COMP 551 Project 2
Report

1. Abstract:

This project is about using different machine learning techniques to predict the sentiment of movie reviews on the IMDB website. The reviews are in the form of text sentences, and sentiments are classified as positive or negative.
In order to extract features from reviews, we first pre-process the text by applying standard techniques such as filtering out stopwords and punctuations, then some feature extraction methods such as n-gram, binary occurrence, and tf-idf are employed. The eight models we tried in classification include Ridge classifier, decision tree, logistic regression, and linear support vector machine. We found that n-gram and tf-idf generally improve the accuracy of prediction. Among these eight models, decision tree classifier is the least accurate, while support vector machine is most accurate.
Also, k-cross validation is used to test how good these models are on the training set before running them against the testing set, and we found that the test set shows a slightly higher accuracy than the result from the k-cross validation, and this is possibly due to the fact that we used 75% training data to select the model and 100% training data to predict on the test set.

2. Introduction:

The task is to train different models in order to predict the sentiment of IMDB reviews. The training dataset consists of 25000 individual text reviews, along with their sentiment, which has the value 1 for positive, and 0 for negative.
We found that the best way to extract features is to perform td-idf with ngram (with n-values 1, 2, and 3) on lowercased reviews that have their punctuations removed. Using the best way to extract features and the best hyperparameter configuration, our best model is linear support vector machine with an accuracy of 90.19.
We customized these models using techniques such as L2 regularization, and we found that a good adjustment of hyper-parameters, such as the regularization strength and verbose, is important for improving accuracy.

3. Related Work:

Sentiment Analysis is a method or technique to detect and extract subjective information, such as opinion and attributes, from language. There has been a tremendous rise in the research being carried out on the related topics in the last 2 decades.

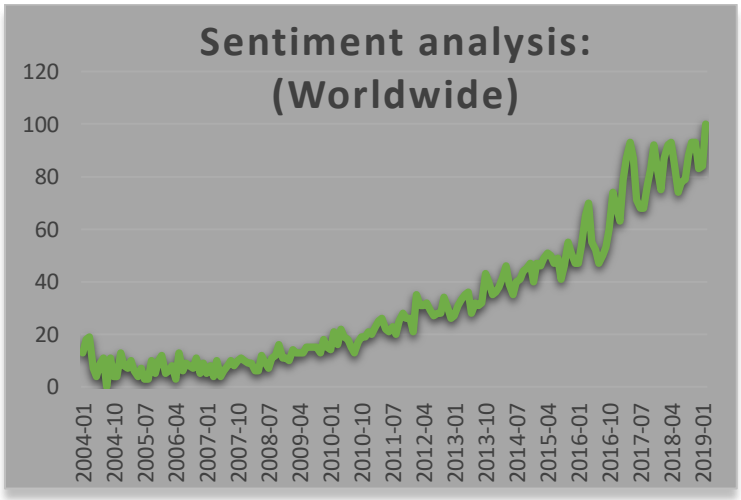

Figure 1 Rise in the interest in the topic 'sentiment analysis' for the past 15 years [1] (y-axis represents the interest with a maximum possible score of 100)

An extensive literature survey was carried out before exploring the dataset or the learning model. Most of the research papers talk about a similar process flow, processing a text, selecting a learning model and validating the model [2] [3] [4]. Few papers have reported using more than one classifier at the same time, using some to classify the test data and using one meta classifier to use the output of previous models to make a final prediction in a process called stacking classifier.

4. Preprocessing:

To perform an analysis, preprocessing of the text is always an important step, because the text may contain expressions that are not relevant to the analysis. We performed the following steps to clean the text.

1. Getting rid of punctuation by removing a set of symbols from the text, followed by lower casing the entire text.
2. Removing the most common words in the English language, called the stop words. These words do not carry important meaning and are usually removed from the texts.
3. Reducing the words to their word stem, base or root form using a module from NLTK. The module is called Porter Stemming and it removes the common morphological endings from the words.
4. Lemmatization is like stemming in the sense that it reduces the words to a common base form. But as opposed to stemming, lemmatization does not simply chop off endings, instead it uses lexical knowledge bases to get correct base forms of the words.

5. Feature Extraction Pipelines:

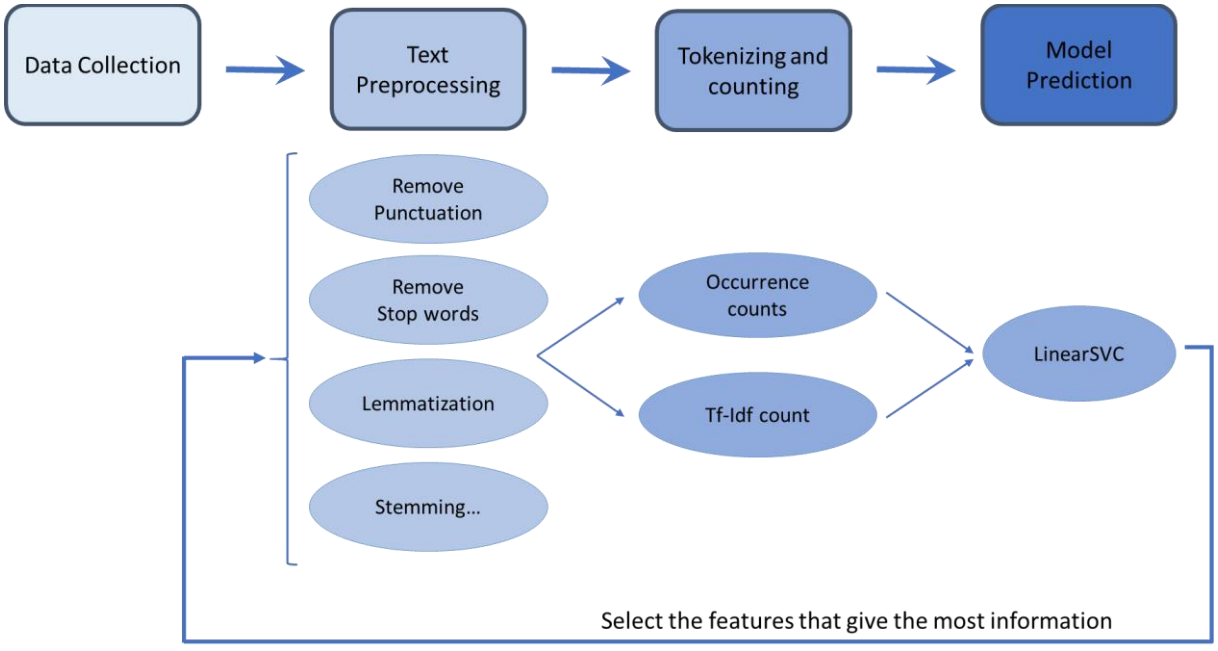

Figure 2: The process flow for feature extraction

The features used in the model are extracted and selected by using the pipeline shown in Figure 2. The extraction pipeline consists of three main steps:

(1.) Text Preprocessing: The collected input data are real-world comments which are “messy” to our models. Thus, text preprocessing methods are needed before further analyzing the data. Standard methods such as removing stop words, replacing punctuations, and lowercasing all words have been studied. Furthermore, text lemmatizing and stemming are also tested

(2.) Words (phrase) tokenizing: The processed texts are then tokenized and converted into numerical values by the occurrence counts of words or their Tf-Idf weighting (two vectorizers from sklearn library are used: CountVectorizer and TfidfVectorizer). Phrase counts (N-grams) are also considered during the tokenizing and counting process.

CountVectorizer: This sklearn Vectorizer automatically removes the punctuations, lowercases the words from your input text and returns the counts of each token calculated by their occurrence.

TfidfVectorizer: Similar to CountVectorizer; however, instead of using the occurrence counts, this Vectorizer utilizes the concept of "Term Frequency-Inverse Document Frequency" which makes sure that less importance is given to the most frequent words.

(3.) Model prediction: The features (tokenized data) are then fitted into prediction models and evaluate the performance of the features extracted from step (1) & (2). LinearSVC from sklearn library is selected to evaluate the accuracy of prediction by using a 10-fold cross-validation method.

The accuracy of a standard processing method (remove punctuations and lowercasing words) using both CountVectorizer and TfidfVectorizer are first calculated as a reference. The new feature extracting methods from step (1) and (2) are then tested separately and compared to this reference; extracting methods with higher performance can then be selected. The results of some of the feature extraction results from the testing pipeline are shown in Figure 3. By the results, we can observe that the using of Tf-idf weighting gives us better performance when compared to occurrence counts. Furthermore, we can anticipate that by combining the extracting methods with most improvements (text lemmatizing, binary occurrence, and ngrams with n = 3), we should obtain our best prediction among all the methods we implemented.

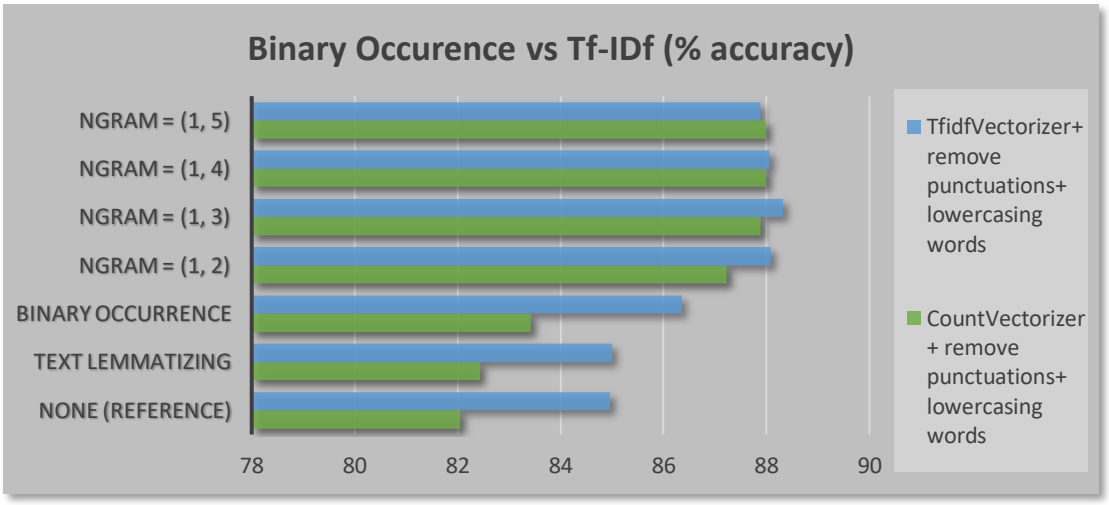

Figure 3: Comparison of implemented pipelines: Word Count vs Tf-Idf

6. Training:

For training the model, the SKLearn library was explored for a model that provides the maximum accuracy over the training set. The models explored were 1. Decision Tree Classifier 2. Multinomial Naïve Bayes 3. Bernoulli Naïve Bayes 4. Logistic Regression 5. Linear SVC 6. Random Forest Classifier 7. Extra Trees Classifier and 8. Ridge Classifier. The cross-validation feature of SKLearn library was used to eliminate the probability of overfitting the model and to estimate the skill of machine learning. We used 10-fold cross validation to compare and select a model for the given prediction model. The results in the form of the mean of the accuracies of 10-fold cross-validations are presented in the next section.

Three best models (based on the accuracy score), Linear Regression, Ridge Classifier and Linear SVC were then selected and were used for further exploration.

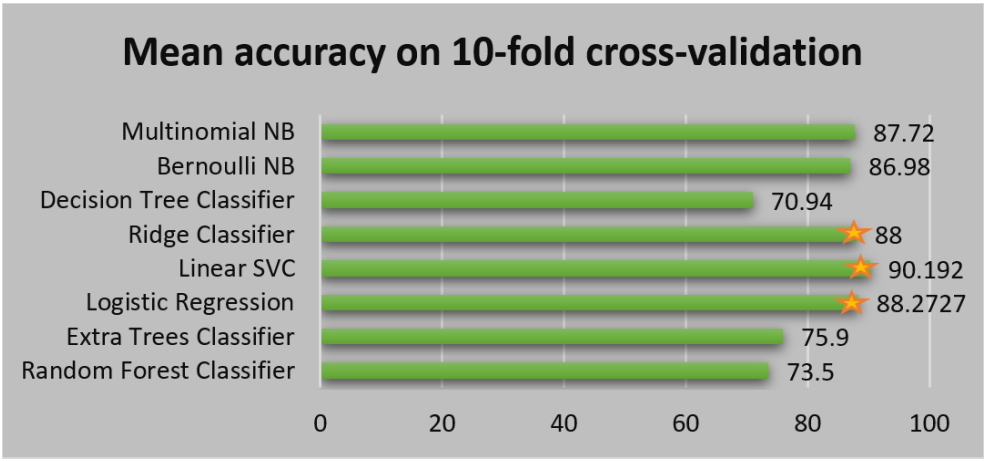

Figure 4: Comparison of 8 different learning models

7. Grid search to find best hyperparameters:

Grid search is an exhaustive search over specified parameter values for an estimator. We ran grid search to find the best hyper-parameters to improve our accuracy compared to the models with hyperparameters set to default.

parameters_LR = [{'C': [0.1, 0.5, 1.0, 10.0], 'penalty': ['l2'], 'tol': [0.0001, 0.00001]}]

parameters_SVC = [{'C': [0.1, 1, 10], 'penalty': ['l2'], 'max_iter': [1000, 10000], 'tol':[0.0001, 0.00001]}]

parameters_RC = [{'normalize': ['True', 'False'], 'alpha': [0.001, 0.0001]}]

In case of Logistic regression, we tried different value of the hyperparameter *'penalty', 'tol', 'C'*, etc. On experimenting with different values of these parameters we found out the best set of hyperparameters for Logistic Regression are '*penalty*' set to *'l2*', *'C*' set to 10 and *'tol'* set to 0.0001.

In case of Support Vector Machines, the best set of parameters found grid-search were 'C' set to 10, penalty set to 'l2', '*max_iters'* set to 1000 and '*tol*' set to 0.0001.

And for Ridge Classifier, the grid search was performed over the hyperparameters '*normalize*' and '*alpha*'. We found out that setting '*normalize*' to '*False*' and '*alpha*' to '0.001' gives the best performance.

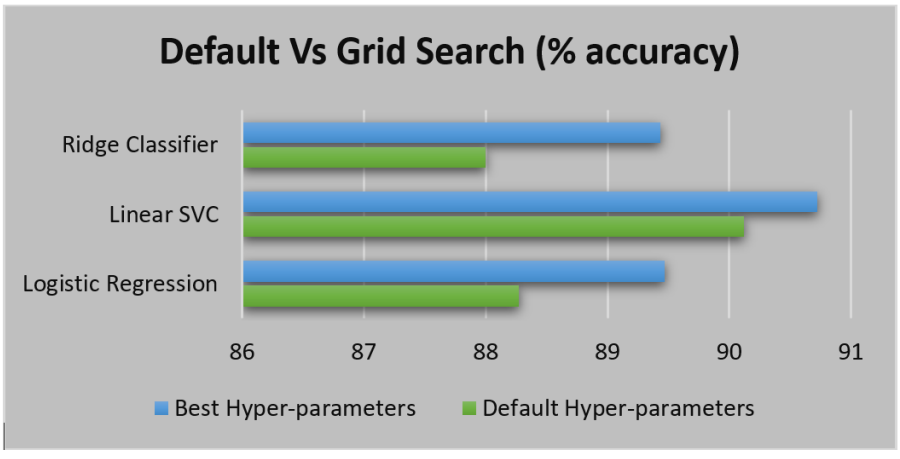

Figure 5: Comparison between the models with hyperparameters set to default vs best hyperparameters

We see an improvement of 1.39% for Logistic Regression, an improvement of 0.66% for Linear SVC and an improvement of 1.625% in case of Ridge Classifier.

Upon submission to Kaggle, we were able to achieve an accuracy of above 91% using the best performing hyper-parameters and the most useful features.

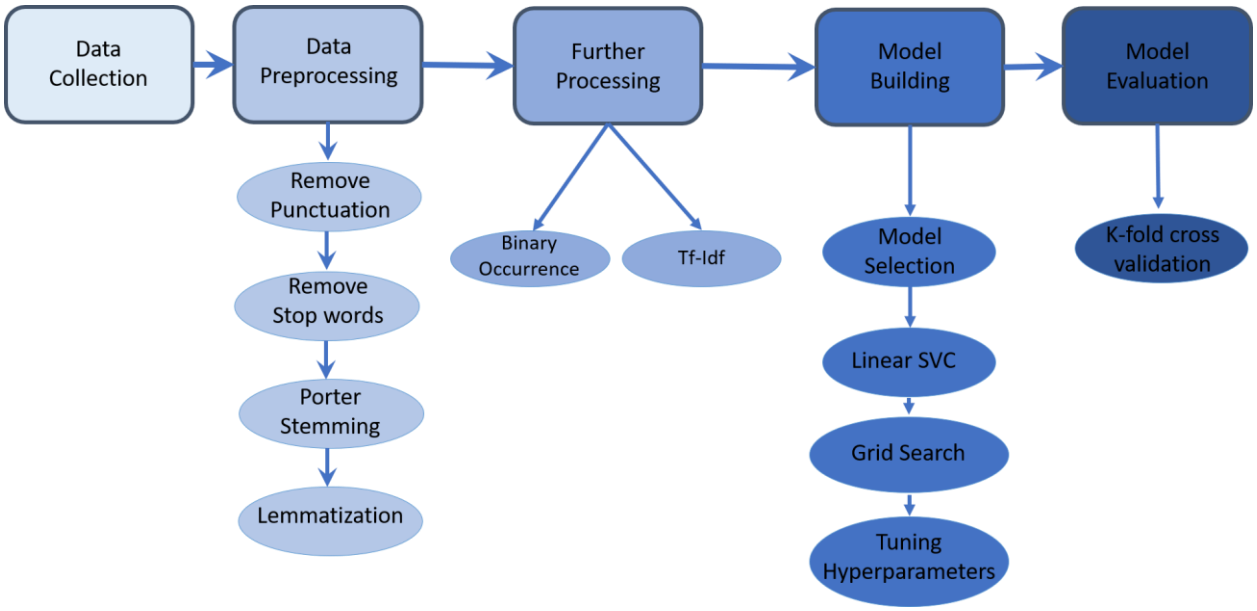

Figure 6: Flow chart of the entire process

8. Discussion and Conclusion: We would like to summarize the takeaways and directions for the future work as follows:

a. The accuracy could have been improved by further extraction of new features. These features could have been part-of-speech tagging, using sentiment value from external libraries, etc.
b. We could have used a stacking classifier as mentioned in few of the research papers we studied.
c. There were few models that we were not able to explore because of computational resource limitations, rbf-SVM, XG_Boost, etc.

Statement of Contribution:

Yu-Cheng Cho: Naïve Bayes from scratch, extensive feature extraction (Occurrence Count vs Tf-Idf), part of Linear SVC and Logistic Regression

Navpreet Singh: Implementation of Logistic regression and Linear SVC, feature extraction, Grid search and part of Naïve Bayes.

Linck Wei: feature extraction, tf-idf and the techniques in the tf-idf library, k-fold cross validation

References

[1] Google Trends [https://trends.google.com/trends/explore?date=all&q=%2Fm%2F0g57xn]

[2] Chapter 'Extracting Product Features and Opinions from Reviews' by Popescu, Ana-Maria, from book 'Natural Language Processing and Text Mining'

[3] Bo Pang and Lillian Lee (2008), "Opinion Mining and Sentiment Analysis", Foundations and Trends® in Information Retrieval: Vol. 2: No. 1–2, pp 1-135. http://dx.doi.org/10.1561/1500000011

[4] Mika V.Mäntylä, Daniel Graziotin and Miikka Kuutila The evolution of sentiment analysis—A review of research topics, venues, and top cited papers

[5] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[6] https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

[7] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[8] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier

[9] https://scikit-learn.org/stable/modules/generated/sklearn.tree.ExtraTreeRegressor.html#sklearn.tree.ExtraTreeRegressor

[10] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier