HYPERPARAMETER STUDIES ON WORD EMBEDDING

Odinukwe Gerald C
Matric Number: 79145

Master Thesis

In partial fulfillment of the requirements for the degree of Master of Science (M.Sc.) in Computer Science

Course Area:	Computer Science
course of studies:	Master in Computer Science
Study Year:	TODO
Supervisors:	Prof. Dr. Christin Siefert
	Prof. Dr. Michael Granitzer

Contents

List of Figures

List of Tables

Abstract

In this thesis, I will perform experiment on text embedding algorithm in order to find out the influence of hyperparameters on the accuracy of a model as well as the interrelation of the different hyperparameters on the accuracy of a model on a sentiment analysis task. There are many hyperparameters one could use in training a model and therefore there is need to know the right ones or the combination that will give a good training performance.
The insight that will be gotten from this experiment will help researchers in setting hyperparameters for text embedding projects thereby reducing the amount of optimization time and also the search space of the hyperparameter.

Acknowledgment

1 Introduction

Building applications in natural language processing domain require the tuning of machine learning algorithms which involves adjusting the values of the hyperparameters in order to achieve good model accuracy. The performance of machine learning tasks largely depends on the settings of the hyperparameters. There is no best model or algorithm in NLP task but getting the best accuracy out of a model depends on how well the hyperparameters is tuned. This means that the right values needs to be assigned to these hyperparameters for a good result to be achieved. The process can be time consuming and also requires experience in setting this hyperparameter values.
This work experiments with three hyperparamater optimization methods. Which will help users in understanding the relative importance of a single hyperparameter on the accuracy of a classification model.

1.1 Motivation

Hyperparameters, are parameters for the learning algorithm in machine learning, for example the learning rate for gradient descent, number of trees in a random forest, the hidden layers in a neural network. Hyperparameter optimization is regarded as the process finding the most optimal hyperparameters for the machine learning algorithm. These hyperparameters influences the performance of the machine learning model and therefore must be carefully tuned. Since a machine learning algorithm can have many hyperparameter, programmers always have issue with selecting the hyperparameters that will perform well on their machine learning algorithm. A common approach in achieving optimal hyperparameter is the hand tuning, this is a trial and error method in which the programmer continues to twist the values of these hyperparameters until he gets a training accuracy he is satisfied with. Another approach is the grid search, in this method we choose a set of possible values for our parameters and create variable to store our model accuracy for each set and making a nexted loop for every value of the hyperparameters to be tried between each other. The classifier will then be initialized inside the loop with the hyperparameters at that iteration which will be trained and scored then compare it with the best score , if it is better, the values will be updated ac-

cordingly. The random search approach uses random combination of ranges of values for a number of iteration that we have set. The bayesian approach takes the advantage of the information our model learned during the optimisation process [43]. This works by picking a prior belief of how our hyperparameter will behave and then search the parameter space by enforcing and updating our prior believe based on the ongoing measurement.

- Time consuming
  These methods can be very time-consuming, and can make training of machine learning model to take so long.

- So many Hyperparamaters
  The conventional hayperparameter optimisation methods is normally an exhaustive search in the hyperparameter space and usually requires training of model multiple times. This is infeasible for a large data set or complex hyperparameter space.

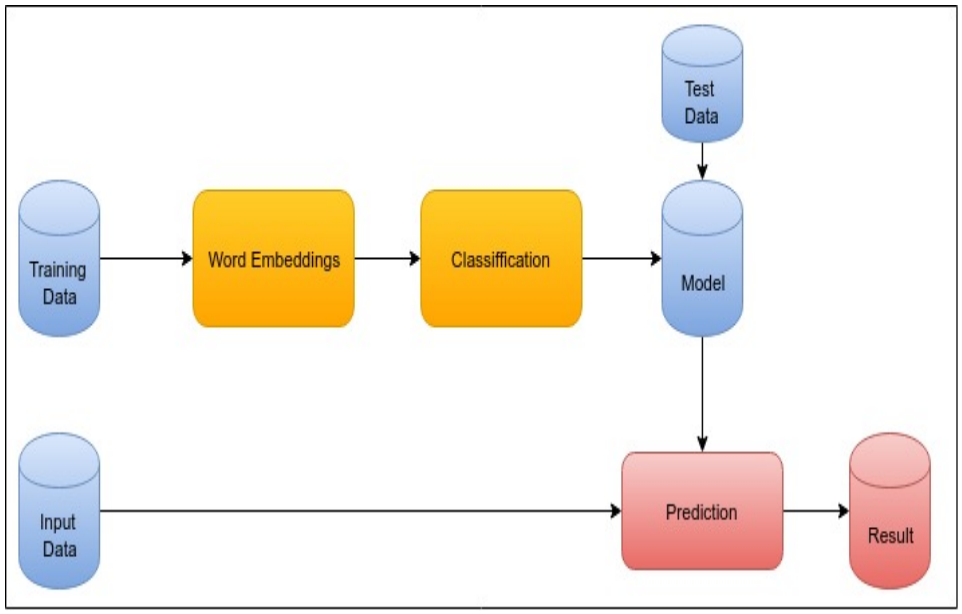


Figure 1.1: Machine learning Steps

1.2 Objectives

The purpose of this master thesis is to try to find an answer to the following questions: What is the influence of the hyperparameters on the accuracy of the model of a sentiment analysis task using word2vec for embedding? What is the interrelation between hyperparameters? How do the optimal hyperparameter found by grid search, Bayesian search, Population based search differ?

To answer these questions, I will be carrying out experiments by systematically changing the hyperparameter settings of text embedding algorithms, this will enable us to have insights about the influence and interrelation of hyperparameters on the model performance on a sentiment analysis task when using text embedding features.

1.3 Structure of the report

This report is divided into 5 main chapters. Following this introductory chapter comes, "Background & Related Work," (Chapter 2), where we have a look at some other works related to this work. The 3rd chapter discusses the implementation of our work along the tools and technologies we used. And in Chapter 4 is the Evaluation chapter where we discuss the result of the experiment. The Discussion and Conclusion is in Chapter 5.

1.4 Approach

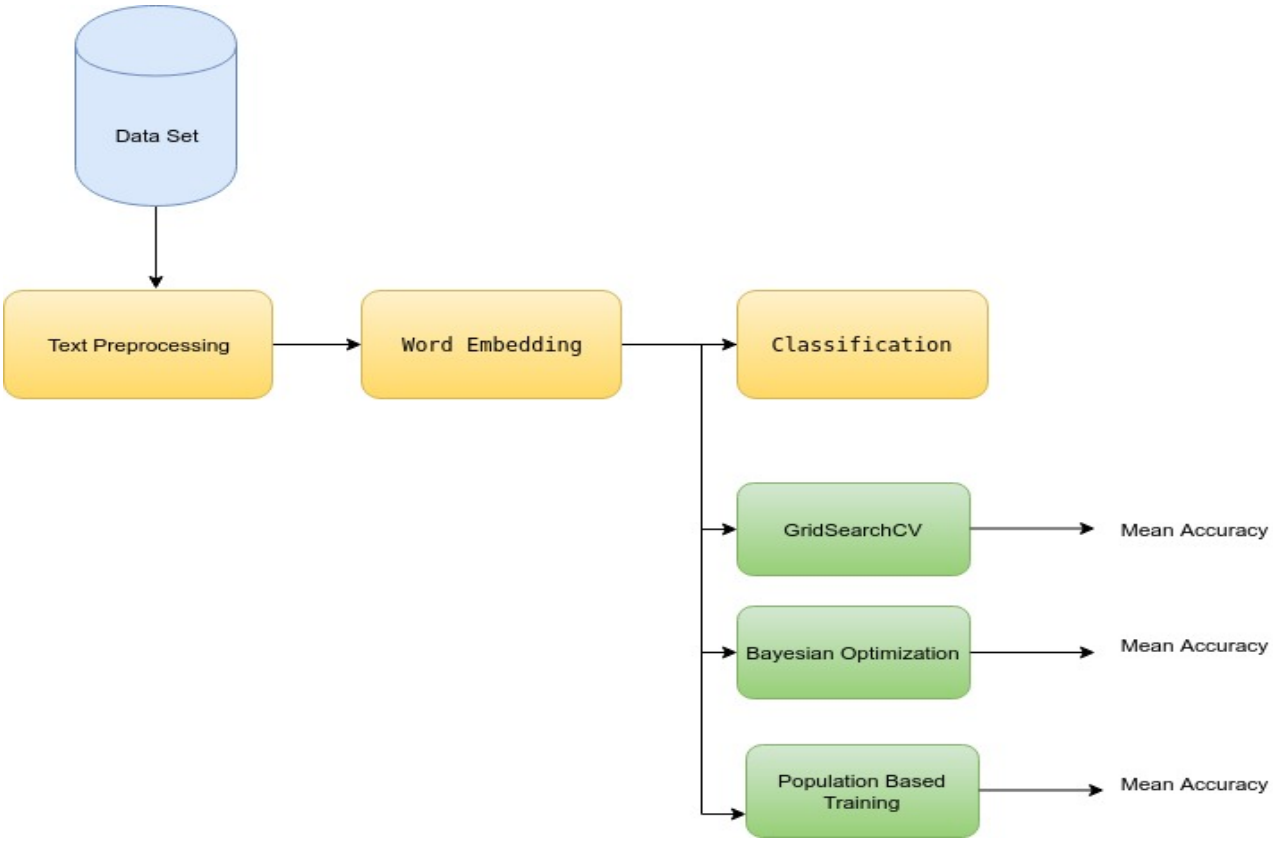


Figure 1.2: Approach

Since the goal of this work is to find the influence of hyperparameters on the model performance. To access the performance of the model, I have chosen

the task of sentiment analysis. Sentiment analysis using Word2Vec was experimented by, Ahmed B [2]. The overall approach is depicted in Figure 1.2.
First, we have to get the dataset, the Amazon movie review dataset [1] will be used for the classification task and also to building of the embeddings.
The Amazon Movie Reviews dataset was downloaded from http://snap.stanford.edu/data/web-Movies.html. This text file was of 3GB zipped file and approximately 9GB when unzipped which has more than 8 million reviews. The movie reviews was uploaded by a professors from Stanford University J. McAuley and J. Leskovec. The Dataset has been randomly split and only 1gb is used for this work.
The data provided in the original file was in the following format:

review/productId	B00006HAXW
review/userId	A1RSDE90N6RSZF
review/profileName	profileName
review/helpfulness	9/9
review/score	5.0
review/time	1042502400
review/summary	Pittsburgh - Home of the OLDIES
review/text	I have all of the doo wop DVD's and this one is as good or better than the 1st ones. Remember once these performers are gone, we'll never get to see them again. Rhino did an excellent job and if you like or love doo wop and Rock n Roll you'll LOVE this DVD !!

Table 1.1: Example review from the Amazon Movie Data Set

Regarding the data format, below are the details of each column name :

- **Product/Product Id:** This is a unique ID generated by Amazon and assigned to a review for the purpose of identification.
- **User Id:** An ID that is linked to the user user who posted the review.
- **Profile Name:** The name of the user that made the review.
- **Helpfulness:** This column specifies the number of users that found a review useful.

- **Time:** This column identifies the time of the review, the date and time at which the review was posted.
- **Summary:** The summary of the movie.
- **Text:** The review of a product from a user is stored in the review/text column. This column has some unstructured text which are the reviews from the users.
- **Score:** The most important for each review is the "review/score" variable which represents a rating which is between 1 to 5.

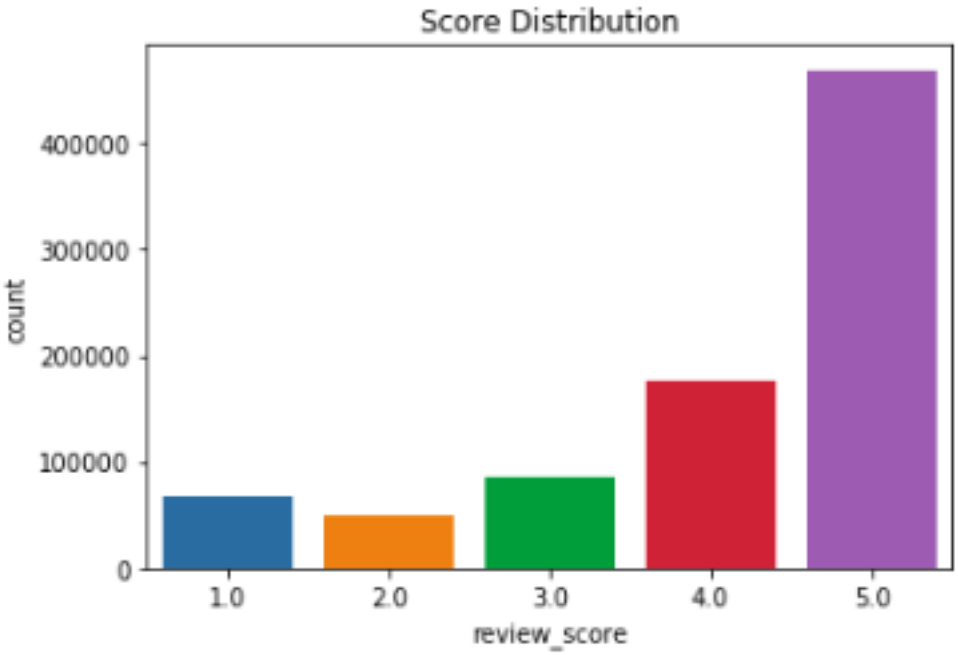


Figure 1.3: Distribution of the Review Score

Preprocessing will be performed on this dataset to make it clean.
Real-world data are mostly

- Incomplete: lack certain attributes of interest, or contains only aggregated data
- Noisy: can contain errors or outliers;
- Inconsistent: can have discrepancies in codes or names.

Data preprocessing is a method is used in resolving this kind of issues. Also, content generated by user on the web is rarely present in a form that is usable for learning. Which makes it important to normalize the text through application of some a series of pre-processing steps. The preprocessing steps that

will be used in this work includes normalization and tokenization.

The tokenization step splits longer strings of text into smaller tokens. In tokenization task, large chunk of text can be split to become sentences while sentences can also be split to become tokenized words. Tokenization can also be referred to as text segmentation or lexical analysis. The texts in the amazon movie review dataset which will be used later for the sentiment analysis task was tokenized into words.

Normalization is referred to a series of related tasks which are done in order to put all text on a level playing field: This includes making all text to be on same case (upper or lower), removing punctuation, changing numbers to their equivalents in words, and so on. Text normalization can mean carrying out a number of tasks, but for this work I have made all the words to have same case, and also removed unmeaningful characters like quotes, punctuations.

Word2vec is used in calculating the feature representation of the dataset which is governed by the model hyperparameters of word2vec e.g., dimensionality of the feature vectors and learning parameters e.g., number of iterations (epochs) over the corpus.

Then a classification is carried out, classification is the act of finding out the class or a set of categories where a new observation belongs, i.e. text categorization, fraud detection, optical character recognition, machine vision (e.g., face detection), natural-language processing. This depends on the training set of data which contains observations that its category membership is known. Classification is regarded in machine learning as an instance of supervised learning, e.g. to learn where a training set of observations that are correctly identified is available. The classification model in this work is trained using the k-nearest neighbor classifier, of which the classier is governed by it's hyperparameters such as k.
The grid search is used on the hyperparameter vector to test different values of the hyperparameters. The grid search uses cross-validation and represents the results as (macro-averaged) accuracy.
Also the Bayesian optimization and population based training hyperparameter optimization method is also used. This is simply to find out which of the optimization method that finds the optimal hyperparameter configuration.

2 Background and Related works

Mechanization has stepped into every walk of life in today's world. This process of automation has created an unquenchable thirst for the methods which ensure successful and meaningful interaction between humans and machines. Technical advancements in the fields of machine learning and artificial intelligence seem to be promising in meeting this demand. Natural language processing (NLP) is one such paradigm wherein an artificial system is trained to understand and interpret the natural human language [3].

Word embedding forms an integral part of NLP wherein the words or phrases in the speech are modelled in terms of vectors comprising of real numbers. Word2vec [4] and and Glove [5] are the most widely used techniques to convert the words into meaningful vectors. An extension of word2vec technique, doc2vec [6], is used when the necessity is to convert the documents, not just words, into vectors suitable for machine processing.

Embeddings from Word2vec can encode semantic linearities such as king - man + woman = queen . Word2vec training can be done using two different architectures, cbow and skip-gram. In cbow the input word vector is predicted by the vectors of the surrounding context words. While in the skip-gram architecture, the vectors of the context word is predicted by the vector of the input word. In both cases the number of context words is a hyperparameter of this model,in the sense that vectors of words which are in the same context will become similar whereas vectors from that of randomly sampled words (i.e., negative sampling) will become dissimilar [66]. Rezaeinia et al. [7] suggests the use of pre-trained word embeddings trained on different huge set of data. In their work, they proposed a novel technique called Improved Word Vectors (IWV) to increase the accuracy of such pre-trained word embeddings used in sentiment analysis models. Yet another technique to successfully meet the demands of sentiment analysis associated with Twitter reviews is Sentiment Specific Word Embedding (SSWE) method. Tang et al. [8] proposed this method in which the emotional information prevalent within the text is encoded in terms of word vectors. Eggensperger et al. [9] proposed to use of surrogate models to optimize hyperparameters of the real-world networks. The idea was to reduce the run-time and the compatibility issues associated with actual hyperparameter optimization which forces the researches to use syn-

thetic test functions. Although a simple look-up table kind of approach can address this issue, it is suitable only for small, finite hyperparameter spaces. In their work they extend look-up technique to make it suit arbitrary, high dimensional hyperparameter space by training regression model using ample of hyperparameter configurations evaluated during inexpensive offline phase. This regression model is then used to predict the performance of a new configuration. They tested a wide range of regression techniques used and found that the surrogate benchmarks based on random forests were the best as they resulted in perfect surrogates for small hyperparameter space and yielded the surrogates very similar to the reach benchmarks for high complexity hyperparameters. These surrogate benchmarks, namely, synthetic test functions and look-up tables, can be used for debugging and unit testing. Their approach was shown to be faster and less space consuming.

Bardenet et al [10] also tried to address the issue of hyperparameter optimization by resorting to surrogate-based optimization technique. In the proposed method, they combined surrogate-based ranking with the optimization techniques meant for surrogate-based collaborative tuning (SCOT). This novel collaborative method of tuning hyperparameters based on past memory was validated on AdaBoost and MLPs. The results obtained were seen to be better than those which would result from the use of common tuning strategies including surrogate-based optimization. Optimization of several hyperparameters based on the computation of gradient of a model selection criterion was presented by Bengio [11]. The work relied on back-propagation through a Cholesky decomposition technique to arrive at the gradient. They used implicit function theorem to derive a formula for second-order derivative composing hyperparameter gradient for which made it a quadratic training criterion.

In the work of Keerthi et al [12], an attempt was made to tune the hyperparameters by minimizing a smooth performing validation function called smoothed k-fold cross-validation error. They used non-linear optimization technique to deal with the gradient of validation function which dealt with the hyperparameters. Empirical results presented showed the proposed technique to be effective as it would identify near-optimal set of hyperparameters within relatively small number of gradient computations demanding just a minor fraction of training time. The results were shown to be effective in comparison to that of grid search even for the model with just two hyperparameters. Nevertheless, when using this technique, one needs to choose the set of hyperparameters wisely so as to ensure good results.

Loshchilov and Hutter [13] proposed the technique of Covariance Matrix Adaptation Evolution Strategy (CMA-ES) to optimize the hyperparameters of the deep neural networks. Their particular interest was towards optimizing con-

tinuous hyperparameters like integers spanning over a wide range. CMA-ES method was shown to perform better than Bayesian optimization technique for large function evaluation budgets wherein the number of hyperparameters requiring to be optimized was more than 10. The dataset considered was that of MNIST on which CMA-ES was implemented to execute in a parallel-fashion.
Analysis of Boolean functions led Hazan and Klivans [14] to develop a novel technique to optimize the hyperparameters of artificial neural networks. The developed algorithm comprised of applying compressed sensing techniques for orthogonal polynomials in an iterative manner. It is advantageous as it can be parallelizable with great ease as it requires only uniform sampling of hyperparameters. Apart from this, it is relatively simple and fast in terms of execution time. The results obtained by employing this technique to CIFAR-10 dataset showed better performance in comparison to Hyperband, Random search and Spearmint optimization tools.
Gaussian process which well-performs with respect to non-parametric regression was shown to be suitable even for hyperparameter optimization in the work of Blum and Riedmiller [15]. Their work used Rprop, a gradient-based optimization algorithm to train the neural network datasets by resorting to maximum likelihood function. They compared their novel technique with conjugate gradient or L-BFGS-B methods by comparing log-likelihood value with the number of gradient evaluations. On synthetic data, Boston housing data and $CO2$ data, L-BFGS-B converged faster than Rprop while Rprop took-over for the cases dealing with real world data. The results obtained by using conjugate gradients were not in-par with these in any of the cases considered. In addition, the authors claim that their technique based on Rprop is easier to implement when compared to other two methods.
Pedregosa [16] proposed the method of optimizing the continuous hyperparameters by extracting the information concerned with inexact gradient function. This methodology allowed the author to update the hyperparameters even before the full convergence of the model parameters. The work even gives an account of conditions necessary for global convergence of the proposed methodology based on regularity conditions of the functions involved and the summability of the errors. The empirical results presented prove the efficiency of this technique with respect to the estimation of $l2$- regularized logistic regression constant and kernel ridge regression constant.
The use of backward propagation was suggested [17] to accomplish hyperparameter tuning. By doing so, they aimed at addressing the issue of computing exact gradients of cross-validation performance dealing with entire hyperparameter set. The gradients so computed were used to optimize the hyperparameters like weight initialization distributions, step-size and momentum

schedules and richly parameterized regulation schemes apart from neural network architectures. This method in which the gradients were computed by exactly reversing the dynamics of stochastic gradient descent with momentum was even shown to reduce the demand for memory requirement significantly. Inspired by this work, Franceschi et al [18] attempted to address the issue of hyperparameter optimization by resorting to two alternative techniques namely, reverse-mode and forward-mode hypergradient computation techniques. Their procedure of reverse mode differentiation (RMD) is based on Lagrangian formulation for parameter optimization, similar to the work of Maclaurin et al [19] except the fact that it requires no reversible dynamics. They tried to deal with the issue of space complexity associated with RMD technique by resorting to forward mode hypergradient computation. The results obtained show forward mode optimization methodology to be suitable in two cases. First, when the number of hyperparameters is less in comparison to that of the overall parameters and second while there is a necessity to update hyperparameters in real-time. The experiments conducted on data cleaning and learning task iterations showed the proposed methodology to be effective both in terms of time and convergence rate. In addition to these, they suggested the use of constrained hyperparameter optimization to deal with the noise-prone cases and while discovering the relationships between different learning tasks.
The thesis work of Socher [20] presents an elaborate explanation on the novel technique, Recursive Deep Learning (RDL) which can be used in the domain of NLP. The models in the proposed RDL family are variations and extensions of unsupervised and supervised recursive neural networks (RNNs). RDL tries to effectively model the general representations for the sentences without assuming word order independence. By the virtue of this feature, it was shown to perform effectively when it comes to sentiment analysis, image-sentence mapping, paraphrase detection, relation classification, parsing and knowledge base completion, with relatively low or completely nil manually designed features.
Algorithms like logistic regression and K-means use fixed-length vectors such as bag-of-words or bag-of-n-grams to represent their text inputs. However these kinds of word representations ignore the semantics of the words, data sparsity, high dimensionality and loose word-order which causes different sentences using the same words to have identical representation. To overcome this issue, Le and Mikolov [21] proposed the concept of paragraph vector, an unsupervised algorithm to represent each document by a dense vector trained to predict the words comprising the file. This algorithm learns fixed-length feature representations from variable-length pieces of texts like sentences, paragraphs and documents. This framework was then used to predict the words in a paragraph by concatenating the paragraph vector with several word vector representations

of the paragraph. Here both word vectors, shared amongst the paragraphs, and the paragraph vectors, unique to a given paragraph, were trained using stochastic gradient descent and back propagation techniques. Empirical results presented show the novel method to be better than bag-of-words models when dealing with text classification tasks like Stanford Treebank, upto 30% and incase of sentiment analysis on IMDB datasets around 16%. Tellez and his team [22] proposed a novel multi-propose text classifying technique called µTC to handle difficult cases like informally written text. The approach composed of various easy-to-implement text transformations, text representations and a supervised learning algorithm. This minimalist competitive classifier had the potential to handle the tasks independently of domain and language. In the experiments conducted, µTC was compared with several state-of-art methods on 30 different datasets to assess its performance with regard to the issues like spam detection, topic and polarity classification, authorship attribution and user profiling. Amongst these, µTC performed the best in case of 20 datasets and showed satisfactory results in case of remaining 10.

The method of using word embeddings in NLP succeeded that of using a word-context matrix. The popular work in regard to the first case is the works of Mikolov et al. [23] named skip-gram with negative sampling (SGNS) training method, the one implemented in word2vec package. Levy and Goldberg [24] portrayed SGNS as a weighted matrix factorization technique which implicitly factorizes a shifted pointwise mutual information (PMI) matrix. They claimed the same even for the NCE embedding method proposed in the work of Mnih [25]. Despite of the good results obtained by shifted PMI matrix, it cannot be implemented directly as it is a dense matrix with a considerably high dimension. They tried to tackle this by approximating PMI to result in a relatively sparse positive shifted PMI matrix. Their technique performed slightly better than word2vec on various linguistic tasks while being much better in terms of optimizing SGNS's objective. They also indicate that a simple spectral algorithm based on SVD performs well in comparison to shifted PPMI when it comes to word similarity tasks, but not in case of word analogy tasks.

Generally, relation classification task is accomplished using statistical machine learning whose performance is dependent in the quality of extracted features. These extracted features often lead to propagation errors in the system as they are derived from the output of pre-existing NLP systems. To overcome this drawback, Zeng et al [78] used a Convolutional Deep Neural Network (CDNN) to extract lexical and sentence level features. They skip the complicated pre-processing stage and transform the entire input word token into vectors. Following this, they form the overall extract feature vector by concatenating the lexical level features extracted according to the given nouns with the sentence

level features learned using convolutional approach. These features are then fed into a softmax classifier to predict the relationship between two marked nouns. In this work, they even used position features (PF) to encode the relative distances between the pairs of the nouns for which the relation label needs to be attached. The importance of PF in relation classification was established in the experiments performed on the dataset SemEval-2008, which showed the proposed methodology to be effective in comparison to many other state-of-art techniques.

Another approach to make relation classification independent of both feature representation and kernel design was proposed by Xu et al [26]. The presented design exploits Shortest Dependency Path (SDP) existing between the two target entities within a sentence. The information acquired is then processed in a Long Short Term Memory (LSTM)-based recurrent neural network. This SDP-LSTM method basically combines the positive aspects concerned with SDP, direction implied by the dependency trees and linguistic information. They even employs a new dropout strategy to SDP-LSTM network to deal with the issue of overfitting. The test results performed on SemEval 2010 relation classification task achieved a F1-score of 83.7% as opposed to SVM technique of Hendrickx et al [27], neural network based classification proposed by Socher et al [28], CNN based classification used by Zeng et al [29] and CR-CNN model presented in the work of Santos et al[30] and Feature-rich Compositional Embedding Model (FCM) used by Gormley and Dredze[31] each of which attained 82.2%, 82.4%, 82.7%, 84.1% and 83.0%, respectively.

As the DNNs get introduced into the field of NLP, there has been extensive works carried on to compare the performance of two main DNN architectures namely, CNN and RNN with respect to various NLP tasks. In the work of Vu and his colleagues [32], CNN was reported to be better than basic RNN lacking gating mechanisms for relation classification. They reasoned it to be because of the better ability of CNN to provide complementary information. CNNs were reported to be better than GRU/LSTM for classifying long sentences by the even by the works of Wen and Adel [33]. In addition, Yin et al [34] argued attention based CNN to be better performing in comparison to attention based LSTM for answer selection while Dauphin et al [35] reported fine-tune gated CNN to outperform RNN when modeling long context dependency.

Opposing to these, Arkhipenko and team [36] claims GRU to be a better choice rather than CNN and LSTM in analyzing semantics of Russian tweets. The work of Yin et al [37] compares CNN, GRU and LSTM in efficiency when dealing with various NLP tasks like answer selection, part-of-speech tagging, sentiment or relation classification, question-relation matching in Freebase, textual entailment and Freebase path query answering. They discovered the

RNNs to be better in majority of the cases except in case of recognizing a particular keyphrase. Nevertheless, it appears that the ideal architecture for a particular task is influenced by various aspects like the accuracy associated with the tuning of hyperparameters like layer size [38] hidden size and batch size [37]
. As stated, irrespective of their application, the accuracy of the NLP systems is seen to be heavily dependent on its hyperparameters i.e. better the tuning of hyperparameter, greater will be the accuracy of the model [39]. Although it is possible to hand-tune these hyperparameter values, automatic hyperparameter optimization methods are preferred as the latter ones often ensure better performance [40]. This fact has led to the development of several hyperparameter optimization methods including grid search [41], random search [42], Bayesian optimization [43] and population based training [44].
In grid search technique, the model is built and evaluated on each of the hyperparameters present within the grid of possible hyperparameter space in order to determine which hyperparameter results in the best value. This most commonly used approach is computationally exhaustive as the number of evaluations to be carried-on grows exponentially with an increase in the number of parameters [45]. In spite of being methodological, this technique sometimes underperforms when compared to relatively simple random search technique, wherein the best hyperparameter is selected by evaluating the model on n random uniform points in the hyperparameter space.
One such evidence is provided by the work of Bergstra and Bengio [42] in which random search model produces results in-par with the grid search based experiments of Larochelle et al. [46]. This is because, in random search, the importance associated with each of the hyperparameter would not be the same due to which there would be more trials for good coverage dimensions rather than being equal for all probable dimensions. They also suggest using random search to analyse the effect of a single hyperparameter on the network by varying its value over a set of remaining hyperparameters with fixed random values. The simplicity associated with random search i.e. the use of non-intelligent, randomly selected hyperparameter values is countered by an unnecessary increase in the variance.
Li et al [47] tried to speed-up the process of hyperparameter optimization by modifying the properties of random search technique. They aimed at early termination of poor performing iterations while adaptively allocating the necessary resources. Their procedure, Hyperband, differs from other configuration validation approaches as it makes minimum number of assumptions. This novel technique was shown to be adaptive even for unknown, widely varying convergence rates and to the behavior of validation losses as a function of

hyperparameters. A comparison presented between the proposed Hyperband method with Bayesian optimization techniques show Hyperband algorithm to be faster when dealing with deep-learning kernel-based systems. Their methodology was claimed to be highly parallelizable with minimal cost while being able to work in conjunction with quasi-random methods including Sobol and Latin hypercube apart from simple random search. They also suggested incorporating meta-learning technique into Hyperband methodology to drive its performance to a much higher level.

In Bayesian method, the existing prediction is updated based on the new evidence obtained. As an optimization technique, it uses a previously evaluated point to compute a posterior expectation of what the loss looks like. Snoek et al. [43] introduced a fully Bayesian treatment for expected improvement, and algorithms for dealing with variable time regimes and parallelized experiments. They performed empirical analysis on three challenging problems in different machine learning areas. Their result showed that Bayesian optimization of hyperparameters is faster and more accurate (hence lower in terms of test error) in comparison to human-tuning on CIFAR-10 dataset.

In their work, Thornton and others [48] attempted to use a fully automated approach for choosing a learning algorithm and then to optimize its hyperparameters. They used state-of-art Bayesian technique to optimize wide range of hyperparameters concerned with full range WEKA-implemented classifiers extending over 30 base classifiers, 3 ensemble methods and 14 meta-methods. The new method developed by the authors is named as Auto-WEKA and can easily merge with SMAC-like technique. The results obtained by performing experiments on 10 popular data sets from UCI repository exhibit better performance in comparison to complete cross validation over default hyperparameter setting of 47 classification algorithms.

Hutter et al. [48] extended the method of Sequential Model-based Optimization (SMBO) to overcome its limitations of supporting only numerical parameters and optimizing only target algorithm performance for single instances. This resulted in the generation of two novel techniques: Random Online Adaptive Racing (ROAR) procedure and Sequential Model-based Algorithm Configuration (SMAC) method. In this work, SMBO, an improved Bayesian optimization method, was shown to be superior in performance over other optimization techniques.

Bayesian optimization technique was also used by Klein et al [49] to accelerate the process of training large datasets. Their work used generative model, which learnt while optimization, to validate the error associated with the training data set. This model was then used to explore preliminary configurations on small subsets by extrapolating them to the full dataset. The developed

Bayesian optimization technique was shown to be much better than the state-of-art Bayesian techniques as it traded high information gain about global optimum against the computational cost.

One of the ways which assists the process of hyperparameter optimization is to apply the knowledge obtained from the previous experiments to the new experiment in-hand. This process of transferring information can be achieved by setting an initialization strategy such as the one proposed by Wistuba et al [50]. The specialty of the proposed initialization strategy was the fact that it demanded for no meta-features and was not limited to any prior-evaluated hyperparameter configurations. The novel initialization strategy approximates the response function by a differential plug-in estimator unlike the usual techniques which make use of non-differentiable meta-loss function. This process is then complemented with gradient-optimization technique to achieve better hyperparameter optimization. Experimental results conducted on two meta-sets were used to compare the performance of the novel technique with other existing initialization strategies and were shown to yield promising results.

Another technique for optimizing hyperparameters is the Sequential Model-based Bayesian Optimization (SMBO) method [51]. Their work showed that SMBO performed better than grid search and random search based optimization techniques. SMBO framework consisting of surrogate model, an acquisition function and the initializing technique was extended to include the search for hyperparameter space [50]. This novel approach of theirs was tested on two newly created, publicly available meta-datasets. The extensive number of experiments performed showed that the pruning of hyperparameters improves the performance of the artificial neural network oriented at optimizing the hyperparameters.

An attempt was made to optimize the hyperparameters of the artificial intelligence system by combining general SMBO framework with simple surrogates [50]. The simple surrogate model AHT was used to replace transfer surrogate models with an intention to reduce the run time and space requirement. The empirical results obtained by employing AHT on two meta datasets prove the efficiency of adaptive transfer learning based AHT.

Actually, the performance of Bayesian optimization techniques used to tune the hyperparameters is seen to greatly improve when meta-knowledge is applied to it. In this case, the knowledge acquired by using an algorithm on a particular dataset is used to fasten up hyperparameter search and optimization for another new dataset. Wistuba et al [52] presents a work in this regard wherein the transfer of information occurs in two stages. The initial phase approximates the mapping function between hyperparameter configuration and hold-out validation performances for the prior-known data sets. These approx-

imations are then combined to rank the configurations of hyperparameters for the new dataset in the second stage. The results obtained were shown to be better than other state-of-art techniques.

Application of meta knowledge also seem to reduce the computational complexity involved when SMBO is used to perform on a new optimization problem. The number of evaluations taken to detect high-performance regions by SMBO can be substantially reduced by using meta-learning[54] to search for hyperparameters, which suggests good configurations for new datasets based on its pre-learning. This new technique was developed by Feurer et al[53] and is called Meta-learning-based-Initialization of SMBO (MI-SMBO). This ready-to-go approach of theirs was evaluated on two established SMBO frameworks namely, Spearmint and SMAC on 57 datasets. The results were shown to be prominent on complex combined algorithm selection and hyperparameter optimization problem rather than on low dimensional hyperparameter optimization.

Another set of hyperparameter optimization technique is explore-exploit method known as Population Based Training (PBT). In this method, a member of a particular population, called worker, exploits the information from the worker of another better-performing population. In addition, this worker can even resort to exploration by randomly changing the values of its hyperparameter so as to improve the performance of the model. The work of Jaderberg et al. [44] showed PBT technique to be effective in terms of convergence as well as performance when dealing with deep reinforcement learning problems. They showed PBT technique to be stable over a wide set problem domains viz., deep reinforcement learning, supervised learning for machine translation and training of Generative Adversarial Networks (GANs). The superior performance of PBT in the case of deep reinforcement was established by using it to optimize UNREAL [55] on DeepMind Lab Levels [56], Feudal Networks [57] on Atari games[58] and simple A3C agents for StarCraft II [59].

Biologically inspired Particle swarm optimization (PSO) technique was shown to efficiently explore the solution space in the work of Lorenzo et al [62]. This potentiality of PSO would allow Deep Neural Networks (DNNs) of minimal topology to obtain competitive classification performance over MNIST dataset. The results of the extensive experiments performed showed PSO to be effective in terms of computational resource requirement and better in classification accuracy in comparison to CIFAR-10, and thus proved that PSO would improve the performance of the existing systems.

PSO technique was used to select hyperparameters associated with Least-Squares Support Vector Machines (LS-SVMs) [61]. The method proposed had the potential to search for multiple hyperparameters simultaneously and had

the advantage of being independent of prior knowledge requirement concerned with analytic property of generalization performance measure. They validated their novel technique on various kinds of kernel families considering benchmark datasets. The best and the quasi-best performance was exhibited for Scaling Radial Basis kernel Function (SRBF) and Radial Basis kernel Function (RBF), respectively.

An attempt was made to optimize the hyperparameters of a Least Square Support Vector Regression (LSSVR) model using PSO algorithm [63]. The results obtained were shown to be better than those got from the application of Back Propagation Neural Network (BPNN), Radial Basis Function Neural Network (RBFNN), Generalized Regression Neural Network (GRNN) and Support Vector Regression (SVR) methods.

One drawback which haunts the otherwise best performing automated hyperparameter optimization techniques, in comparison to human oriented optimization, is their computational overhead. The major factor in favor of manual hyperparameter tuning is the fact that humans can terminate the runs which would result in deteriorated performance of the system quickly and effectively in comparison to the automated system. One method to achieve this is by resorting to "Bag-Of-Tricks" technique used in conjunction with stochastic gradient descent (SGD) method as depicted in the work of Montavon et al. [64].

Domhan et al [65] proposed another similar technique to terminate the training of poor-performing artificial neural networks at an early stage. This method was based on a probabilistic model which extrapolated the information regarding the performance of the system from the first part of its learning curve. This novel methodology when used in conjunction with hyperparameter optimization techniques was shown to terminate the bad runs at a better rate. As a result, the speed associated was seen to increase almost by a factor of two when tested on object recognition benchmarks MNIST, CIFAR-100 and CIFAR-10.

On finding and optimizing the hyperparameter of interest, it would be beneficial to know its impact on the model under consideration. One such work which depicts the influence and interrelation of hyperparameters on text embeddings used for text classification task is that of Witt and Seifert [66]. In their study of doc2vec based document classification task, the hyperparameters like window-size and number of negative samples were shown to have negligible influence on model accuracy. Next, they claimed that better performing system can be obtained by resorting to dbow and hierarchical sampling. In addition, all learning parameters were shown to be closely related; while being independent of the model parameters.

Classifiers like K Nearest Neighbour (KNN) are used in performing classification tasks in Natural language processing (NLP)and was used on studying the influence of hyperparameters of doc2vec. The goal of the nearest neighbor classifier is to find an amount of predefined training samples which are closet in terms of distance to a new point , and then predicting of the label from it [79]. The number of samples can be on a local density of points (radius-based neighbor learning) or a constant defined by user (k-nearest neighbour learning) .Metric measures such as standard Euclidean distance is commonly used as the distance of choice. Because Neighbors-based methods simply "remember" all of its training data, they are therefore known as non-generalizing machine learning methods.

Even with it's simplicity, nearest neighbors success has been seen in a large number problems(classification and regression) in which handwritten digits or satellite image scenes is one of them. The neighbors classifier is often works well in situations where boundary of decision is very irregular.

In this work, we aim at examining the influence of hyperparameters over the accuracy of the model. In addition, we also intend to explore the interrelationship between the hyperparameters of the model working on sentiment analysis task. Adding to these, we examine how the hyperparameters found using grid search, Bayesian search and population search differ from each other. The importance of this experiment lies in the fact that choosing a right set of hyperparameters amongst a large set of others is the key factor in obtaining a good training performance. The outcome of this work benefits the researchers by aiding them in optimizing the values of the hyperparameters in text embedding projects at a faster rate by reducing the search space associated with each of them.

3 Implementation

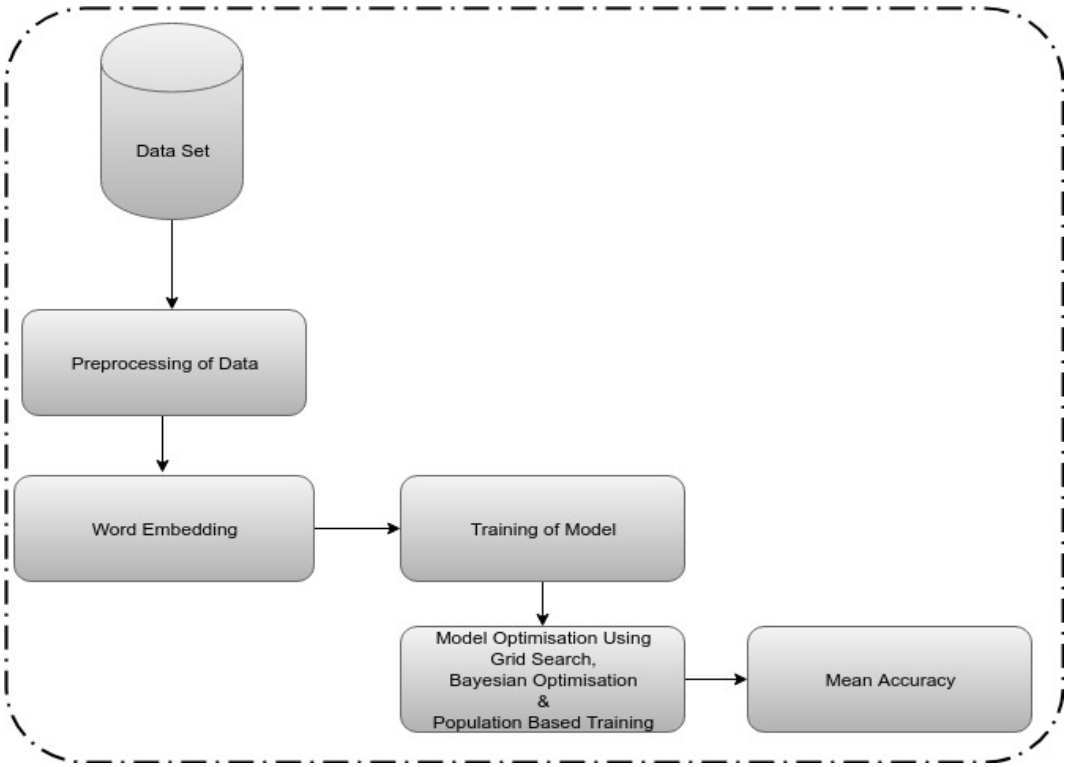


Figure 3.1: Implementation

The implementation followed the path set as shown in the figure above. This has been implemented using python 3.5. And as a result, I have created python classes with machine learning libraries. These classes contains the fit and transform method in order to comply with scikit-learn pipeline interface. I have also created some python function which also have machine learning libraries.

The following libraries are some of the python library used in this work:

- gensim: A natural language processing python library. This library was used in this work to provide the implementation of word2vec. It can also be used to make text mining cleaning easy.
- Pandas: A Python library for manipulation of Panel Data and analysis. It is used in this project for reading and manipulation of our dataset.

- Numpy: A scientific and numerical computing extension for Python programming language. It is used in manipulation of array.
- Scikit-learn pipeline: An scikit class that enables chaining of multiple.
- Scikit-learn grid search: An scikit class that enables systematically searching through multiple combinations of hyperparameter
- Nltk: A Python library used for computational linguistic, NLTK has some natural language processing tools which includes a tokenizer, a stemmer, some classifiers. We have used the tokenizer tool for the tokenizing of the dataset.
- re: A module that provides regular expression matching operations. Both Unicode and 8-bit strings can be searched using this. I have used this in this work to remove unwanted characters at the preprocessing stage of the data set.

This chapter covers various implementation and functionality aspects of our work.
Using python 3.5, I have implemented the following processing component to study the influence of hyperparameters on model accuracy. It is composed of the following components:

- Text preprocessor which filters and normalizes the input (preprocess)
- An adapter whose work is to allows easy iteration over the of Pandas DataFrames contents (Corpus_adapter)
- The Word2Vec Transformer which builds a word embedding model and transforms text pieces into their Word2Vec representation.
- A Word2Vec Classifier predicts the class of texts.

Using the Sklearn Pipeline, these aforementioned pieces are integrated together with Hyperparameter-optimization methods (Grid-search, Bayesian optimisation and Population based training) and the processing is carried out.

3.1 The Data Set

Only the review/text and review/score column will be used from the Amazon movie review dataset. The data set was converted to csv file. From the figure 2.2 above, we can see that the most of the review/scores falls between 4 and 5, and with an average score which is of 4.18. The distribution is more skewed to the left, we will be making a binary prediction. Using the pandas library, a new variable called Sentiment was created on the data set in which 0 represents negative, if a review/score is between 1 and 3, and 1 represents positive if a review/score is equal to 4 or 5 [67].

```
# Changing Amazon Data Set ti CSV format

import re
INPUT_FILE_NAME = "/home/ebuka/Desktop/masterarbeit/movies.txt"
OUTPUT_FILE_NAME = "/home/ebuka/Desktop/masterarbeit/movies.csv"

header = [
    "product/productId",
    "review/userId",
    "review/profileName",
    "review/helpfulness",
    "review/score",
    "review/time",
    "review/summary",
    "review/text"]

f = open(INPUT_FILE_NAME,encoding='utf-8',errors='ignore')
outfile = open(OUTPUT_FILE_NAME,"w")

# Write header
outfile.write(",".join(header) + "\n")
currentLine = []
for line in f:
   line = re.sub('[,]', '', line)
   line = line.strip()
   if line == "":
      outfile.write(",".join(currentLine))
      outfile.write("\n")
      currentLine = []
      continue
   parts = line.split(":",1)
   currentLine.append(parts[1])

if currentLine != []:
    outfile.write(",".join(currentLine))

f.close()
outfile.close()
```

The function below loads the dataset and extracts the two needed columns (The sentiment and the review/text column)

```
import pandas as pd

data= pd.read_csv('tempfile.csv', sep = ',',error_bad_lines=False)
data.loc[data.review_score>3,'Sentiment']=1
data.loc[data.review_score<=3,'Sentiment']=0
data =data[['review_text','Sentiment']]
data = data[data.Sentiment.isnull() == False]
data['Sentiment'] = data['Sentiment'].map(int)
data = data[data['review_text'].isnull() == False]
data.reset_index(inplace=True)
data.drop('index', axis=1, inplace=True)
data.to_pickle(pd_corpus_path + '/tempfile.pkl')
data.shape
```

```
b'Skipping line 141602: expected 2 fields, saw 3\nSkipping line 17776
b'Skipping line 582264: expected 2 fields, saw 3\nSkipping line 61878
```

```
(849529, 2)
```

Figure 3.2: Creation of Sentiment Column

With the creation of the Sentiment variable which is set to 0 (representing Negative) when the Score is between 1 and 3, and 1 (representing Positive) when the Score is equal to 4 or 5, we now have a new distribution:

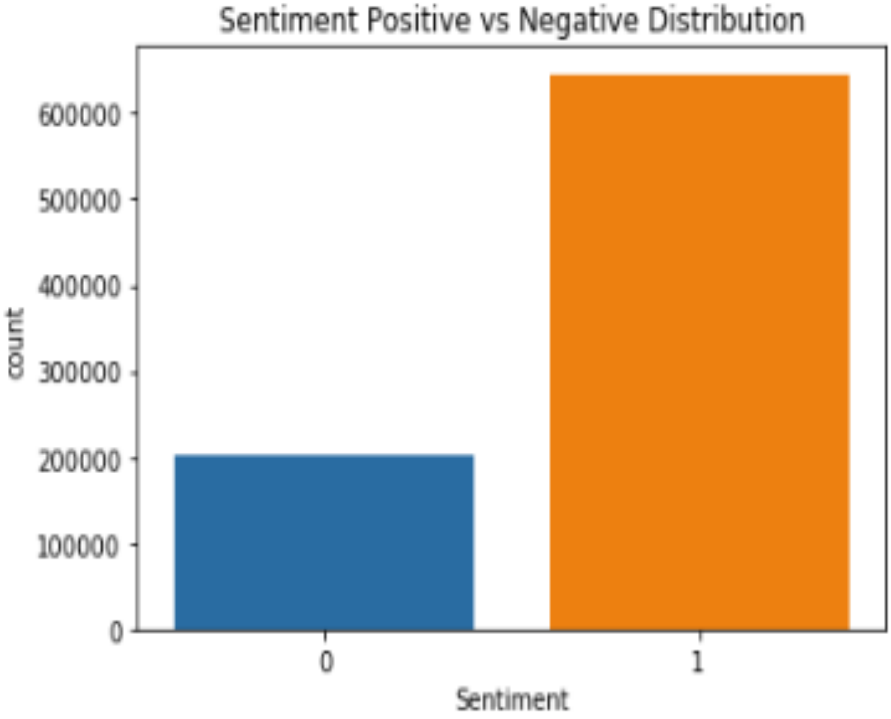


Figure 3.3: Distribution of Negative vs Positive Sentiments

Proportion of positive review	76%
Proportion of positive review	24%
Vocabulary size for positive reviews	1428101
Vocabulary size for negative review	716971

Table 3.1: Information of used Dataset

	Review_Text	Sentiment
0	Synopsis: On the daily trek from Juarez Mexic...	0
1	THE VIRGIN OF JUAREZ is based on true events ...	0
2	The scenes in this film can be very disquieti...	1
3	THE VIRGIN OF JUAREZ (2006) directed by ...	0
4	Informationally this SHOWTIME original is ess...	0
5	The murders in Juarez are real. This movie is...	0
6	Mexican men are macho rapists gangsters or in...	0
7	Over the past few years public television has...	1
8	I recvd this video (DVD version) as a Christm...	1
9	Wow! When I saw this show on PBS–that was it...	1
10	I have the Doo Wop 50 and 51 DVDs and was anx...	1
11	Having worked in television for 34 years I ca...	1
12	The people who have reviewed this DVD before ...	1
13	I have all of the doo wop DVD's and this one ...	1
14	The performance of Little Anthony and the Imp...	1
15	get, it, also, get, dop, wop, and, doo, wop, ...	1
16	Excellent excellent performers. Excellent vi...	1
17	This video is awesome and of particular inter...	1
18	As I stated in my reviews for Doo Wop 50 and ...	1
19	I own both the VHS and DVD versions of this p...	1

Table 3.2: Unpreprocessed Data set

3.2 Preprocessing Of Dataset

Texts can contain lots of noise and uninformative parts, like HTML tags. This increases the dimensionality of the problem thereby making the text classification process more difficult. Algorithms like removal of punctuation and symbols, tokenization, stemming, and stopword can be used to prepare and polish text before classification.
Using the NLtK tokenizer package in python, the data set has been tokenized and the use of re package, to make regular expression matching of unwanted characters and removed them.
The preprocessing tasks applied on the dataset includes:

- Tokenization of all the sentences
- removal of multiple dots with optional white spaces in between
- Removal of multiple white spaces
- Changing of all the characters to lower case
- Removal of all email addresses
- Replacing of newlines
- Removal of every digits apart from digits which represent year
- Conversion of 'ueber' to 'uber'
- Passing only meaningful characters by removing characters like #@!;:*=-%

```
class preprocess():
    df = data
    def tokenize(self,tweet):
        try:
            token = unicodedata.normalize("NFKD",
                tweet).encode("ascii", "ignore").decode("utf8") #
                converts 'ueber' to 'uber'
            token = re.sub('\
                |\?|\.|\!|\/|\;|\:|\<|\>|\+|\$|\*|\)|\(|\&|\=|\%|\-|\'|\"|\%{',
                ' ', token)# Lets pass only meaningful characters
            if '\n\n' in token:# remove header
                token = token[token.index('\n\n'):]
            token = re.sub(r'([^a-zA-Z0-9 \-\_%])', '', tweet)# Lets
                pass only meaningful characters
```

```
            token = re.sub(r'((\.\s*){2,})', '', token)# removes
                multiple dots with optional whitespaces in between
            token = re.sub(r'(\s{2,})', ' ', token) # Removes multiple
                whitespaces
            token = token.lower()# lower cases everything
            token = re.sub(r'\b(?!(\D\S*|[12][0-9]{3})\b)\S+\b', '',
                token) # removes all digits except digits that
                represent years
            token = re.sub(r'<.*?>', '', token)# remove html
            token =
                re.sub(r'[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+',
                '', token)# remove email addresses
            token = token.replace('\n', '')# replace newlines
            tokens = tokenizer.tokenize(token)
            return tokens
        except:
            return 'NC'


    def postprocess(self):

        self.data = self.df
        self.data['tokens'] =
            self.data['review_text'].progress_map(self.tokenize) ##
            progress_map is a variant of the map function plus a
            progress bar. Handy to monitor DataFrame creations.
        self.data = self.data[self.data.tokens != 'NC']
        self.data.reset_index(inplace=True)
        self.data.drop('index', inplace=True, axis=1)
        self.data.drop(['review_text'],inplace=True, axis=1)
        return self.data
```

The preprocessing code performs a basic cleaning operation, which includes removing unimportant or disturbing elements from the text so as to prepare it for the next phases of the task. Tabs and line breaks has been replaced using a blank. There are commas in between the preprocessed text because they are tokenised.

	Tokens	Sentiment
0	synopsis, on, the, daily, trek, from, juarez,...	0
1	the, virgin, of, juarez, is, based, on, true,...	0
2	the, scenes, in, this, film, can, be, very, d...	1
3	the, virgin, of, juarez, directed, by, kevin,...	0
4	informationally, this, showtime, original, is...	0
5	the, murders, in, juarez, are, real, this, mo...	0
6	mexican, men, are, macho, rapists, gangsters,...	0
7	over, the, past, few, years, public, televisi...	1
8	i, recvd, this, video, dvd, version, as, a, c...	1
9	wow, when, i, saw, this, show, on, pbs, that,...	1
10	i, have, the, doo, wop, and, dvds, and, was, ...	1
11	having, worked, in, television, for, years, i...	1
12	the, people, who, have, reviewed, this, dvd, ...	1
13	i, have, all, of, the, doo, wop, dvds, and, t...	1
14	the, performance, of, little, anthony, and, t...	1
15	get, it, also, get, dop, wop, and, doo, wop, ...	1
16	excellent, excellent, performers, excellent, ...	1
17	this, video, is, awesome, and, of, particular...	1
18	as, i, stated, in, my, reviews, for, doo, wop...	1
19	i, own, both, the, vhs, and, dvd, versions, o...	1

Table 3.3: Preprocessed Data set

3.3 The Corpus adapter

A class, Corpus adapter which wraps around the pandas dataframe is implemented in this work. It creates a sub dataset through picking of sample size documents randomly. Because this class implements the iter method, instances allow simple iteration over the sub dataset.

```
import pandas as pd

class Corpus_adapter():
    def __init__(self, corpus, sample_size=58, random_state=42):
        self.df = pd.read_pickle(corpus)

        assert sample_size >= 0, 'Sample size must be positive'
        if sample_size >= len(self.df):
            print('sample_size to large. will be set to max val:
                {}'.format(len(self.df)))
            sample_size = len(self.df)

        rnd = np.random.RandomState(random_state)
        self.sample = rnd.choice(self.df.index, sample_size,
            replace=False)

    def __iter__(self):
        for idx in self.sample:
            d = self.df.ix[idx]
            yield {
                    'idx': idx,
                    'text': d['tokens'],
                    'category': d['Sentiment']
                }
```

3.4 Word Embedding

The words has been represented by their vectors using an embedding layer. These embeddings are generated by using the word2vec approach, so that words that have similar meanings tends to be close to each other. The word2vec approach based on the assumption that words which are similar happen in similar context. By using the word2vec method, one is able to determine some semantic relationships, for instance, that a king and a man are male, or that a queen and a woman are all female.
A word2vec transformer class (Class W2VTransformer ()) is implemented in this work with the methods that are necessary to be processed by sklearn.pipeline.Pipeline, these methods are the fit and transform method. And also methods necessary to be processed by sklearn.model_selection.GridSearchCV which are the get_params and the set_params methods. It contains the Word2Vec model which is trained by the fit method. The transform method accepts an array-like object containing text sentences that are transformed to their vector representation.

```
# Code for making Word2vec Embeddings


class W2VTransformer( ):

    def __init__(self,**kwargs):

        self.kwargs=kwargs
        self.params = OrderedDict({
                'size': 24,
                'window': 100,
                'workers': 1,
                'min_count': 10,
                'sg': 1,
                'hs': 0,
                'iter': 5,
                'negative': 10,
                'alpha': .025,
                'min_alpha': 1e-4,
                'batch_words': 1000,
                'seed': 42,
            })
        self.params.update(kwargs)
```

```
        self.model = None

    def fit(self, X, y=None):
        #self.X = X
        """
        Fit the model according to the given training data.
        Calls gensim.models.Word2Vec
        """
        class IterList():
            def __init__(self, X):
               self.X = X
            def __iter__(self):
                return iter(self.X)
        listIter = IterList(X)

        self.sentences =listIter
        self.model = Word2Vec(**self.params )
        self.model.build_vocab(tqdm(self.sentences))
        self.model.train(tqdm(self.sentences),total_examples=self.model.corpus_count,
            epochs=self.model.epochs)
        self.model.init_sims()
        return self

    def transform(self, X):
        #self.dim = self.params['size']
        self.dim = self.model.vector_size
        return np.array([
        np.mean([self.model[w] for w in words if w in self.model]
            or [np.zeros(self.dim)], axis=0)
       for words in X])

    def get_params(self, deep=True):
        return self.params

    def set_params(self, **params):
        self.params.update(**params)
        return self
```

3.5 Training of the Model

To train the model, the k-Nearest neighbor classifier with cosine similarity distance metric (predict method) is implemented in this task in the class W2VClassifier(). The goal of the nearest neighbor classifier is to find an amount of predefined training samples which are closet in terms of distance to a new point , and then predicting of the label from it. It has a score method that determines the quality of the prediction with respect to the given dataset.

```
class W2VClassifier():
    def __init__(self, **kwargs):
        self.params = {
            'k': 5
        }
        self.params.update(kwargs)

    def fit(self, X, y):
        self.data = X
        self.targets = y

    def predict(self, X):
        dists = np.dot(X, self.data.transpose())
        best = np.fliplr(np.argsort(dists, axis=1))
        res = []
        for i, bestk in enumerate(best[:, 0:self.params['k']]):
            counter = defaultdict(int)
            for j, idx in enumerate(bestk):
                counter[self.targets[idx]] += dists[i][idx]
            counter = [(cat, val) for cat, val in counter.items()]
            res.append(sorted(counter, key=lambda x: -x[1])[0][0])
        return np.array(res)
    def score(self, X, y):
        pred = self.predict(X)
        return np.mean(pred == y)
    def get_params(self, deep=True):
        return self.params
    def set_params(self, **params):
        self.params.update(**params)
        return self
```

3.6 Hyperparameter Optimization

To optimize the hyperparameters of word2vec, three optimization methods: the grid search, bayesian optimization and the population based training is implemented in this work.

- Grid Search: The hyperpatemeter search (GridSearchCV) from scikit-learn is used here. It is provided in scikit-learn by the GridsearchCV class. It has a dictionary which contains the hyperparameters to be evaluated in the argument of param_grid. The accuracy is the score that is being optimized, the Cross validation of 5-fold is has been set to be used in evaluating of each individual model.

- Bayesian Optimization:

  The implementation of bayesian optimization is based on the work by Nils W. and Christin S. [66]. Bayesian optimization is a derivative-free paramater optimization method. Many different algorithm exists for the Bayesian optimization, but the Gaussian Process with Acquisition Function is used in this work.

- Population Based Training: The population based training is implemented based on Deep-minds paper [44]. The population-based training builds an idea of evolution, here you set parameter space (just like in random search) and create a random population of some length. In each iteration you go through every worker, and pass him full population from which you randomly pick extra worker. If the new worker is better than the current worker, then you set his parameters to a current worker. Also if a new worker was made, you need to add random noise to every hyper-parameter (set some standard deviation). The idea comes from genetic algorithm where population wants to find best parameters for training. Deep mind's idea was just like it with the added functionality of transfer logic (to copy weights of the model and pass it to another. Its start search is based on a random search where generating random population at the start then for each iteration the population wants to get closer to the best worker. To do so is done that for each worker (exploit) we pass him full population, he randomly samples another worker and if he is better than current worker, current copies parameters of sampled and adds them random noise (explore).

To perform the experiment:

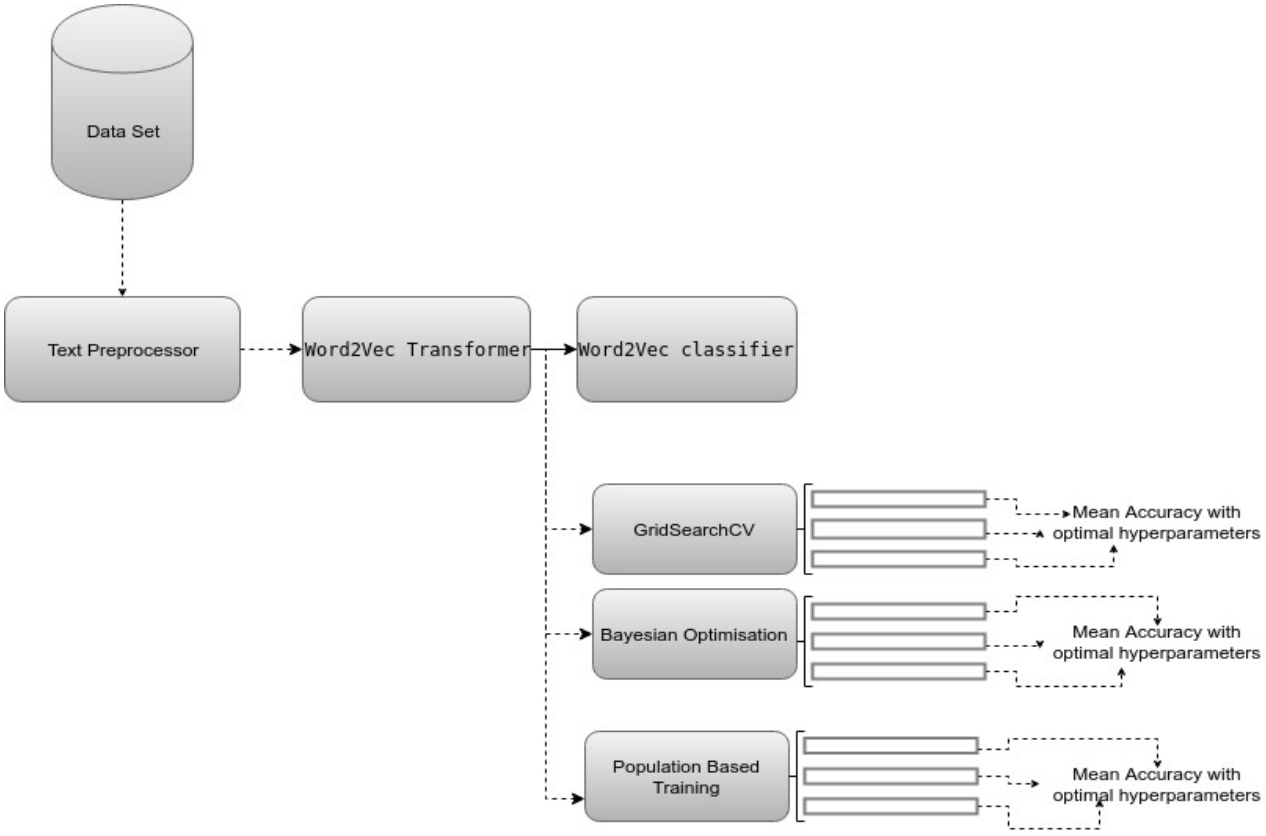


Figure 3.4: Experiment Step

- The dataset gets loaded.
- A transformer-classifier pipeline is built (Which contains the W2VTransformer class the W2VClassifier class)
- The hyperparameter space gets defined.
- The grid search, bayesian optimization and population based training is carried out

A Pipeline class integrates the transformation step with a classification step to become one object. The resulting object behaves just like a normal scikit-learn classifier. All the preprocessing is hidden within the Pipeline object.
The hyperpatemeter search (Grid search) works on the classifier and as well as a set of pre-defined hyperparameters which it is supposed to test for. GridSearchCV executes an exhaustive search on the entire parameter. The grid search uses Cross-validation and therefore Cross-validation is also performed along the way. The hyperparameter combination used in training the model is printed out with their mean accuracies and standard deviation.

3.7 Experiment

The grid search is used on the vectors of the hyperparameter with 5-fold cross-validation and the result is represented in terms of (macro-averaged) accuracy. The grid search performs an exhaustive search on a randomly chosen hyperparameter values ranging from high to low on 10,000 sample size. Also,the Bayesian optimization and Population based training was used to find an optimal hyperparameter configuration and their result were compared to the corresponding results gotten from using grid search. The training was done in two stages, the first stage uses grid search on the randomly chosen hyperparameter values while the second stage trains using the combination of the optimal hyperparameter found in this work and that of the related work. Training was further performed on the full by splitting it into two, 70 percent was used a the training data while 30 percent was used as the test data. And also using the optimal hyperparameters that were found on this work and also the optimal hyperparameter that was found from the related work. This is to enable comparing of the performance against state-of-the-art approach. The sample size of 10,000 was used in the first stage in consideration of the training time while sample size of 20,000 was used in the second stage.

θ	Description	Type	Values (Stage 1)	Values (Stage 2)
$\theta arch$	Skip-gram / cbow	boolean	1,0	1
θns	Negative sampling	integer	5,15,50	20,50
θhs	Hierarchical softmax	boolean	1,0	0,1
θwin	Window Size	integer	5,20,50	5,50
θd	Dimensionality Size	integer	20,100,200,300	64,100,300
$\theta \alpha$	learning rate	real	.0001, .001, .01	.10,0.01
$\theta epochs$	Number of iterations	integer	1,20,70	1,30,20
θts	size of training documents	integer	10,000	20,000

Table 3.4: Overview of the hyperparameter space θ in the stages S1, S2

The source code of the experiments as well as additional material is available online. Table 4 shows an overview of the parameter settings. The training was done using the amazon movie review dataset. The hyperparameter θ values were randomly chosen ranging from low to high values. The word2vec hyperparameter considered in the training are $\theta arch$ which represents the training algorithm, keeping the value at 1 means that skip-gram will be used while 0 means otherwise that cbow will be used. θns if > 0, represents negative sampling will be used when value is > 0, the negative specifies the number of

noise words which should be drawn. θhs signifies hierarchical softmax, keeping the value at 1 means that hierarchical softmax will be used for model training while 0 means that negative sampling will be used. θwin window represents the maximum distance between the word that is being predicted within a sentence and the current. θd this represents the Dimensionality of the word vectors, $\theta \alpha$ is the initial learning rate while $\theta epochs$ represents the number of iterations on the corpus.

4 Evaluation

We present in this chapter the results of the performed experiment. The presented result is based on a training dataset with document size of 10,000 samples and shows the general performance, the interrelation and the influence of hyperparameters on the model accuracy.

4.1 General Performance

Table 5 below shows the result of the classification that was carried out using 10,000 training examples of the dataset for grid search, bayesian optimisation and population based training. This is arranged according to the performance of the model (the best and the worst performing model) and shows the accuracy a of the training.

Method	Rank	$\theta arch$	θns	θhs	θwin	θd	$\theta \alpha$	$\theta epochs$	a
GS	1	1	50	0	50	100	0.01	20	.7816
GS	2	1	15	0	50	100	0.01	20	.7790
GS	3	1	5	1	50	100	0.01	20	.7769
GS	4	1	5	0	50	100	0.01	20	.7767
GS	5	1	15	1	20	100	0.01	20	.7766
GS	1292	0	5	1	20	200	0.01	1	.6979
GS	1293	0	5	1	20	20	0.01	1	.6973
GS	1294	1	5	0	50	300	0.0001	1	.6973
GS	1295	1	50	0	5	300	0.0001	20	.6972
GS	1296	0	5	1	20	300	0.001	1	.6952
BO	1	1	3	1	24	86	0.0122	36	.7789
BO	2	0	23	0	119	108	0.0367	38	.7755
BO	3	1	15	0	45	94	0.1027	40	.7738
BO	4	1	24	1	85	132	0.0305	35	.7735
BO	5	1	5	0	19	84	0.0624	7	.7735
PBT	1	0	14	0	143	139	0.0972	9	.7730
PBT	2	0	16	0	143	138	0.1201	9	.7724
PBT	3	0	13	0	140	138	0.1178	7	.7718
PBT	4	0	14	0	146	132	0.1218	7	.7715
PBT	5	0	12	0	141	139	0.1111	9	.7713

Table 4.1: Experiment Result

For the general performance of the training, the overall accuracy of the model varies depending on different settings of the hypeparameters. The table 5 above shows the the best result gotten using the amazon movie review dataset on a training example of 10,000. The best performing model is assigned the rank of 1 as the result is sorted by rank. From table 2 also, it can be seen that the influence of the hyperparameters are not so noticeable, for instance the difference in the accuracy of one model to the model after it is not so much. It can be observed that the best performing models has its architecture set to 1 while the architecture of 0 appears more at the worst performing models. Also among the four values of the dimension sizes tested (20,100,200 and 300), the best performing models has its dimension size at 100 while 300 is found more among the worst performing model. It can also be observed that the overall performance was not very bad since the best performing model has an accuracy of 0.7816 while the worst performing model has an accuracy of 0.6952. The difference between the two are not so much.

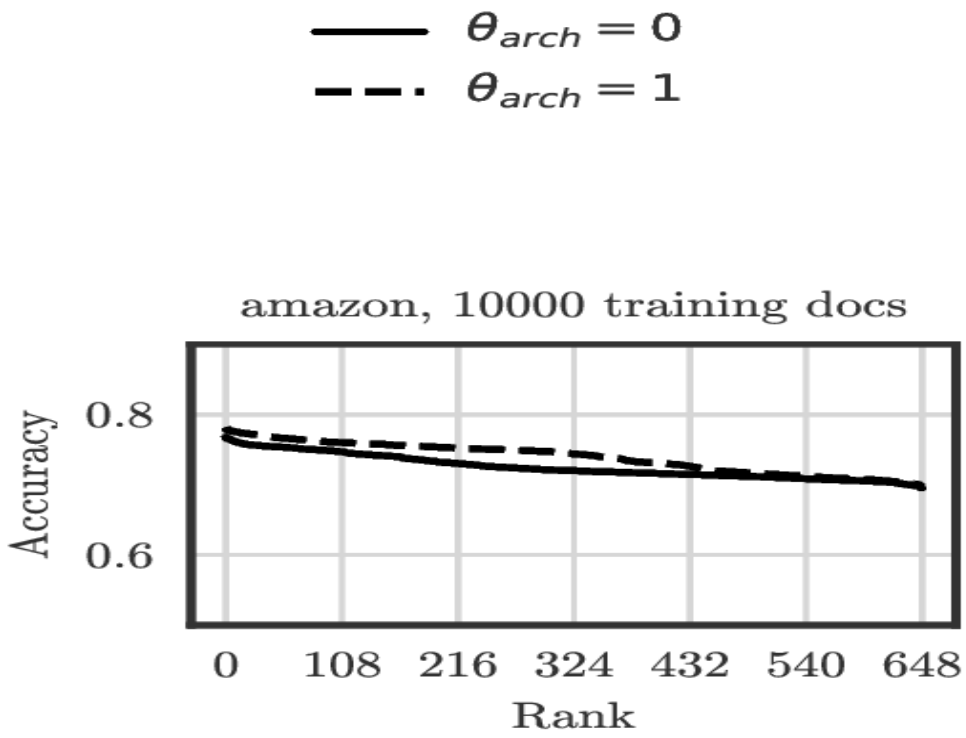


Figure 4.1: Varrying Architecture $\theta arch$

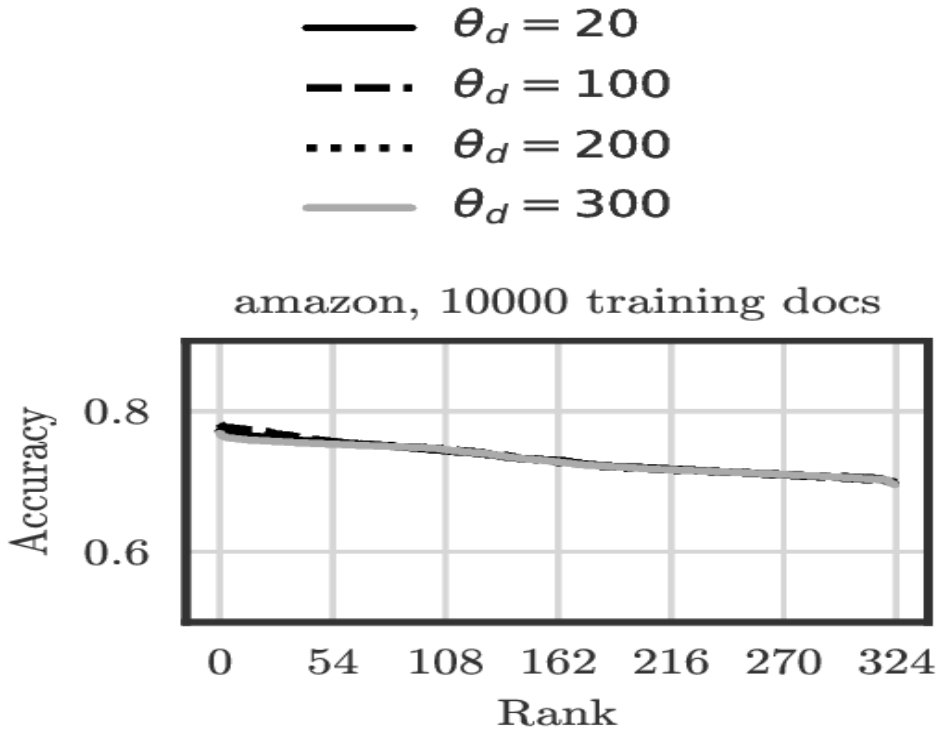


Figure 4.2: Varrying Dimensionality size θd

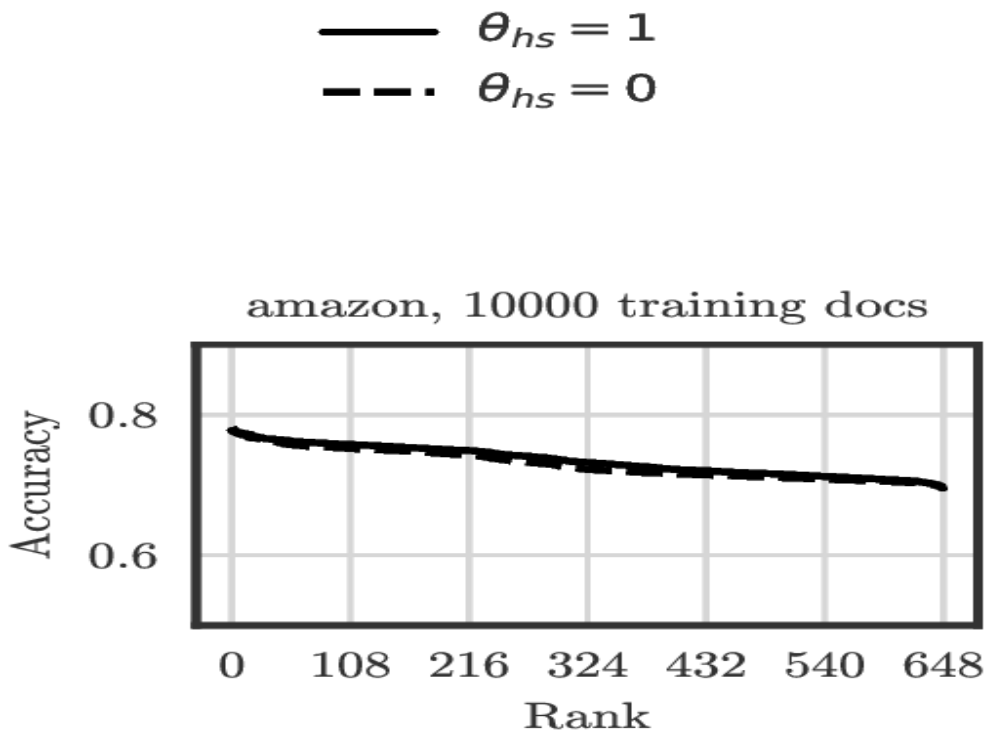


Figure 4.3: Varrying Hierarchical Softmax θhs

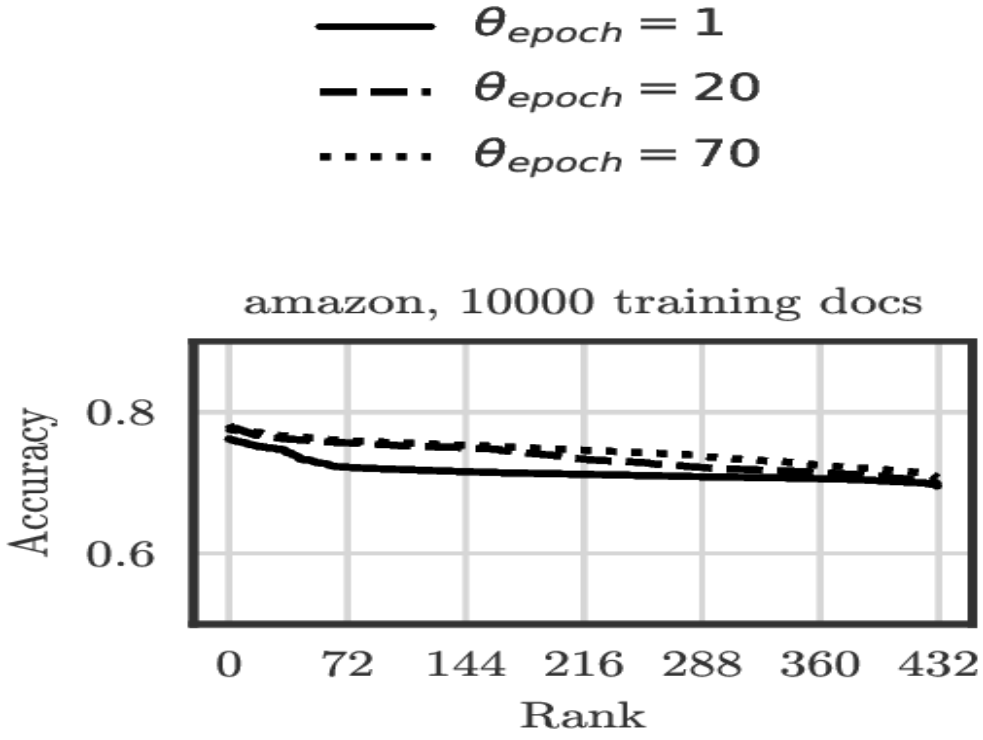


Figure 4.4: Varrying Iteration $\theta epochs$

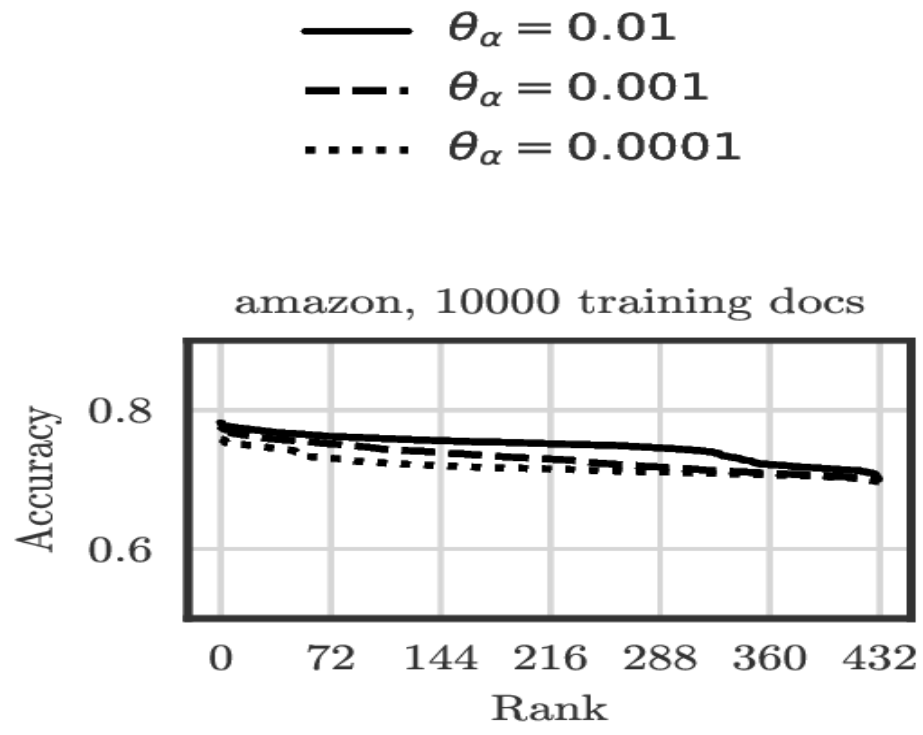


Figure 4.5: Varrying Learning Rate $\theta\alpha$

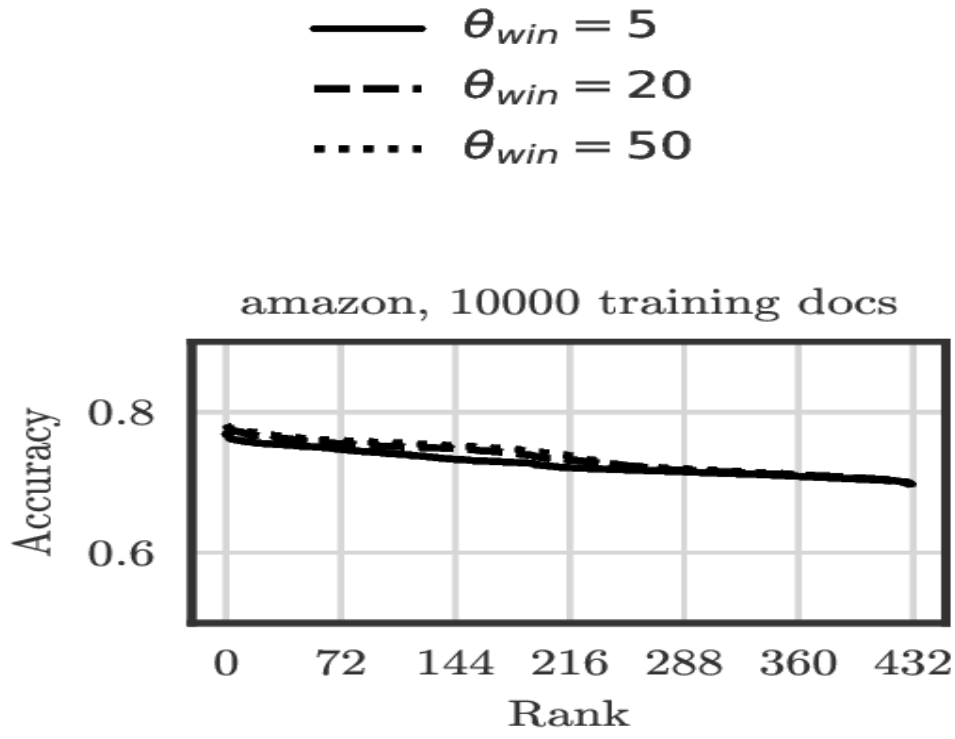


Figure 4.6: Varrying Window θwin

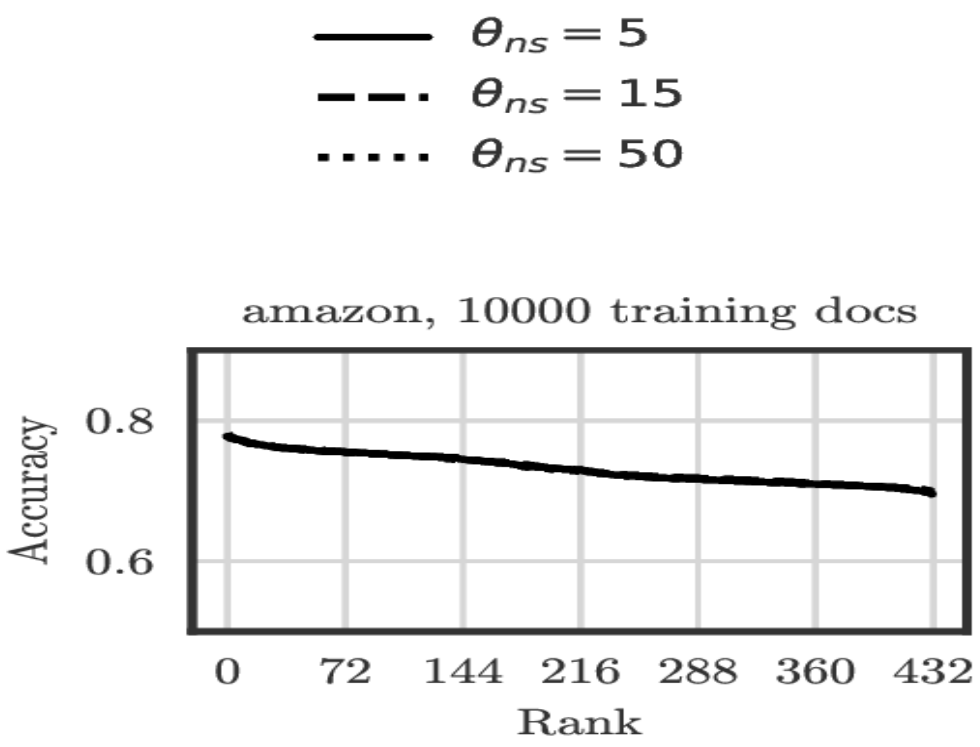


Figure 4.7: Varrying Negative Samples θns

The above figures (fig 5.1 - 5.7) shows the classifier performance on different hyperparameter settings. It represents the ranking of the classifier for the training done using 5 fold cross validation.

Method	Rank	$\theta arch$	θns	θhs	θwin	θd	$\theta \alpha$	$\theta epoch$	a
GS	1	1	50	0	50	64	0.01	20	.7861
GS	2	1	50	0	50	64	0.01	30	.7857
GS	3	1	20	0	50	100	0.01	30	.7856
GS	4	1	20	0	50	64	0.01	20	.7847
GS	5	1	20	1	50	64	0.01	20	.7839

Table 4.2: Result of the Second Stage of the Experiment

SN	$\theta arch$	θns	θhs	θwin	θd	$\theta \alpha$	$\theta epoch$	a
1	1	50	0	50	100	0.01	20	.8409
2	1	50	1	5	64	0.10	30	.8295

Table 4.3: Result of the training using the full dataset of 849529 sample size, the dataset was split into two and 70% has been used for the training while 30% was used as the test data. First row represents the optimal hyperparameter settings from this work while the second row is that of the the related work [66]

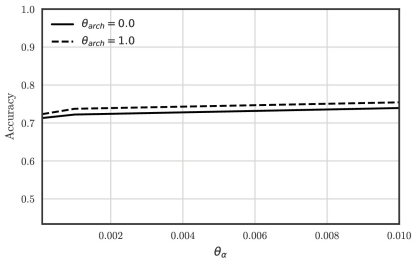
Table 4.2 above further displays the result of the second stage of the training on a sample size of 20,000 using the optimal hyperparameter settings found on this work and from the related work.

4.2 Influence of Hyperparameters

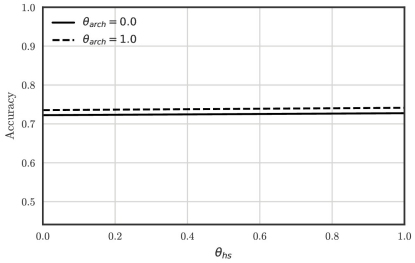
As shown in the figure fig 5.1 - 5.7, The plotted is the accuracy of the model for the the different hyperparameter values for all the tested hyperparameters of word2vec. To create the plots, the results of all the models were collected, as one hyperparameter is assigned a specific value. The order of all the models are arranged according to their accuracy. Between the two model architecture, the skip-gram architecture ($\theta arch = 1$) models performed better than the cbow ($\theta arch = 0$) models in almost all cases as represented in Figure 5.1. Three epoch values of 1, 20 and 70 were used, the the performance of the models using the higher epoch of 70 were better than the models using lower epoch of 20 then followed by 1. For the parameter hs, models which are not using hierarchical softmax also outperformed the models that are making use of hierarchical softmax. Very high embedding sizes doesn't have good impact on the accuracy but keeping the value of the embedding size at moderate yield classifiers with better accuracy . Little effect was found on using window size while the effect of , the number of negative samples were not noticeable on the accuracy of the model, this is also the case with varying of the dimension size as shown in fig 5.2.
Lastly, the the hyperparameter that gives the strongest influence on the model accuracy is the learning rate and the epoch. Varying these two hyperparameter leads to a great influence and interrelation on accuracy as can be seen in fig 5.4 and 5.5

4.3 Interrelation of Hyperparameters

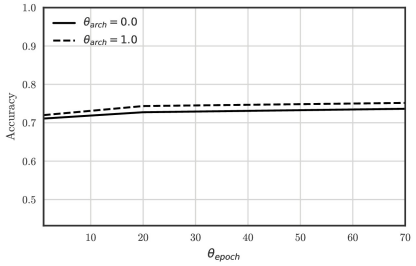
The figure below represents the interrelation of the hyperparameters. Each plot displays an overview of some selected parameters that were tested in this work, the full plot for all the hyperparameters can be find on the link below. The legend inside the graph specify the subplot of the variable of interest while the x-axes depict the plot of other variables. All y-axes in the plots depict the mean success rate of the classification task. Each subplot describes the influence of the corresponding variables on the mean results of the parameter search.



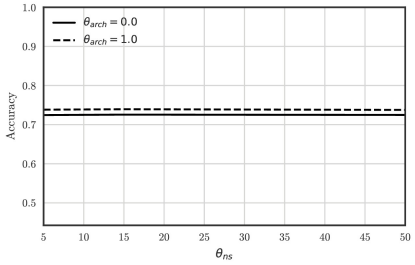
(a) Comparing $\theta arch$ with $\theta \alpha$



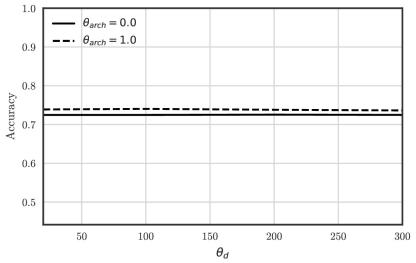
(b) Comparing $\theta arch$ with θhs



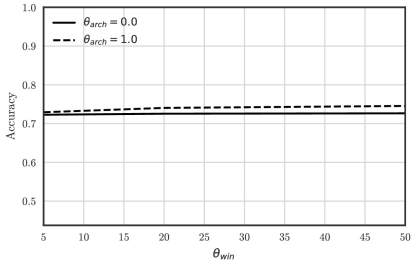
(c) Comparing $\theta arch$ with $\theta epoch$



(d) Comparing $\theta arch$ with θns

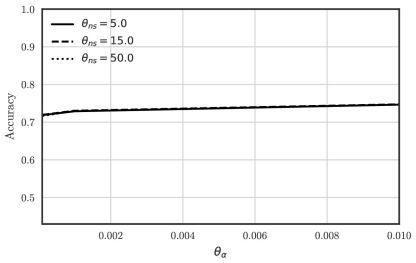


(e) Comparing $\theta arch$ with θd

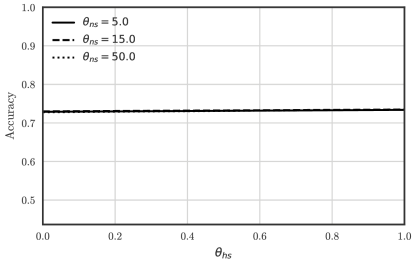


(f) Comparing $\theta arch$ with θwin

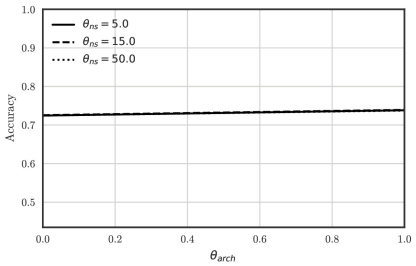
Figure 4.8: Plot of Interrelation of $\theta arch$ and Other hyperparameter



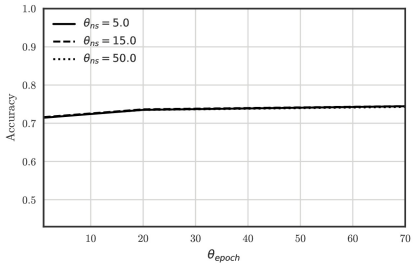
(a) Comparing θns with $\theta \alpha$



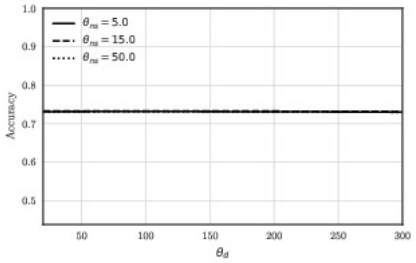
(b) Comparing θns with θhs



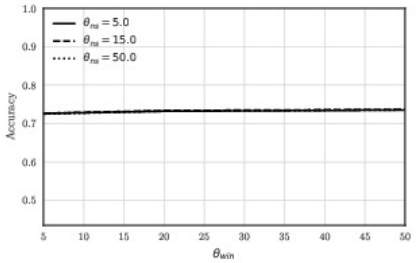
(c) Comparing θns with $\theta arch$



(d) Comparing θns with $\theta epoch$

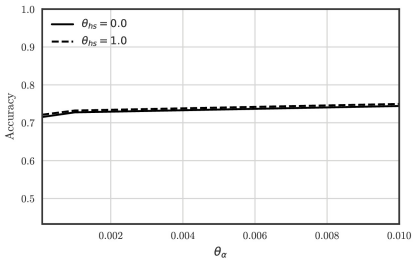


(e) Comparing θns with θd

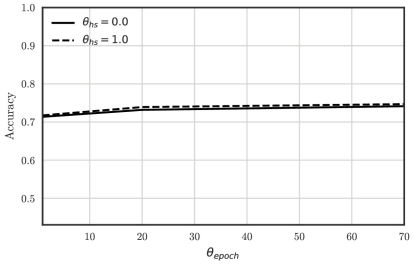


(f) Comparing θns with θwin

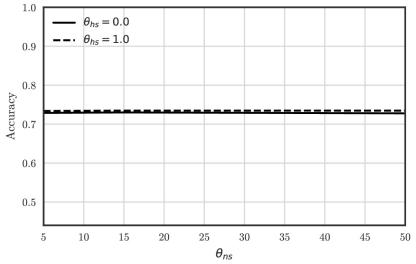
Figure 4.9: Plot of Interrelation of θns and Other hyperparameter



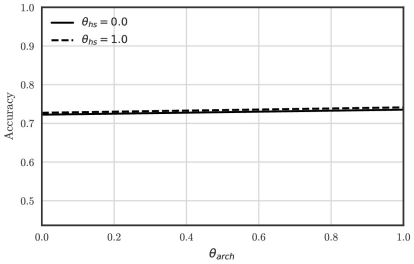
(a) Comparing θhs with $\theta \alpha$



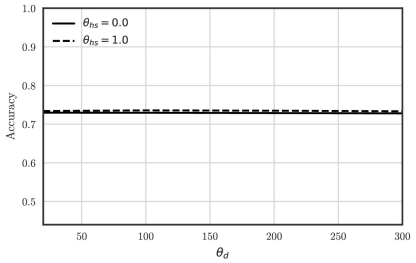
(b) Comparing θhs with $\theta epoch$



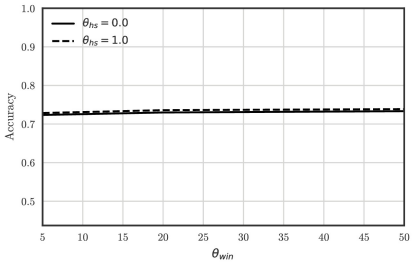
(c) Comparing θhs with θns



(d) Comparing θhs with $\theta arch$

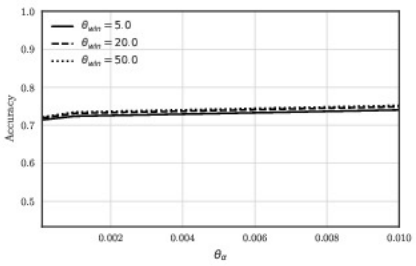


(e) Comparing θhs with $\theta size$

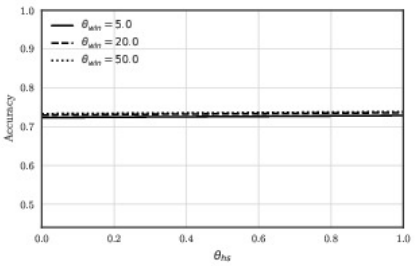


(f) Comparing θhs with θwin

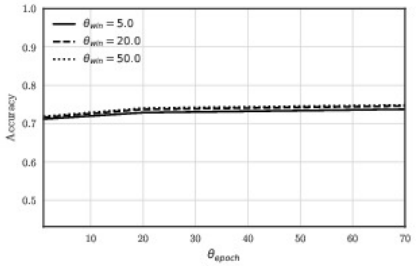
Figure 4.10: Plot of Interrelation of θhs and Other hyperparameter



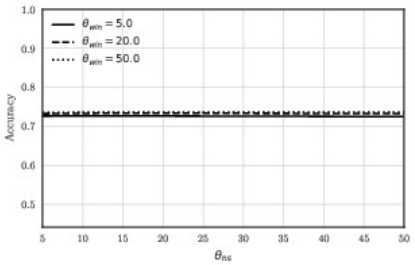
(a) Comparing θwin with $\theta \alpha$



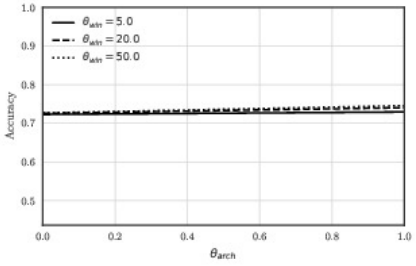
(b) Comparing θwin with θhs



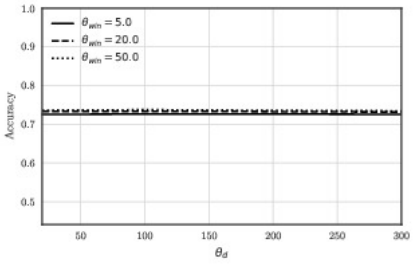
(c) Comparing θwin with $\theta epoch$



(d) Comparing θwin with θns

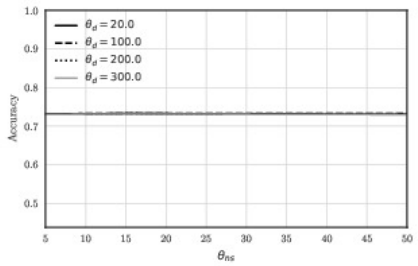


(e) Comparing θwin with $\theta arch$

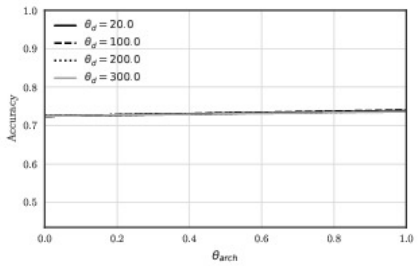


(f) Comparing θwin with θd

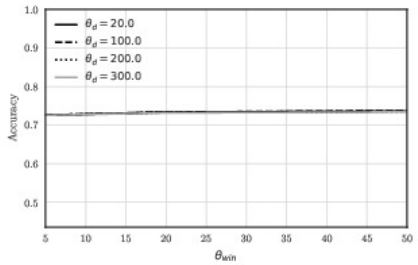
Figure 4.11: Plot of Interrelation of θwin and Other hyperparameter



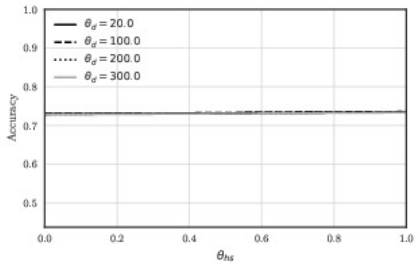
(a) Comparing θd with θns



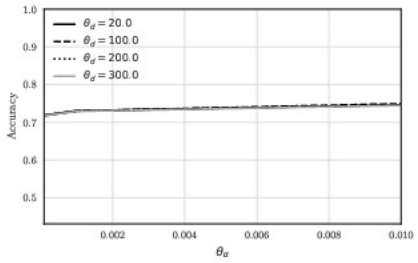
(b) Comparing θd with $\theta arch$



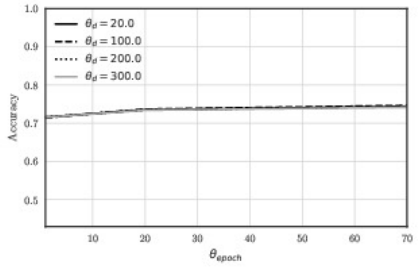
(c) Comparing θd with θwin



(d) Comparing θd with θhs



(e) Comparing θd with $\theta \alpha$



(f) Comparing θd with $\theta epoch$

Figure 4.12: Plot of Interrelation of θd and Other hyperparameter

From figure 5.8 to 5.12, interrelation was not found between most of the hyperparameters. But for all the hyperparameters, their performance remains steady at an epoch of 20 with no further noticeable increase in performance after an iteration of 20.

4.4 Discussion

This master thesis studies the hyperparameters of word2vec with the aim of finding their influence on the accuracy of a classification model, the interrelation between the hyperparameters and which of the hyperparameter optimization methods gives optimal hyperparameter. To do this, I have performed a sentiment analysis using the amazon movie review dataset, k nearest neighbour as the classifier, while grid search was used to study the influence and the interrelation of the hyperparameters, bayesian optimization and population based training was also used in order to find optimal hyperparameter settings.
Grid search performed better than bayesian optimisation and population based training with an accuracy of 0.7816 although the difference in accuracies of the best performing model given by the three hyperparameter methods are not much, with the best performing model of bayesian optimisation having 0.7789 and population based training having 0.7730. Also the variation of accuracy across models is not so much as the best performing model accuracy is 0.7816 while the worst is 0.6952. The small variation of accuracy across models might be as a result of the proportion of the negative and the positive comments in the dataset which amounts to 76% for the positive comments and 24% for the negative comments.

Result from the experiments showed that generally, hyperparameter settings is very important on the accuracy as they can have a large impact on the accuracy. Though not all the hyperparameters have big influence in the accuracy. The hyperparameters can be grouped into two: The group with good influence on the accuracy, the second group without influence on the accuracy.

The type of model architecture, epoch and learning rate falls in the first group. The effect of architecture can be visibly seen as the skip-gram outperformed the cbow in in all cases. This confirms skip-gram as the best choice for a sentiment analysis task. As shown in our literature that the choice of the training algorithm and the hyperparameter selection is a task specific decision [80].
The second group of hyperparameters are the those that doesn't have a noticeable effect on the accuracy. For hyperparameters like the window size, negative samples, dimensionality size and hierachical softmax and negative sampling. No much influence is found in using them (see Figure 5.2, 5.3, 5.6 and 5.7). The effect of hierarchical softmax and dimensionality is not strong unlike in the literature [66] where classifier with hierachical softmax is found to perform better than those without. From the result of the two stages of experiment, the best performing model has $\theta d = 100$ in the first stage of the experiment

and $\theta d = 64$ in the second stage. While the accuracy at $\theta d = 20$, 200 and 300 does not fall among the best. From experiment, higher dimensionality size did not ensure good accuracy as seen in the case of literature [4] where the performed experiment is a syntactic and semantic language tasks.
No interrelation was found between the hyperparameters but for epoch, looking at figure 5.8 - 5.12, it can be seen that for all the hyperparameters when compared with epoch, there is a steady increase in performance until it gets to $\theta epoch = 20$ and no further effect on the accuracy after $\theta epoch = 20$. The value of $\theta epoch$ must be considered always as it seems that using values that exceeds 20 is pointless.

4.5 Conclusion

In this paper we studied the hyperparameters of word2vec word embeddings by performing a sentiment analysis task.
From the result of the experiments, it can be seen that that the influence of window size, negative samples, hierachical softmax and dimensionality size on the accuracy of a model are negligible.
High dimensionality size did not give good accuracy as the best performing value for dimensionality size falls between 64 - 100.
From the result of the experiment, the skip-gram architecture gives good accuracy than the cbow architecture. Interrelations was not found between the hyperparameters but for all the hyperparameters when tuned with epoch maintained a steady performance improve until $\theta epoch = 20$.

These understandings can be useful during manual selection of hyperparameters, also to reduce hyperparameter search space the optimisation because it will aid in choosing of the hypeparameters initial values for grid-search, bayesian optimisation and population based training.

5 Future Work

Since we have studied the word2vec hypeprparameters in this work using a sentiment analysis task.
For future work, the first suggestion is to try the experiment using other different machine learning classifier since knn has been used in this work.
The second suggestion for future work is to carry the experiment on another dataset. The amazon movie review dataset has been used in this work. The accuracy of model can be influenced by the used data set therefore trying the experiment on other dataset will be good for a future work. Also using more sample size for the experiment is recommended.

6 Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Passau, den <date>

<First Name, Last Name>

7 References

1. Leskovec, J. (2013) Web data: Amazon movie reviews.
Available at: https://snap.stanford.edu/data/web-Movies.html. May 2018
2. Ahmed B. (2016) Web data: Sentiment Analysis on twitter using word2vec and keras. Available at: https://ahmedbesbes.com/sentiment-analysis-on-twitter-using-word2vec-and-keras.html. May 2018
3 Chopra, Abhimanyu, Abhinav Prashar, and Sain Chandresh. Natural Language Processing: International Journal of Technology Enhancements and Emerging Engineering Research 1 (4): 131–34 (2013)
4 Tomas Mikolov, Kai Chen,Greg Corrado and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space (2013)
5 Jeffrey Pennington, Richard Socher, Christopher D. Manning. Glove: Global Vectors for Word Representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 1532–43. https://doi.org/10.3115/v1/D14-1162 (2014)
6 Jey Han Lau et al. An empirical evaluation of doc2vec with practical insights into document embedding generation. arXiv:1607.05368 (2016)
7 Rezaeinia, Seyed Mahdi, Ali Ghodsi, and Rouhollah Rahmani. Improving the Accuracy of Pre-trained Word Embeddings for Sentiment Analysis. arXiv preprint arXiv:1711.08609 (2017)
8 Tang, Duyu, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. Learning Sentiment-Specific Word Embedding. Acl, 1555–65. https://doi.org/10.3115/1220575.1220648 (2014)
9 Eggensperger, Hutter F., Hoos H. and Leyton-Brown K. Surrogate benchmarks for hyperparameter optimization. In CEUR Workshop Proceedings, Vol. 1201, pp. 24–31 (2014)
10 Bardenet R., Brendel M., Kegl B. and Sebag M. Collaborative hyperparameter tuning. Proceedings of the 30th International Conference on Machine Learning (ICML-13), 28(2), 199–207. Retrieved from http://hal.archives-ouvertes.fr/in2p3-00907381/11 Bengio Y. . Gradient-based optimization of hyperparameters. Neural Computation, https://doi.org/10.1162/089976600300015187 (2000)
12 Keerthi S., Sindhwani V., and Chapelle O. An Efficient Method for Gradient-Based Adaptation of Hyperparameters in SVM Models. Advances in Neural Information Processing Systems(2006)

13 Loshchilov I. and Hutter F. Cma-Es for Hyperparameter Optimization Deep Neural Networks (2016)
14 Hazan E., Klivans A. and Yuan Y. Hyperparameter optimization: a spectral approach. arXiv preprint arXiv:1706.00764 (2017)
15 Blum M. and Riedmiller M. Optimization of Gaussian Process Hyperparameters using Rprop. European Symposium on Artificial Neural Networks Computational Intelligence and Machine Learning (2013)
16 Pedregosa F . Hyperparameter optimization with approximate gradient. arXiv preprint arXiv:1602.02355 (2016)
17 Maclaurin D., Duvenaud D. and Adams R. Gradient-based hyperparameter optimization through reversible learning. In International Conference on Machine Learning pp. 2113-2122 (2015)
18 Franceschi L., Donini M., Frasconi, P. and Pontil M. Forward and reverse gradient-based hyperparameter optimization. arXiv preprint arXiv:1703.01785 (2017)
19 Maclaurin D., Duvenaud D. and Adams R. Gradient-based hyperparameter optimization through reversible learning. In International Conference on Machine Learning (pp. 2113-2122 (2015)
20 Socher R. Recursive Deep Learning for Natural Language Processing and Computer Vision. PhD Thesis (2014)
21 Le Q. and Mikolov T. Distributed representations of sentences and documents. In International Conference on Machine Learning pp. 1188-1196 (2014)
22 Tellez E., Moctezuma D., Miranda-Jimenez S. and Graff, M. An automated text categorization framework based on hyperparameter optimization. Knowledge-Based Systems (2018)
23 Mikolov T., Chen K., Corrado G. and Dean J. (2013). Distributed representations of words and hrases and their compositionality (2013)
24 Goldberg Y. and Levy O. "word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method." arXiv preprint arXiv:1402.3722 (2014)
25 Mnih A, and Hinton G. E. A Scalable Hierarchical Distributed Language Model. Advances in Neural Information Processing Systems (2008)
26 Xu Y., Mou L., Li G. and Chen, Y. Classifying Relations via Long Short Term Memory Networks along Shortest Dependency Paths (2015)
27 Hendrickx I., Kim S. N., Kozareva Z., Nakov P., O Seaghdha D., Pado S., and Szpakowicz S. Multi-Way Classification of Semantic Relations Between Pairs of Nominals (2010)
28 Socher R., Pennington J., Huang E. H., Ng A. Y. and Manning, C. D. Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions.

EMNLP 2011 - Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference, 151–161. https://doi.org/10.1.1.224.9432 (2011)
29 Zeng D., Liu K., Lai S., Zhou G. and Zhao J. Relation Classification via Convolutional Deep Neural Network. (2014)
30 Santos C, Xiang B. and Zhou B. Classifying relations by ranking with convolutional neural networks. arXiv preprint arXiv:1504.06580 (2015)
31 Gormley M. R. and Dredze M. Factor-based Compositional Embedding Models. NIPS Workshop on Learning Semantics (2014)
32 Vu N., Adel H., Gupta P. and Schütze H. Combining recurrent and convolutional neural networks for relation classification. arXiv preprint arXiv:1605.07333. (2016)
33 Wen Y., Zhang W., Luo R.and Wang, J. Learning text representation using recurrent convolutional neural network with highway layers. arXiv preprint arXiv:1606.06905. (2016)
34 Yin W., Schütze H., Xiang B. and Zhou, B. Abcnn: Attention-based convolutional neural network for modeling sentence pairs. arXiv preprint arXiv:1512.05193 (2015)
35 Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. Language modeling with gated convolutional networks. arXiv preprint arXiv:1612.08083 (2016)
36 Arkhipenko A., kirillskorniakov, Gomzin G. A. and turdakov T. D. Comparison of neural network arChiteCtures for sentiment analysis of russian tweets (2016)
37 Yin W., Kann K., Yu M. and Schütze, H. Comparative study of cnn and rnn for natural language processing. arXiv preprint arXiv:1702.01923 (2017)
38 Chung J., Gulcehre C., Cho K. and Bengio Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555 (2014)
39 Zhang R. and Rezaee Z. "Do Credible Firms Perform Better in Emerging Markets? Evidence from China." Journal of Business Ethics 90 (2): 221–37. https://doi.org/10.1007/s10551-009-0038-8 (2009)
40 Bergstra J., Yamins D., and Cox D. "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures." ICML, 115–23. http://jmlr.org/proceedings/papers/v28/bergstra13.html (2013)
41 LeCun Y., Bottou L., Bengio Y., and Haffner P. Gradient-Based Learning Applied to Document Recognition. https://doi.org/10.1109/5.726791 (1998)
42 Bergstra J., James and Bengio U. Y. "Random Search for Hyper-Parameter Optimization." Journal of Machine Learning Research (2012)
43 Snoek J., Larochelle H. and Adams R. "Practical Bayesian Optimization

of Machine Learning Algorithms." Adv. Neural Inf. Process. Syst. 25, 1–9. https://doi.org/2012arXiv1206.2944S (2012)
44 Jaderberg M., Dalibard V., Osindero S., Czarnecki W., Donahue J., Razavi A. and Vinyals, O. "Population Based Training of Neural Networks." Arxiv. http://arxiv.org/abs/1711.09846 (2017)
45 Lutins E. "Grid Searching in Machine Learning: Quick Explanation and Python Implementation." Medium. https://medium.com/@elutins/grid-searching-in-machine-learning-quick-explanation-and-python-implementation-550552200596 (accessed May 24, 2018)
46 Larochelle H., Erhan D., Courville A., Bergstra J., and Bengio Y. "An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation." In Proceedings of the 24th International Conference on Machine Learning - ICML '07, 473–80. https://doi.org/10.1145/1273496.1273556 (2007)
47 Jamieson K. Hyperband Bandit Based Configuration Evaluation for Hyperparameter Optimization (2017)
48 Thornton C., Hutter F., Hoos H. H., and Leyton-Brown, K. Auto-WEKA: Automated selection and hyper-parameter optimization of classification algorithms. CoRR, abs/1208.3719 (2012)
49 Klein A., Falkner S., Bartels S., Hennig P. and Hutter, F. Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. ArXiv Cs.LG. https://doi.org/10.1214/17-EJS1335SI (2016)
50 Wistuba M., Schilling N., and Schmidt-Thieme, L. Hyperparameter search space pruning – A new component for sequential model-based hyperparameter optimization. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 9285, pp. 104–119). https://doi.org/10.1007/978-3-319-23525-7_7 (2015)
51 Brochu E., Cora V. M. and De Freitas, N. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning (2010)
52 Wistuba M., Schilling N. and Schmidt-Thieme, L. Hyperparameter optimization machines. In Proceedings - 3rd IEEE International Conference on Data Science and Advanced Analytics, pp. 41–50. https://doi.org/10.1109/DSAA (2016a) 53 Feurer M., Springenberg J. T. and Hutter, F. Initializing Bayesian Hyperparameter Optimization via Meta-Learning. AAAI, 1128–1135 (2015)
54 Brazdil P., Carrier C. G., Soares, C. and Vilalta R. Metalearning: Applications to Data Mining. Metalearning: Applications to Data Mining (2008)
55 Jaderberg M., Mnih V., Czarnecki W. M., Schaul T., Leibo J. Z., Silver D. and Kavukcuoglu K. Reinforcement learning with unsupervised auxiliary tasks. arXiv preprint arXiv:1611.05397 (2016)

56 Beattie C., Leibo J. Z., Teplyashin D., Ward T., Wainwright M., Lefrancq A. and Sadik A. Deepmind lab. arXiv preprint arXiv:1612.03801 (2016)
57 Vezhnevets A. S., Osindero S., Schaul T., Heess N. and Jaderberg M Feudal networks for hierarchical reinforcement learning. arXiv preprint arXiv:1703.01161 (2017)
58 Bellemare M. G., Naddaf Y., Veness J., and Bowling M. The arcade learning environment: An evaluation platform for general agents. J. Artif. Intell. Res.(JAIR), 47:253–279 (2013)
59 Vinyals O., Ewalds T., Bartunov S., Georgiev P., Vezhnevets A and Schrittwieser J. Starcraft II: A new challenge for reinforcement learning. arXiv preprint arXiv:1708.04782 (2017)
60 P. R., Nalepa J., Kawulok M., Ramos, L. S. and Pastor, J. Particle swarm optimization for hyper-parameter selection in deep neural networks. Proceedings of the 2017 on Genetic and Evolutionary Computation Conference (2017)
61 Guo X. C., Yang J. H., Wu C. G., Wang C. Y. and Liang, Y. C. A novel LS-SVMs hyper-parameter selection based on particle swarm optimization. In Neurocomputing (Vol. 71, pp. 3211–3215). https://doi.org/10.1016/j.neucom.2008.04.027 (2008) 62 Lorenzo P. R., Nalepa J., Kawulok M., Ramos L. S. and Pastor J. R. Particle swarm optimization for hyper-parameter selection in deep neural networks. Proceedings of the 2017 on Genetic and Evolutionary Computation Conference(2017)
63 Liao R., Zheng H., Grzybowski S. and Yang L. Particle swarm optimization-least squares support vector regression based forecasting model on dissolved gases in oil-filled power transformers. Electric Power Systems Research, 81(12), 2074–2080. https://doi.org/10.1016/j.epsr.2011.07.020 (2011)
64 Montavon G., Orr G. and Müller, K. R. Neural networks-tricks of the trade second edition. Springer, DOI, 10, 978-3 (2012)
65 Domhan T., Springenberg J. and Hutter F. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In IJCAI International Joint Conference on Artificial Intelligence (Vol. 2015–Janua, pp. 3460–3468 (2015)
66 Witt Nils, and Christin Seifert. Understanding the Influence of Hyperparameters on Text Embeddings for Text Classification Tasks. In Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 10450 LNCS:193–204. https://doi.org/10.1007/978-3-319-67008-9_16 (2017)
67 Yann Dupis. Amazon Fine Food Reviews Sentiment Analysis with Recurrent Neural Network. https://github.com/yanndupis/RNN-Amazon-Fine-Food-Reviews (2017)
78 Zeng D., Liu K., Lai S., Zhou G. and Zhao J. Relation Classification via

Convolutional Deep Neural Network. (2014)
79 Pedregosa et al., JMLR 12, pp. 2825-2830. Scikit-learn: Machine Learning in Python(2011)
80 Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality (2013)