by the definitions of accuracy, precision, recall and f1 score.



		Predicted Class	
Actual Class		yes(+)	no(-)
	yes(+)	True Positive	False Negative
	no(-)	False Positive	True Negative

FIGURE 6.1: Confusion Matrix

True Positives (TP) - These are the correctly predicted positive values. The values of both actual class and the predicted class are equal and the value is *yes*(+).

True Negatives (TN) - These are the correctly predicted negative values. The values of actual class and predicted class are equal and the value is *no*(-).

False Positives (FP) – When actual class is *no*(-) and predicted class is *yes*(+).

False Negatives (FN) – When actual class is *yes*(+) but predicted class in *no*(-).

False positives and false negatives occur when the actual class contradicts with the predicted class. Based on the above parameters, now we define evaluation metrics as:

- **Accuracy** is defined as the ratio of number of correctly predicted observations to the total number of observations. It is a great performance measure if the dataset is symmetric i.e., if the values of false positive and false negatives are almost same.

$$Accuracy = (TP + TN)/(TP + FP + TN + FN)$$

- **Precision** is defined as the ratio of the number of correctly predicted positive observations to the total predicted positive observations.

$$Precision = TP/(TP + FP)$$

- **Recall** also known as Sensitivity, is the ratio of correctly predicted positive observations to the all observations in actual class: $yes(+)$.

$$Recall = TP/(TP + FN)$$

- **F1 score** It is the harmonic mean of Precision and Recall. This score takes both false positives and false negatives into account and is usually more useful than accuracy.

$$F1Score = 2 * (Recall * Precision)/(Recall + Precision)$$

6.5 Experiment setup and Results

The methodology and architecture of the models are discussed in chapter 4. Here we discuss about the experimental setup for each model and their evaluation using the performance metrics on each dataset.

6.5.1 Model Setup

Among the models we used, one class SVM, Isolation Forest and Autoencoder are considered to be the baselines. We compared our models with these baselines.

One Class SVM

The processed text is converted into vector embeddings, for example, using Universal Sentence Encoder. Once the embeddings are created, the features are standardize by removing the mean and scaling to unit variance. We used principal component analysis for dimensionality reduction followed by one class svm for classification. For this task, setting the number of principal components to $components = min(samples, features)$ gave best model performance which preserves almost 100 percent variance in the data. We experimented with various kernels('linear', 'poly', 'rbf', 'sigmoid') and due to better performance we chose 'rbf' for the final model. With the above setting, the model is trained using training data where the data with one class is fed to the model. The model is tested with the samples from the known class and unknown classes.

Isolation Forest

Isolation Forest isolates observations by selecting a feature randomly and then selecting a split value randomly between the maximum and minimum values of the selected feature. We used the model from sklearn for classification. The setting for the model is same as the setting for one class SVM. We use embeddings followed by standardization and dimensionality reduction using PCA. After this step the isolation forest algorithm is applied and trained on the train data. The model is tested on the test data to find if a new document belongs to the trained class.

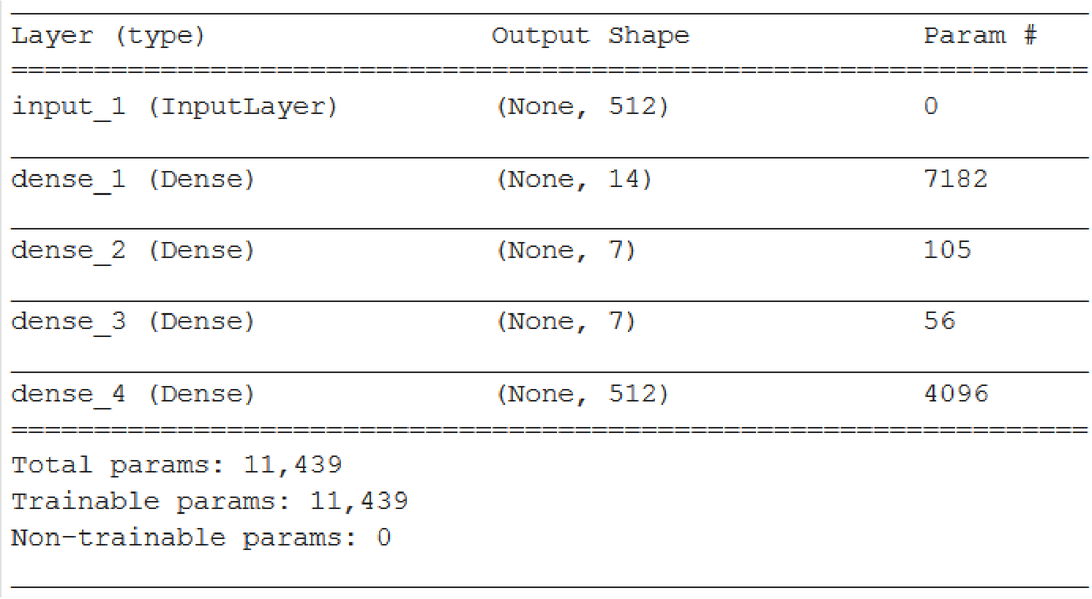
Local Outlier Factory(LOF)

LOF measures the local deviation of density of a given sample with respect to its neighbors. We set the number of neighbours equal to half of the training instances which gave good results and contamination to 0.1.

Auto Encoder

Our Autoencoder uses an input layer of 512 dimension and 4 fully connected layers with 14, 7, 7 and 512 neurons respectively. The first two layers are used for our encoder, the last two for the decoder. Model parameters, output shapes and the layers used for our model are shown in figure 6.2. Tanh and reLu are used as activation functions with Adam optimizer. Mean Squared Error is the loss function. L1 regularization is used during training. The model is trained for 100 epochs with a batch size of 32 samples and save the best performing model to a file which is further loaded for testing. It has total of 11,439 trainable parameters. We experimented with increasing the layers, but it did not give us any better results than this setup.

Majority Voting based Ensemble Layer The predicted class labels(1: KC or -1: UC) from the 3 base models are sent to the ensemble layer where the most frequent label(majority class) is considered to be the final class(known or unknown) to prove our hypothesis *H1*.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 14)                7182
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 105
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 56
_________________________________________________________________
dense_4 (Dense)              (None, 512)               4096
=================================================================
Total params: 11,439
Trainable params: 11,439
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 6.2: Autoencoder parameters

Neural network based Ensemble Layer With slight behavioral change of the ensemble layer we prove hypothesis *H2* using a neural network. The scores from the above 3 base models are sent to a multi-layer perceptron(MLP) along with the embeddings of the documents for the final label prediction. The MLP neural network has 3 hidden layers with 10 neurons in each layer. This method extracts more context about the *UC* by looking at the documents from this class.

6.5.2 Experimental results on different datasets

We perform one-class classification experiments on below datasets which allow us to quantitatively evaluate performance using f1 score.

Table 6.1 show the F1 scores for 20 Newsgroups dataset. Table 6.2 show the F1 scores for Reuters dataset. Table 6.3 show the F1 scores of Arxiv full-text documents. Table 6.4 shows the F1 scores for the dataset webkb. The rows in each table contain the test scores of one-class classifiers trained with the various classes of the datasets.

From our experiments on all the 4 datasets, we made the following observations:

First comparing how well one-class classification works in general for the datasets, in the reuters dataset (Fig. 6.3) overall results are better than with the other datasets, which is due to the very specific content of the classes. In contrast Arxiv (Fig. 6.4) is more difficult to handle because of the length and the broad topics of the texts, and in webkb (Fig. 6.5) the unspecific nature in the contents makes it also difficult to recognize

	USE					**BERT**					**FastText**				
	M_s	M_i	M_a	E_m	E_n	M_s	M_i	M_a	E_m	E_n	M_s	M_i	M_a	E_m	E_n
g1	0.78	0.61	0.71	**0.79**	0.80	0.58	0.65	0.54	0.59	0.81	0.44	0.58	0.34	0.44	0.54
g2	0.67	0.56	0.65	**0.68**	0.80	0.57	0.62	0.55	0.59	0.77	0.32	0.5	0.51	0.47	0.79
g3	0.88	0.59	0.82	**0.89**	0.94	0.52	0.72	0.65	0.66	0.92	0.26	0.52	0.47	0.44	0.87
g4	0.67	0.58	0.67	**0.71**	0.82	0.34	0.6	0.49	0.49	0.79	0.19	0.37	0.49	0.37	0.76
g5	0.70	0.56	0.64	**0.73**	0.84	0.54	0.64	0.56	0.58	0.82	0.34	0.52	0.29	0.35	0.81
g6	0.64	0.57	0.62	**0.65**	0.74	0.55	0.6	0.45	0.55	0.67	0.32	0.51	0.51	0.48	0.70
g7	0.77	0.53	0.73	**0.80**	0.86	0.48	0.65	0.56	0.57	0.85	0.29	0.49	0.1	0.29	0.85
g8	0.66	0.57	0.70	**0.73**	0.85	0.45	0.67	0.53	0.54	0.84	0.26	0.44	0.42	0.38	0.80
g9	**0.72**	0.65	0.70	**0.72**	0.87	0.40	0.6	0.52	0.52	0.70	0.25	0.42	0.46	0.39	0.77
g10	**0.59**	0.54	0.55	**0.59**	0.73	0.42	0.56	0.46	0.47	0.67	0.25	0.42	0.11	0.25	0.65
g11	**0.88**	0.61	0.81	**0.88**	0.92	0.47	0.73	0.61	0.61	0.88	0.23	0.46	0.44	0.4	0.83
g12	0.6	0.55	0.59	**0.62**	0.74	0.5	0.58	0.49	0.52	0.71	0.25	0.44	0.44	0.41	0.72
g13	0.57	0.56	**0.59**	**0.59**	0.70	0.53	0.63	0.53	0.56	0.70	0.26	0.5	0.43	0.42	0.67
g14	0.72	0.56	0.75	**0.77**	0.85	0.42	0.67	0.53	0.54	0.80	0.23	0.41	0.13	0.23	0.79
g15	0.58	0.55	0.56	**0.59**	0.73	0.51	0.63	0.53	0.55	0.72	0.27	0.46	0.47	0.42	0.68
g16	0.79	0.55	0.74	**0.80**	0.87	0.55	0.68	0.62	0.62	0.86	0.31	0.56	0.46	0.46	0.83
g17	0.69	0.49	0.71	**0.73**	0.78	0.50	0.64	0.56	0.56	0.74	0.33	0.51	0.41	0.42	0.72
g18	**0.59**	0.51	0.57	**0.59**	0.71	0.46	0.59	0.55	0.54	0.67	0.25	0.5	0.41	0.41	0.70
g19	**0.61**	0.52	0.54	**0.61**	0.71	0.47	0.6	0.51	0.52	0.68	0.30	0.53	0.36	0.38	0.66
g20	**0.58**	0.54	0.51	**0.58**	0.60	0.52	0.51	0.52	0.52	0.58	0.24	0.52	0.25	0.28	0.60

Table 6.1: Evaluation of Models for 20 Newsgroups

	USE					**BERT**					**FastText**				
	M_s	M_i	M_a	E_m	E_n	M_s	M_i	M_a	E_m	E_n	M_s	M_i	M_a	E_m	E_n
acq	0.72	0.74	0.7	0.73	0.94	0.7	0.74	0.7	0.74	0.95	0.72	0.71	0.73	0.73	0.96
cru	0.84	0.53	0.72	0.84	0.91	0.77	0.72	0.63	0.8	0.92	0.63	0.62	0.42	0.62	0.94
ern	0.72	0.75	0.75	0.77	0.95	0.68	0.8	0.61	0.76	0.97	0.62	0.79	0.32	0.63	0.97
int	0.78	0.57	0.62	0.79	0.86	0.72	0.77	0.54	0.77	0.86	0.54	0.61	0.39	0.53	0.88
mny	0.64	0.53	0.64	0.64	0.80	0.61	0.64	0.54	0.65	0.82	0.54	0.59	0.34	0.54	0.80
shp	0.5	0.6	0.53	0.61	0.76	0.5	0.46	0.47	0.51	0.78	0.53	0.46	0.32	0.46	0.74
trd	0.76	0.55	0.61	0.77	0.89	0.65	0.76	0.62	0.77	0.92	0.69	0.67	0.44	0.67	0.89

Table 6.2: Evaluation of Models for Reuters

the unknown class. The 20 Newsgroups dataset (Fig. 6.6) offers some insight into the influence of specificity of class content on the degree of difficulty of identifying unknown texts: for the very broad class g20(religion),all the models perform worse than for the specific class g3(hockey).

For the two problematic datasets(arxiv and webkb) our autoencoder approach works better than the one-class svm and isolation forest methods, in particular when using USE.

	USE					**BERT**					**FastText**				
	M_s	M_i	M_a	E_m	E_n	M_s	M_i	M_a	E_m	E_n	M_s	M_i	M_a	E_m	E_n
AI	0.59	0.52	0.68	0.6	0.83	0.61	0.67	0.6	0.63	0.82	0.68	0.64	0.55	0.66	0.81
BIO	0.56	0.49	0.66	0.66	0.85	0.49	0.54	0.6	0.59	0.84	0.39	0.54	0.42	0.54	0.80
ESS	0.59	0.47	0.64	0.64	0.80	0.58	0.59	0.6	0.62	0.81	0.49	0.66	0.54	0.6	0.78

Table 6.3: Evaluation of Models for Arxiv dataset

	USE					**BERT**					**FastText**				
	M_s	M_i	M_a	E_m	E_n	M_s	M_i	M_a	E_m	E_n	M_s	M_i	M_a	E_m	E_n
crs	0.59	0.56	0.7	0.69	0.93	0.59	0.64	0.5	0.59	0.91	0.36	0.56	0.36	0.4	0.90
fac	0.61	0.71	0.67	0.68	0.82	0.54	0.51	0.6	0.62	0.83	0.51	0.59	0.6	0.57	0.81
pro	0.49	0.58	0.48	0.5	0.80	0.53	0.46	0.53	0.54	0.77	0.38	0.58	0.51	0.53	0.79
stu	0.41	0.49	0.5	0.51	0.83	0.39	0.55	0.54	0.55	0.86	0.35	0.4	0.38	0.4	0.86

Table 6.4: Evaluation of Models for WebKB dataset

Overall, USE seems to be the most appropriate embeddings for the task in hand, which is understandable considering the fact that recognising sentence similarity is the objective for which USE is trained. However, there are cases for which BERT achieves slightly better results, so that it might be worth while working with this embedding if USE does not give satisfactory result, but one should also bear in mind that BERT causes higher computational costs.

Isolation forest is the worst for USE but the best for BERT and fastText, it is because one-class SVM and autoencoder are good for identifying similarities(which requires looking at sentences) while Isolation forest is good for identifying anomalies.

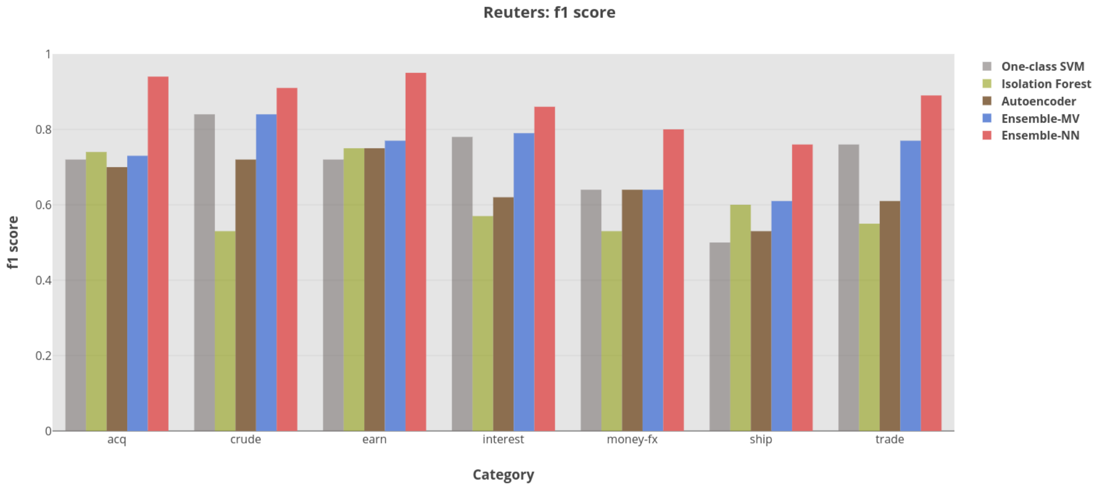


Figure 6.3: Reuters - f1scores for our 5 models

As for the benefits of using ensemble methods, majority voting proves to be beneficial in almost all cases considering lack of any knowledge about the unknown class. The potential of the ensemble is observed in our experiment. This will be a suboptimal solution, as it has no potential of favouring the respective best of the three base models. However neural network based ensemble model performs even much better after relaxing the assumption that the classifier has to be finalised after the initial training with the unknown class. This is because the extra information about the unknown classes progressively improve the ensemble model for better classification. The numbers clearly proves that our hypothesis *H2* holds true. The classifiers OCSVM and autoencoder are better with recall, whereas the precision scores are low.In contrast, isolation forest is better with precision, but low recall. With the ensemble approach using majority voting E_m the precision and recall scores are better compared to individual models. Ensemble with neural network E_n gave even better scores of precision and recall scores clearly outperforming the scores of base classifiers.

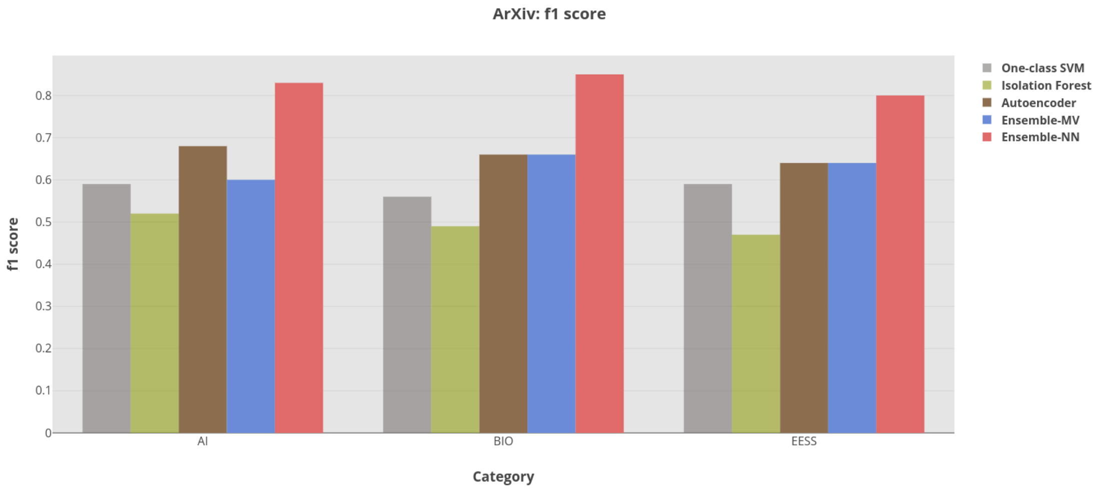


FIGURE 6.4: ArXiv - f1scores for our 5 models

As a general result we find our hypotheses confirmed and recommend the ensemble methods using USE embedding as the first choice for the given task.

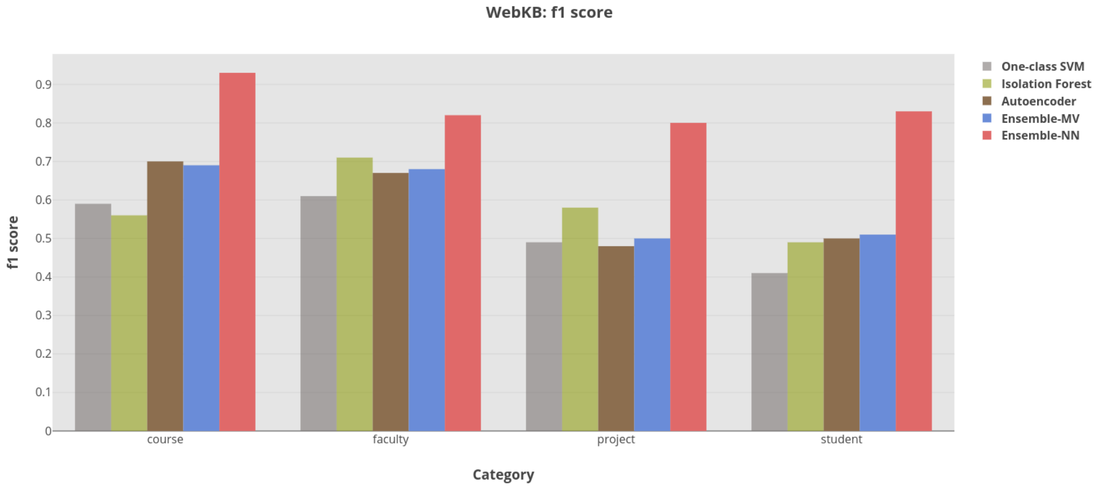


Figure 6.5: WebKB - f1scores for our 5 models

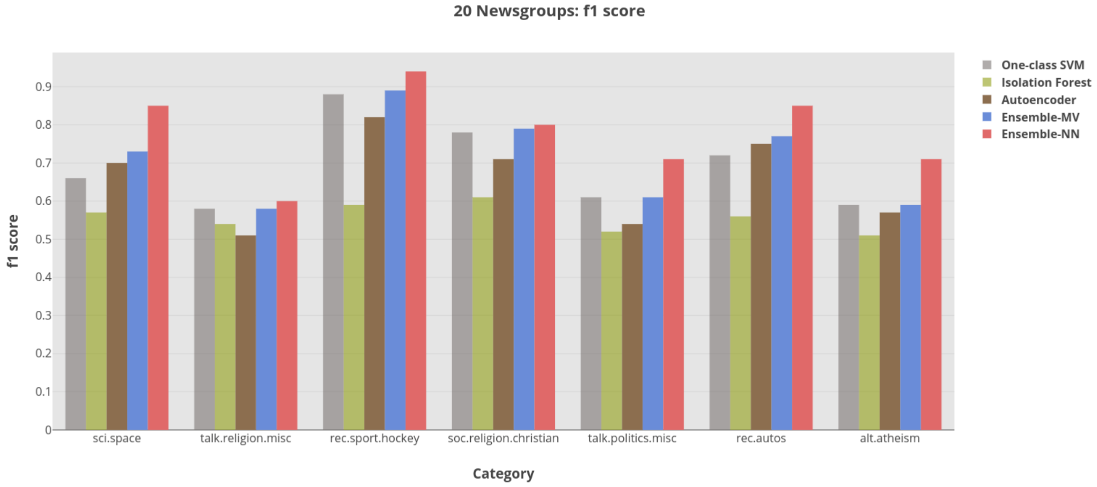


Figure 6.6: 20 Newsgroups - f1scores for our 5 models

Chapter 7

Conclusion and Future work

In this thesis, we evaluated one class classification for documents based on autoencoder and ensemble learning, an approach to increase the performance of state-of-the-art one class classification methods. We worked with three different one class classification methods namely one class SVM, Isolation forest and Autoencoder. On all datasets, we show that autoncoder method is equally and in some cases perform better than the state-of-the-art methods. We built ensemble models using majority voting and neural network approaches. Our ensemble learning methods outperform all individual models. Our results prove that our hypotheses hold true for the task in hand. The main advantage of our framework is flexibility. The modules(embeddings and other models) of the model can be can be easily fine-tuned and extended with other methods.

This thesis widely opens the door for numerous research possibilities. Different models described in this thesis can be extended to achieve elevated milestones. A possible future work will be to extend the model by building a classifier that further classifies the unknown class instances. Another future work for this research is to build a base classifier using generative adversarial networks(GANs) which is very successful in images.

Appendix A

Data collection

Apart from using the existing standard text classification datasets, as part of this thesis, we developed a tool to extract full-text PDFs from ArXiv. In this appendix we will give a quick overview of ArXiv full-text extraction using pdfminer.

Pdfminer is a text extraction tool for PDF documents. It focuses entirely on getting and analyzing text data. Pdfminer allows one to obtain the exact location of text in a page, as well as other information such as fonts or lines. It includes a PDF converter that can transform PDF files into other text formats (such as HTML).

The following query(url) will return an xml document tree with the URLs and meta data of all the pdf documents that belong to a specific Subject during specific time period:

*http://export.arxiv.org/oai2?verb=ListRecords&from=**2018-01-01**&until=**2018-10-31** &metadataPrefix=arXiv&set=**eess***

Here the user specifies the **from** and **until** dates along with the **set**(Subject)

The document URLs are scraped using BeautifulSoup and sent to Pdfminer to extract the useful text and properties. Pdfminer has various tools to extract texts, images, tables, charts and graphs etc. As we are interested in the text mining, we considered the texts and extracted the textual properties from the pdf leaving out the other properties. Headers, footers and references are excluded from the pdf. We performed text cleaning on the extracted text to obtain useful text.

Attached below is code snippet for **full-text pdf extraction**:

```
from pdfminer.pdfparser import PDFParser
from pdfminer.pdfdocument import PDFDocument
from pdfminer.pdfpage import PDFPage
from pdfminer.pdfinterp import PDFResourceManager
from pdfminer.pdfinterp import PDFPageInterpreter
from pdfminer.layout import LAParams
from pdfminer.converter import PDFPageAggregator
from pdfminer.layout import LTTextBox
from io import BytesIO
from urllib.request import urlopen, Request
import urllib
import json
import pandas as pd
from nltk.corpus import stopwords
import requests
from bs4 import BeautifulSoup

def scrape_arxiv(path):
    """
        Extract the required data from the arxiv pdf and stores as json

        Args:
                path: URL of pdf to be scraped

        Returns: Pair of
                1. True/False whether something was found at that URL
                2. JSON structured output of analyzed document,
                or None if not of expected structure

        """
    outstring = ’’
    try:
        text = urlopen(Request(path)).read()
        if ’PDF unavailable for’ in text.decode(errors=’replace’):
            return False, None
        else:
            memory_file = BytesIO(text)
            parser = PDFParser(memory_file)
            document = PDFDocument(parser)
            # Check if the document allows text extraction. If not, return None.
            if not document.is_extractable:
                return True, None
            rsrcmgr = PDFResourceManager()
            laparams = LAParams(line_margin=0.4, detect_vertical=True)
            device = PDFPageAggregator(rsrcmgr, laparams=laparams)
            interpreter = PDFPageInterpreter(rsrcmgr, device)
            # Process each page contained in the document.
            for page in PDFPage.get_pages(memory_file):
                interpreter.process_page(page)
                layout = device.get_result()
                for element in layout:
                    if isinstance(element, LTTextBox):
                        box_props = get_LTText_Props(element)
                        if box_props[’text’].strip() == ’References’:
                            return True, outstring
                        else:
                            if len(pre_process(box_props[’text’])) > 140:
                                outstring = outstring + box_props[’text’]
            return True, outstring
    except urllib.error.HTTPError:
        # If URL does not exist return None
        return False, None
```

Appendix B

Models

Attached below is code snippet for one class svm, isolation forest and autoencoder models:

```
from sklearn.svm import OneClassSVM
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.preprocessing import StandardScaler
import seaborn as sns
from sklearn.metrics import (confusion_matrix, precision_recall_curve, auc,roc_curve)
from keras.models import Model, load_model
from keras.layers import Input, Dense
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras import regularizers
from sklearn.neural_network import MLPClassifier


# OCSVM model
def oneclass_svm(dataset, kernel, nu):
    svm = OneClassSVM(kernel=kernel, nu=nu).fit(dataset)
    return svm

# Isolation Forest
def isolationForest(dataset, rng):
    isolationforest = IsolationForest(behaviour='new', max_samples=100,
    random_state=rng, contamination='auto').fit(dataset)
    return isolationforest

def autoencoder_model(X_Test, Y_Test):
    input_dim = X_train.shape[1]
    print('Input dimension: ',input_dim)
    encoding_dim = 14
    nb_epoch = 100
    batch_size = 32

    input_layer = Input(shape=(input_dim, ))
    encoder = Dense(encoding_dim, activation="tanh",
                    activity_regularizer=regularizers.l1(10e-5))(input_layer)
    encoder = Dense(int(encoding_dim / 2), activation="relu")(encoder)
    decoder = Dense(int(encoding_dim / 2), activation='tanh')(encoder)
    decoder = Dense(input_dim, activation='relu')(decoder)
    autoencoder = Model(inputs=input_layer, outputs=decoder)


    autoencoder.compile(optimizer='adam',
                        loss='mean_squared_error',
                        metrics=['accuracy'])

    checkpointer = ModelCheckpoint(filepath="model_news20.h5",
                                   verbose=0,
                                   save_best_only=True)
    tensorboard = TensorBoard(log_dir='./logs_reuters',
                              histogram_freq=0,
                              write_graph=True,
                              write_images=True)

    history = autoencoder.fit(X_train, X_train,
                        epochs=nb_epoch,
                        batch_size=batch_size,
                        shuffle=True,
                        validation_data=(X_Test, X_Test),
                        verbose=1,
                        callbacks=[checkpointer, tensorboard]).history
```

Attached below is code snippet for various **document embedding** techniques used:

```
from flair.embeddings import FlairEmbeddings,ELMoEmbeddings, WordEmbeddings
from flair.embeddings import DocumentPoolEmbeddings, FastTextEmbeddings ,Sentence, XLNetEmbeddin
from sentence_transformers import SentenceTransformer
import tensorflow as tf
import tensorflow_hub as hub

# Embedding using Universal Sentence Encoder
def embed_module(module):
    with tf.Graph().as_default():
        sentences = tf.placeholder(tf.string)
        embed = hub.Module(module)
        embeddings = embed(sentences)
        session = tf.train.MonitoredSession()
    return lambda x: session.run(embeddings, {sentences: x})

def use_embeddings():
    train_data_list = []
    test_data_list = []
    module_url = "/opt/notebooks/embedding_model/"
    # Import the Universal Sentence Encoder's TF Hub module
    embed = embed_module(module_url)
    train_data_list = embed(final_train['text'].tolist())
    test_data_list = embed(final_test['text'].tolist())
    val_data_list = embed(final_val['text'].tolist())
    return train_data_list, test_data_list, val_data_list
```

```
# Bert
def bert_embeddings():
    train_data_list = []
    test_data_list = []
    model = SentenceTransformer('bert-base-nli-mean-tokens')
    train_data_list = model.encode(final_train['text'].tolist())
    test_data_list = model.encode(final_test['text'].tolist())
    return train_data_list, test_data_list

# Flair - GloVe - XLNet - FastText
def other_embeddings(embd):
    sess = tf.InteractiveSession()
    train_data_list = []
    test_data_list = []
    if embd == 'glove':
        print('Starting Glove Embedding...')
        glove_embedding = WordEmbeddings('glove')
        document_embeddings = DocumentPoolEmbeddings(embeddings=[glove_embedding])
    elif embd == 'fasttext':
        print('Starting Fasttext Embedding...')
        fasttext_embedding = WordEmbeddings('en')
        document_embeddings = DocumentPoolEmbeddings(embeddings=[fasttext_embedding])
    elif embd == 'elmo':
        print('Starting ELMo Embedding...')
        elmo_embedding = ELMoEmbeddings()
        document_embeddings = DocumentPoolEmbeddings(embeddings=[elmo_embedding])
    else:
        # init Flair embeddings
        flair_forward_embedding = FlairEmbeddings('multi-forward')
        flair_backward_embedding = FlairEmbeddings('multi-backward')
        glove_embedding = WordEmbeddings('glove')
        # now create the DocumentPoolEmbeddings object that combines all embeddings
        document_embeddings = DocumentPoolEmbeddings(embeddings=[glove_embedding,
        flair_forward_embedding, flair_backward_embedding])
    print('Train embedding Started...')
    for text in final_train['text'].tolist():
        text = Sentence(text)
        document_embeddings.embed(text)
        emb = text.get_embedding().detach().numpy()
        emb = tf.constant(emb).eval()
        train_data_list.append(emb)
    print('Embedded Train data!!')
    print('Test embedding Started...')
    for text in final_test['text'].tolist():
        text = Sentence(text)
        document_embeddings.embed(text)
        emb = text.get_embedding().detach().numpy()
        emb = tf.constant(emb).eval()
        test_data_list.append(emb)
    print('Embedded Test data!!')
    return train_data_list, test_data_list
```

Appendix C

Acronyms

In the experimental results section, we defined few acronyms for the embeddings, models and the categories due to space constraints. Below table has the details about the acronyms:

TABLE C.1: Models Embeddings table

Acronym	Expansion
USE	Universal Sentence Encoder
M_s	Model - One Class SVM
M_i	Model - Isolation Forest
M_a	Model - Autoencoder
E_m	Ensemble Model - Majority Voting
E_n	Ensemble MOdel - Neural Network

TABLE C.2: Reuters categories table

Acronym	Expansion
acq	acq
cru	crude
ern	earn
int	interest
mny	money
shp	ship
trd	trade

Table C.3: 20 Newsgroups categories table

Acronym	Expansion
g1	soc.religion.christian
g2	comp.windows.x
g3	rec.sport.hockey
g4	rec.motorcycles
g5	sci.crypt
g6	comp.sys.ibm.pc.hardware
g7	sci.med
g8	sci.space
g9	misc.forsale
g10	sci.electronics
g11	rec.sport.baseball
g12	comp.graphics
g13	comp.os.ms-windows.misc
g14	rec.autos
g15	comp.sys.mac.hardware
g16	talk.politics.mideast
g17	talk.politics.guns
g18	alt.atheism
g19	talk.politics.misc
g20	talk.religion.misc

Table C.4: ArXiv categories table

Acronym	Expansion
AI	Artificial Intelligence
BIO	Biology
ESS	Electrical Engineering and Systems Science

Table C.5: WebKB categories table

Acronym	Expansion
crs	course
fac	faculty
pro	projects
stu	students

List of Figures

List of Tables

Bibliography

[1] Larry M. Manevitz and Malik Yousef. One-class svms for document classification. *J. Mach. Learn. Res.*, 2:139–154, 2001.

[2] Fei Tony Liu, Kai Ting, and Zhi-Hua Zhou. Isolation forest. pages 413 – 422, 01 2009. doi: 10.1109/ICDM.2008.17.

[3] Sridhama Prakhya, Vinodini Venkataram, and Jugal Kalita. Open set text classification using convolutional neural networks. *International Conference on Natural Language Processing, 2017*, Dec 2017. URL `http://par.nsf.gov/biblio/10059464`.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[5] Chuanxing Geng, Sheng-Jun Huang, and Songcan Chen. Recent advances in open set recognition: A survey. *ArXiv*, abs/1811.08581, 2018.

[6] He Zhang and Vishal M. Patel. Sparse representation-based open set recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1690–1696, 2017.

[7] Abhijit Bendale and Terrance E. Boult. Towards open set deep networks. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1563–1572, 2015.

[8] Raghavendra Chalapathy, Aditya Krishna Menon, and Sanjay Chawla. Anomaly detection using one-class neural networks. *ArXiv*, abs/1802.06360, 2018.

[9] Walter J. Scheirer, Anderson Rocha, Archana Sapkota, and Terrance E. Boult. Towards open set recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 35, July 2013.

[10] W. J. Scheirer, L. P. Jain, and T. E. Boult. Probability models for open set recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11): 2317–2324, Nov 2014. doi: 10.1109/TPAMI.2014.2321392.

[11] Abhijit Bendale and Terrance E. Boult. Towards open world recognition. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1893–1902, 2015.

[12] Thomas Mensink, Jakob J. Verbeek, Florent Perronnin, and Gabriela Csurka. Distance-based image classification: Generalizing to new classes at near-zero cost. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35:2624–2637, 2013.

[13] Abhijit Bendale and Terrance E. Boult. Towards open set deep networks. *CoRR*, abs/1511.06233, 2015. URL `http://arxiv.org/abs/1511.06233`.

[14] Nathalie Japkowicz, Catherine Myers, and Mark Gluck. A novelty detection approach to classification. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, pages 518–523, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1-55860-363-8, 978-1-558-60363-9. URL `http://dl.acm.org/citation.cfm?id=1625855.1625923`.

[15] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390294. URL `http://doi.acm.org/10.1145/1390156.1390294`.

[16] Mohammad Sabokrou, Mohammad Khalooei, Mahmood Fathy, and Ehsan Adeli. Adversarially learned one-class classifier for novelty detection. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3379–3388, 2018.

[17] B. Schölkopf, JC. Platt, J. Shawe-Taylor, AJ. Smola, and RC. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7): 1443–1471, March 2001.

[18] David M. J. Tax and Robert P. W. Duin. Support vector data description. *Mach. Learn.*, 54(1):45–66, January 2004. ISSN 0885-6125. doi: 10.1023/B:MACH.0000008084.60811.49. URL `https://doi.org/10.1023/B:MACH.0000008084.60811.49`.

[19] Larry Manevitz and Malik Yousef. One-class document classification via neural networks. *Neurocomputing*, 70:1466–1481, 03 2007. doi: 10.1016/j.neucom.2006.05.013.

[20] Geli Fei and Bing Liu. Breaking the closed world assumption in text classification. In *HLT-NAACL*, 2016.

[21] Lei Shu, Hu Xu, and Bing Liu. Doc: Deep open classification of text documents. In *EMNLP*, 2017.

[22] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1181. URL `https://www.aclweb.org/anthology/D14-1181`.

[23] Abhijit Bendale and Terrance E. Boult. Towards open world recognition. *CoRR*, abs/1412.5687, 2014. URL `http://arxiv.org/abs/1412.5687`.

[24] Lukas Ruff, Yury Zemlyanskiy, Robert A. Vandermeulen, Thomas Schnake, and Marius Kloft. Self-attentive, multi-context one-class classification for unsupervised anomaly detection on text. In *ACL*, 2019.

[25] Chuanxing Geng, Sheng-Jun Huang, and Songcan Chen. Recent advances in open set recognition: A survey. *ArXiv*, abs/1811.08581, 2018.

[26] David Kriesel. *A Brief Introduction to Neural Networks.* 2007. URL `availableathttp://www.dkriesel.com`.

[27] Maysam Abbod, James Catto, Derek Linkens, and Freddie Hamdy. Application of artificial intelligence to the management of urological cancer. *The Journal of urology*, 178:1150–6, 11 2007. doi: 10.1016/j.juro.2007.05.122.

[28] Chen Liang Prasenjit Mitra† C. Lee Giles Wenyi Huang, Zhaohui Wu. A neural probabilistic model for context based citation recommendation. 2003.

[29] Tomas Mikolov, G.s Corrado, Kai Chen, and Jeffrey Dean. Efficient estimation of word representations in vector space. pages 1–12, 01 2013.

[30] Zellig S. Harris. Distributional structure. doi: 10.1080/00437956.1954.11659520.

[31] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[32] Jeffrey Pennington, Richard Socher, and Christoper Manning. Glove: Global vectors for word representation. volume 14, pages 1532–1543, 01 2014. doi: 10.3115/v1/D14-1162.

[33] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *CoRR*, abs/1607.04606, 2016. URL `http://arxiv.org/abs/1607.04606`.

[34] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.

[35] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder. *CoRR*, abs/1803.11175, 2018. URL `http://arxiv.org/abs/1803.11175`.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL `http://arxiv.org/abs/1706.03762`.

[37] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. *CoRR*, abs/1508.05326, 2015. URL `http://arxiv.org/abs/1508.05326`.

[38] Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. Deep unordered composition rivals syntactic methods for text classification. In *Association for Computational Linguistics*, 2015.

[39] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. *CoRR*, abs/1705.02364, 2017. URL `http://arxiv.org/abs/1705.02364`.

[40] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

[41] Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/C18-1139`.

[42] Esperanza García-Gonzalo, Zulima Fernández-Muñiz, Paulino Jose Garcia Nieto, Antonio Sánchez, and Marta Menéndez. Hard-rock stability analysis for span design in entry-type excavations with learning classifiers. *Materials*, 9:531, 06 2016. doi: 10.3390/ma9070531.

[43] Bernhard Schölkopf, Robert Williamson, Alex Smola, John Shawe-Taylor, and John Platt. Support vector method for novelty detection. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 582–588, Cambridge, MA, USA, 1999. MIT Press. URL `http://dl.acm.org/citation.cfm?id=3009657.3009740`.

[44] Hany Alashwal, Safaai bin deris, and Razib Othman. One-class support vector machines for protein protein interactions prediction. *Int J Biomed Sci*, 1, 01 2006.

[45] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[46] 20 newsgroups dataset. URL `http://people.csail.mit.edu/jrennie/20Newsgroups/`.

[47] Ken Lang. Newsweeder: Learning to filter netnews. In Armand Prieditis and Stuart Russell, editors, *Machine Learning Proceedings 1995*, pages 331 – 339. Morgan Kaufmann, San Francisco (CA), 1995. ISBN 978-1-55860-377-6. doi: https://doi.org/10.1016/B978-1-55860-377-6.50048-7. URL `http://www.sciencedirect.com/science/article/pii/B9781558603776500487`.

[48] Reuters-21578 dataset. URL `http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html`.

[49] Arxiv homepage. URL `https://arxiv.org/`.

[50] Webkb homepage, 2017. URL `http://www.cs.cmu.edu/~webkb/`.

[51] Edward Loper and Steven Bird. Nltk: The natural language toolkit. In *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*, 2002.

[52] S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30, March 2011. ISSN 1558-366X. doi: 10.1109/MCSE.2011.37.

[53] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[54] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[55] François Chollet et al. Keras. `https://keras.io`, 2015.