COMP 551 Mini-Project 2

Group 5
Eric leung, ID 260720788
Donovan Chiazzese, ID 260742780

I. Abstract

IMDB (Internet Movie Database) is an online database of information related to films television programs, home videos and video games, and internet streams, including cast, plot summaries, and fan reviews and ratings. In this project, we developed predictive models to classify the sentiment of IMDB reviews. Our goal was to classify a IMDB reviews as either positive or negative based on the language they contain. We found that the more features we added the better our model performed, but unfortunately the curve obeyed the law of diminishing returns.

II. Introduction

In mini-project 2, we applied a variety of classifiers to classify 25,000 IMDb reviews (all in .txt format) as positive or negative. A Bernoulli Naive Bayes model was implemented from scratch (without using any external libraries) as required. We then were tasked with implementing 2 or more other models using Scikit learn to compare the performance in movie review sentiment analysis. We implemented logistic regression, multinomial Naive Bayes, decision trees and K-nearest Neighbours.

III. Related work

(This article is cited from the following link: https://arxiv.org/pdf/1711.10377.pdf)

It is a research paper on Twitter sentiment analysis using Machine Learning Algorithms on Python. Twitter sentiment analysis is an application of sentiment analysis on data which are retrieved from Twitter (tweets). Since the past decades, the research field on sentiment analysis has grown tremendously. Moreover, one of the hardest challenges for sentiment analysis is that each tweet has a limit of 140 characters. Thus, this causes users to use different slang words and abbreviations which are difficult to extract their sentiment. The approach to extract sentiment from tweets, 1) Downloading and caching the sentiment dictionary, 2) Importing the data from Twitter, 3) Filter out the stop words, 4) Tokenize each word in the dataset, inputting them into the program, 5) Compare each word with both positive and negative sentiments in the dictionary, then increment positive count or negative count , 6) Obtain the percentage of sentiment based on the positive count and negative count in order to decide the polarity. It was found that the neutral sentiment for tweets are significantly high, this indicates the limitations of the current works in the industry and there is a need to improve the sentiment analysis on Twitter.

IV. Dataset and setup

In order to run our experiments, we needed to preprocess the data which were given to us. The dataset was given to us in two main directories called train (our training set), and test (our test set). Moreover, we first looped through each file (review) in the training set using the *os.listdir()* method from the os module in Python. Last but not least, we extracted the content of the review and split the review by whitespace. Since $os.listdir()$ lists out the files within the directory in random order, we took the part of the filename (string) which is before the underscore index as the file's order in our training matrix X. Thus, we finished preprocessing our dataset, and the order of the files were preserved as well.

V. Proposed approach

Throughout this project, we used various models and features to classify the given dataset (IMDb reviews):

- Bernoulli Naive Bayes: Naive Bayes as described in class is typically a strong classification model in spite of its simplicity; It is competitive among state of the art models in text analysis. We implemented naive bayes as a product of each feature's probability for each class. We classified the reviews as positive if the positive product of probabilities was greater than the negative and

vice versa.

The features extracted were the top 10000 words. We then filtered these by "information gain", removing any features that did not have a percent difference between positive and negative reviews over 40%. This reduced the amount of "noise" in our model. The remaining 4669 features were used throughout all the models. We first experimented the results of our predictions with binary occurrence, and then tf-idf weighting. Tf-idf weighting performed marginally better, by about $\approx$1% on our Kaggle submission. As per usual practice with k-fold cross-validation, we split our dataset into 10 subsets, cycling the validation set through each one of the 10 subsets. This helped reduce overfitting and improved our model on more general data. We were careful to extract our features from only the training set and not the validation set for each iteration. As a result, we had to write the code by hand instead of using the Sci-kit learn library.

- Logistic Regression classifier: This model acted as the base model for task 3 and 4. This model was implemented using the SciKit Learn module as well in a similar fashion as the Decision Tree classifier, by calling $model = LogisticRegression()$. In addition, for the feature part, we first extracted the 50,000 most frequent words appeared among all reviews. Next, we compared the occurrence frequency precent difference of a word between positive reviews and negative reviews in order to obtain the features which will provide the **most information gain**.

  Lastly, instead of calculating the training data matrix, we stored our training dataset into a .pk file, just to make our testing process runs faster.

- Decision Tree model: It was imported from the SciKit Learn library, using the following line of code, $model = tree.DecisionTreeClassifier()$. As for the input training data, our Decision Tree model uses the same feature as our Logistic Regression classifier. We checked the occurrence of each adjective in the reviews text files (both pos and neg directories). Then we store the occurrences into a matrix called **trainingDataDT**.

  On the other hand, the target parameter depends on which directory the review belongs to, say if review x is contained in the **pos** directory, then the target variable (t) for x will be $t_x = 1$ and $t_x = 0$ if x is from the **neg** directory.

VI. Results

On average the runtime of logistic regression was the quickest. To avoid longer runtimes we imported the pickle library to save our features onto a file once we extracted them from the dataset. This saved us hours of runtime during experimentation. The best performing model out of Naive Bayes, Logistic Regression, Decision Trees, K-Nearest Neighbours was by far **Logistic Regression**, naturally, as Logistic Regression is appropriate when the dependent variable is binary. Another natural fit is that typically our features don't typically exceed values of ± 3.29 which is optimal for logistic regression.

We applied these practices and features to each Naive Bayes, linear regression, k-nearest neighbors and decision trees. Among the 4 models, linear regression performed best as expected. Linear regression is a strong model for binary feature sets. The order of performance on Kaggle was as follows:

TABLE I
BERNOULLI NAIVE BAYES

	Test dataset
Kaggle Score	0.72893 (tf-idf)
Kaggle Score	0.72893 (Binary-occurrence)
Classified as positive	50.8%
Classified as negative	49.2%

TABLE II
DECISION TREE MODEL

	Test dataset
Kaggle Score	0.69626
Classified as positive	50.2%
Classified as negative	49.8%

The runtimes include calculating the feature values, fitting the model with Sci-Kit learn and prediction:

Discussions and Conclusion

These implementations were a learning experience. A key takeaway is that running these algorithms is expensive and time-consuming. A trick that we developed

TABLE III
LOGISTIC REGRESSION MODEL

	Test dataset
Kaggle Score	0.84320
Classified as positive	50.4%
Classified as negative	49.6%

TABLE IV
K NEAREST NEIGHBOURS

	Test dataset
Kaggle Score	
Classified as positive	%
Classified as negative	%

early on was to use the pickle library to save our X matrix and features for future use without having to extract and calculate them every time. A useful tool that would have likely been effective but did not fit into our time budget was ensembling. It would have been a good experience to see the performance of these models when they have been stacked.

STATEMENT OF CONTRIBUTIONS

Donovan Chiazzese implemented the Bernoulli Naive Bayes model, applied tf-idf weighting for feature extraction, and K-nearest neighbours.

Eric Leung managed the data preprocessing part, feature extraction pipeline using binary occurrences, also ran experiments on the Decision Tree and Logistic Regression classifiers using K-fold cross validation.

The project writeup was completed by both Eric and Donovan.

TABLE V
RUN TIME FOR EACH MODEL (IN MINUTES)

	Model
Bernoulli Naive Bayes	2.58
Decision Tree	5.49
Logistic Regression	3.21
K-nearest neighbour	7.01