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 were then tasked with implementing 2 or more other models using Sci-kit learn to compare the performance in movie review sentiment analysis. We implemented Logistic Regression, Decision Trees and K-nearest Neighbours and the strongest model by a solid margin was logistic regression. This was no surprise as logistic regression tends to do well with binary data.

III. Related work

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

A common step when dealing with text data is pre-processing and tokenizing the data. A general problem with modern data, especially from casual web users is it is full of grammatical errors and slang. This is an issue that Hamid Bag and Md Johirul Islam ran into when performing their research 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). One of the hardest challenges for sentiment analysis is that each tweet has a limit of 140 characters, this causes users to use different slang words and abbreviations which which makes extracting their sentiment difficult. The approach to extract sentiment from tweets is, 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 that 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 provided data. We were provided with two main directories called train (our training set), and test (our test set). We first looped through each file (review) in the training set using the *os.listdir()* method from the os module in Python. Then, 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, the order of the files were preserved.

V. Proposed approach

Throughout this project, we used various models and features to classify the given dataset (IMDB reviews): 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. These features were then filtered by "information gain", removing those that did not have a percent difference between positive and negative reviews over 40%. This significantly reduced the amount of "noise" in our model. The remaining 4669 features were used in all experiments with the other models. We first tested the results of the predictions with binary occurrence, and then tf-idf weighting. Tf-idf weighting performed marginally better, by about $\approx$1% on Kaggle.

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%

In order to regularize our features, we initially populated our X matrix with ones instead of zeros; This emulated Laplace smoothing. As a result, every feature had an initial occurrence of one. Once this was implemented, L2 regularization did not perform well, since Laplace smoothing was already a form of regularization, and actually weakened/underfit our model. We tried the two separately, and Laplace smoothing performed better by 0.21%. Initially improving the model by 0.31%. This remained the case throughout all the models.

TABLE II
PERFORMANCE

	Validation set
Without regularization	87.77%
Laplace Smoothing	88.08%
L2 regularization	87.866%
L2 + Laplace	80.715%

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.

We applied these practices and features to each Naive Bayes, Logistic Regression, K-nearest neighbors and Decision Trees. Among the 4 models, Logistic Regression performed the best as expected. Logistic Regression is a strong model for binary feature sets.

KNN on the other hand would probably have done better if we had a larger dataset and much less noisy data. Lastly, decision trees was no surprise as this type of classification analysis requires many features, and decision trees become complex and convoluted when there are too many features.

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 dictionary onto a file once we extracted them from the dataset. This saved us hours of runtime during experimentation.

TABLE III
DECISION TREE MODEL

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

TABLE IV
LOGISTIC REGRESSION MODEL

	Test dataset
Kaggle Score	0.86853
Classified as positive	50.4%
Classified as negative	49.6%

TABLE V
K NEAREST NEIGHBOURS

	Test dataset
Kaggle Score	0.68253
Classified as positive	51.81%
Classified as negative	48.19%

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

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

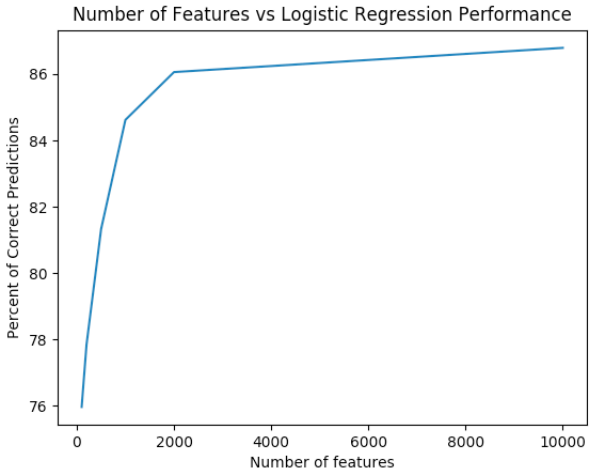
TABLE VI

RUN TIME FOR EACH MODEL

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

variable is binary. Secondary, our features don't typically exceed values of ± 3.29, which is optimal for logistic regression. Logistic regression performed better as the number of features grew, but exhibited the law of diminishing returns.

We tested the performance on the a validation set with 100, 200, 500, 1000, 2000 and 10000 features.



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 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.