**Classification of Movie Sentiments**

CIS 4500 Term Project

November 28, 2019

Miriam Snow

0954174

# 1 Introduction

### Problem and Applications

The problem being addressed in this project is classification and analysis of movie review sentiments. To do this, machine learning techniques were used to analyze a balanced data set of positive and negative movie reviews. Each machine learning technique was compared against each other to determine which one most accurately classifies which reviews are indeed positive or negative.

The major applications here are plentiful, however, a movie review website like *IMDB* or *Rotten Tomatoes* may find a solution like this particularly useful. Although users on movie review websites are able to rate movies with a thumbs up (positive) or thumbs down (negative), with tools such as the ones implemented in this project, users can simply write in natural language their sentiments about the film. The system would then be able to analyze and classify whether the sentiment is positive or negative.

### Implementation Overview

For this project, implementation was done with Python 2.7.15. Python is an interpreted language meaning execution is rather slow, but with packages like NumPy and SciPy, vectorized operations perform relatively fast. The main machine learning methods and techniques used for feature selection and classification were derived from Python's Scikit-learn package.

Two different feature selection techniques and three different classification techniques were implemented and used for experimentation. The purpose of the feature selection techniques is to select $x$ number of top features associated with positive and negative reviews based on a calculated value. These can then be used to train the classifier and test each sentiment based on the most common features used for training.

The two feature selection techniques used in this project were "Count Vectorizer" (or bag of words as it is more commonly known) and "TFIDF Vectorizer". The three classification techniques used were "Multinomial Naive Bayes", "Logistic Regression" and "Linear Support Vector Classification".

All combinations of the feature selection techniques and classification techniques were tested along with 6 different feature selections sizes; 500, 1000, 1500, 2000, 2500, and 3000. This resulted in 36 combinations and experimentation results.

### Data Set

The data set used for performing these experiments was released at Cornell University in 2004. It is a collection of 1000 negative and 1000 positive movie review sentiments, each in different text documents. The data has already been preprocessed via tokenization, sentence splitting and case

normalization. However, further preprocessing was done before performing the experiments for this project, in an attempt to render better results.

# 2 Techniques

### Count Vectorizer Feature Selection

This technique, more commonly known as "bag of words" simply selects features based on how frequently they are used. Therefore, the top 500 features in a Count Vectorizer feature selection implementation, would be the top 500 most frequently used words. This method produces a vector that lists the document frequencies of each feature in each document.

In the example below, there are three documents listed, and after executing the Count Vectorizer, a display of all of the feature names can be shown. After this, the vector is displayed where each column represents its corresponding feature, and each row represents its corresponding document.

```
Documents:
some like it hot, some like it cold
some like it really hot
really hot is too hot

List of Features:
cold hot is it like really some too

Count Vectorizer Vector:
[[1 1 0 2 2 0 2 0]
 [0 1 0 1 1 1 1 0]
 [0 2 1 0 0 1 0 1]]
```

Figure 2.1 Count Vectorizer Example

This means, to identify the frequency of "hot" in document 3, the value to be considered is at the second column and third row, since "hot" is the second word in the list of features. While performing experiments for this project, this vector is used to select the top $x$ number of most frequent features.

A built-in list of stop words from the Scikit-learn package are automatically removed from the vector while performing this type of feature selection. Stop words are extremely common words like "the", "and", "a", and are ignored since they are most likely to appear in both negative and positive reviews. For the purpose of this project, we only want to focus on content filled words that will prove to be useful in classification techniques.

### TFIDF Vectorizer Feature Selection

Similarly to the Count Vectorizer technique, the TFIDF Vectorizer technique selects features based on how frequently they are used in each document, but also takes into consideration the inverse document frequency value. These two values are computed together to retrieve the weight of a term using the formula below.

$$w_{ij} = tf_{ij} * log(N/df_j)$$

The weight of term $j$ is computed by multiplying the term frequency of term $j$ in document $i$, by the log of the total number of documents divided by the number of documents that contain term $j$.

Therefore, the top 500 features in a TFIDF Vectorizer feature selection implementation would be the top 500 words with the highest weight value. Similar to the Count Vectorizer, this method also produces a vector, however, the weights of each feature in each document are listed, rather than the frequency.

```
Documents:
some like it hot, some like it cold
some like it really hot
really hot is too hot

List of Features:
cold hot is it like really some too

TFIDF Vectorizer Vector:
[ 0.35 0.21 0.00 0.53 0.53 0.00 0.53 0.00]
[ 0.00 0.36 0.00 0.47 0.47 0.47 0.47 0.00]
[ 0.00 0.59 0.50 0.00 0.00 0.38 0.00 0.50]
```

Figure 2.2 TFIDF Vectorizer Example

In the example above the same three documents are displayed along with a list of feature names. After executing the TFIDF Vectorizer, the vector is displayed where each column once again represents its corresponding feature and each row represents its corresponding document.

During implementation of this technique, stop words were once again removed and a maximum document frequency of 70% was specified. This means that words located in more than 70% of the documents are removed, as they are likely too common and therefore not useful in classification.

**Multinomial Naive Bayes Classifier**

The Multinomial Naive Bayes Classifier technique is an extension of Bayes' theorem which suggests that:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

Using this equation, Multinomial Naive Bayes determines a probability value for each term in the document to perform the classification analysis. The classic Naive Bayes classifier simply takes into account whether a term exists in a document or not. However, the Multinomial Naive Bayes classifier is more specific because the term's frequency or weight (depending on which feature selection method is used) is also taken into consideration and used to calculate the probability value.

After the negative and positive probability values for each term are computed, they are compared to determine whether the document should be classified as positive or negative.

For the document containing the terms "I like it cold", an example of a Multinomial Naive Bayes classification calculation might be as follows.

$$P(i\ like\ it\ cold|pos) = P(i|pos) * P(like|pos) * P(it|pos) * P(cold|pos)$$

$$P(i\ like\ it\ cold|neg) = P(i|neg) * P(like|neg) * P(it|neg) * P(cold|neg)$$

*if* $P(i\ like\ it\ cold|neg) > P(i\ like\ it\ cold|pos)$ *then "i like it cold" is classified as negative else "i like it cold" is classified as positive*

All Naive Bayes classification techniques assume that all features are independent of each other. Independence can be problematic and require smoothing, however, for the purpose of this project, the model still performs reasonably well during each experiment, and therefore no smoothing is required.

**Logistic Regression Classifier**

This technique is a statistical classification technique which is similar to linear regression, however linear regression determines a continuous output, while logistic regression predicts an output using the sigmoid function to calculate the probability that the document should be classified as positive or negative.

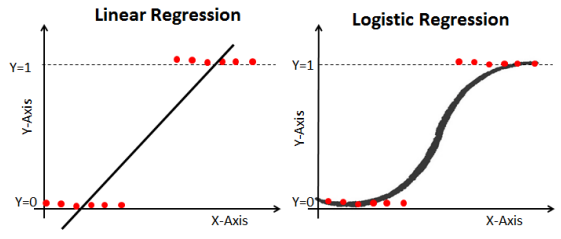


Figure 2.3 Linear vs Logistic Regression

The sigmoid function causes the logistic regression curve to appear in an s shape rather than a straight line. The equation is seen below.

$$f(x) = \frac{1}{1 + e^{-(x)}}$$

For each document, if the sigmoid function produces an output value of 0.5 or higher, the document has at least a 50% chance of being positive and is therefore classified as positive. Otherwise, the document has less than a 50% chance of being positive and is therefore classified as negative.

For the purpose of this project, the default "lbfgs" solver was used (which stands for Limited memory Broyden–Fletcher–Goldfarb–Shanno algorithm) with the maximum number of iterations being specified at 500.

**Linear Support Vector Classification Classifier**

The Linear Support Vector Classification technique is a distance based classifier, rather than a probability based classifier like Multinomial Naive Bayes and Logistic Regression.

The theory behind Linear Support Vector Classification is the idea of a model representing data as points (called support vectors) on either side of a line (called the dividing hyperplane). In our case, these points would be positive and negative documents.

A wide gap surrounds the line (called the best margin), and when new data is added, a decision equation is used to decide where it is placed in the gap. Data is then classified as positive or negative depending on which side of the line it is closest to. See the figure below for a visual representation of this technique.

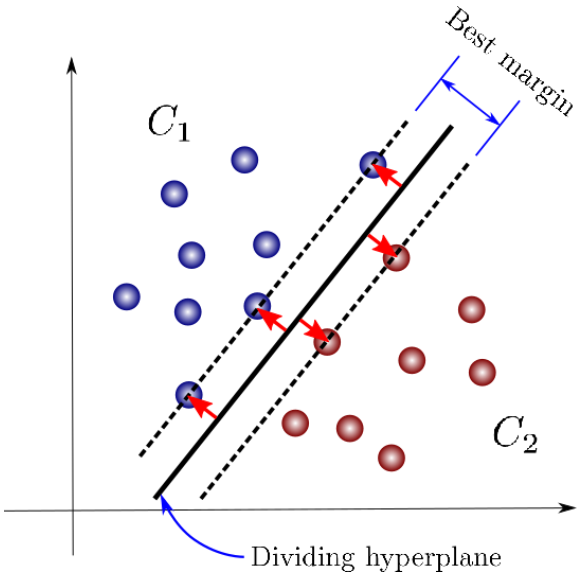


Figure 2.4 Linear Support Vector Model

For the purpose of this assignment, the maximum number of iterations for implementation of the Linear Support Vector Classification technique was specified at 15,000.

# 3 Implementation

### Data Analysis

The first task was to perform data analysis on the entire data set. This was done by calculating the number of tokens, the number of sentences, and the average sentence length by token for each document. In addition, the total average number of tokens, sentences and sentence length for each collection was computed. These values are displayed in two separate CSV files, one for the positive data set collection, and one for the negative data set collection.

This data analysis is useful to compare against the results of each feature selection, feature size and classification technique combination after the experiments have been completed.

### Preprocessing

The original data set given for the project is balanced, randomized and split into two directories each containing 1000 positive and negative documents, respectively. In addition, each document is minimally preprocessed via tokenization, sentence splitting, and case normalization, however further preprocessing needed to be done for implementation of this project.

To begin, the data set was split into two different sections, the first being the validation set which is the top 15% of the data and represents the “held out data”. This was set aside to validate the best performing models at a later point. The remaining 85% of the data set includes 850 documents from the positive and negative directories, respectively, totalling 1700 documents, and was used to perform 5-Fold Cross Validation.
After the data was split, it was formatted into a CSV file, making it much easier to work with when implementing feature

selection and classification techniques. This is because it can be easily turned into a data frame using the Pandas Python package. The CSV file contains two columns; “Sentiment” which indicates whether the document was positive (1) or negative (0), and “SentimentText” which lists the contents of the document, without newlines or commas.

Even though the original data set was already minimally preprocessed, for the purpose of this assignment, punctuation removal and stemming was also performed. Stemming was completed by using the Python NLTK stem package, namely Porter Stemmer. Stop words were removed in the feature selection stage as indicated previously.

**K-Fold Cross Validation**

K-Fold Cross Validation is a method of splitting the data set into $k$ number of sections. These sections are then used for training and testing the model through a series of iterations, using a different train and test section each time.

For example, a three fold cross validation implementation may follow the following series of iterations.

1. train on section two and three, test on section one
2. train on section one and three, test on section two
3. train on section one and two, test on section three

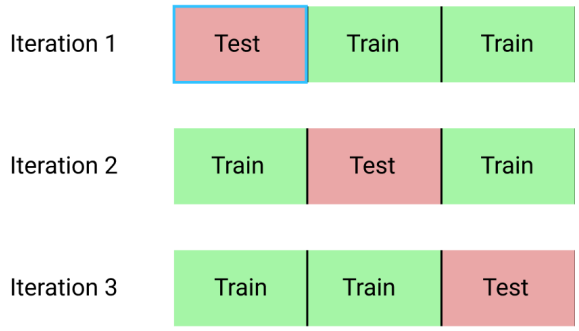


Figure 3.2 Three Fold Cross Validation

For the purpose of this project, the CSV file containing the latter 85% of the data was read into a data frame and 5-Fold Cross Validation was performed. This means for each iteration of the Cross Validation evaluation, the data was split up into an 80% training set and 20% test set.

The training set is first used for execution of a feature selection technique, and the top $x$ number of features are chosen. The feature selection technique and maximum number of features was specified based on the experiment being tested. Then, the selected top features are used in accordance with a classification technique of choice to measure the performance of the classifier on the test set.

To determine how well the feature selection, feature size and classification technique combination performed, an accuracy score, confusion matrix and F-Measure score were computed for each iteration. These scores are then totalled together to determine an average accuracy score for the model.

5-Fold Cross Validation was completed on all 36 combinations of feature, feature size,

and classifier. Then, the results were compared to determine which model had the best average performance score.

**Validation**

After the top performing models are found based on the results gathered from 5-Fold Cross Validation, the same models are evaluated using the "held out data" that was set aside originally. To do this, the entire data set was split into a 15% test set and 85% training set.

This "held out data" will provide unbiased results on how well the model is performing since it is using never-before-seen data that was not used with the 5-Fold Cross Validation evaluation method. The results of this validation are compared with the results from the K-Fold Cross Validation method as seen in the tables below.

# 4 Results

After performing all 36 combinations of feature selection, feature size and classifier techniques, the results summarized in the tables below indicate the F-Measure scores gathered. In each table, the yellow highlighted values indicate the top best performing models.

To better understand the values computed in the tables below and displayed to the command line when running each experiment, refer to the following formulas.

| | Classified 1 | Classified 0 |
|---|---|---|
| Actually 1 | a<br>true positive | b<br>false negative |
| Actually 0 | c<br>false positive | d<br>true negative |

Figure 4.1 Confusion Matrix

The figure above represents a confusion matrix. A true positive and true negative means that the document was correctly classified as positive or negative, respectively. A false positive means that the document was classified as positive, but it is actually a negative sentiment. A false negative means that the document was classified as negative, but it is actually a positive sentiment.

An accuracy score is determined by the following formula and depicts a measurement of how "accurate" the model is. Unfortunately, this score focuses only on the true positive and true negative values, while not taking into account the false negative and false positive values.

$$Accuracy = \frac{a+d}{a+b+c+d}$$

Therefore, an F-Measure score was computed using the following formula, and used for comparison. This score takes into account both false negatives and false positives and is a better depiction of how well the model is performing.

$$FMeasure = \frac{2a}{2a+b+c}$$

**5-Fold Cross Validation**
**F-Measure Scores (in percentages)**

| | **Multinomial Naive Bayes** | **Logistic Regression** | **Linear Support Vector Classification** |
|---|---|---|---|
| **Count Vectorizer 500** | 77.54101639563297 | 76.38542145626776 | 74.43191703858673 |
| **Count Vectorizer 1000** | 79.95871899035765 | 80.95762097731708 | 79.82575705925387 |
| **Count Vectorizer 1500** | 80.40294638907847 | 81.90787894316085 | 81.22540216484552 |
| **Count Vectorizer 2000** | 80.18767712666647 | 82.71285650167638 | 81.08638867827892 |
| **Count Vectorizer 2500** | 80.80278277400902 | 83.72206133045586 | 82.21255509088247 |
| **Count Vectorizer 3000** | 80.78237793664134 | 84.44538062628399 | 82.59536556340737 |
| **TFIDF Vectorizer 500** | 77.93414418538221 | 80.01922163123547 | 78.71503266108608 |
| **TFIDF Vectorizer 1000** | 80.04374803820141 | 83.56951211501112 | 81.93346085381522 |
| **TFIDF Vectorizer 1500** | 80.02686604625897 | 83.64630908680172 | 83.81063333934196 |
| **TFIDF Vectorizer 2000** | 79.02121233628927 | 83.24998568554265 | 83.97112032261356 |
| **TFIDF Vectorizer 2500** | 80.1489362147011 | 83.73843699203529 | 84.25847971342785 |
| **TFIDF Vectorizer 3000** | 80.01812889826272 | 83.97573269981656 | 84.64976809705327 |

**Validation Set**
**F-Measure Scores (in percentages)**

| | **Multinomial Naive Bayes** | **Logistic Regression** | **Linear Support Vector Classification** |
|---|---|---|---|
| **Count Vectorizer 500** | 71.03448275862068 | 72.42524916943522 | 69.73684210526315 |
| **Count Vectorizer 1000** | 73.943661971831 | 76.49122807017544 | 75.0 |
| **Count Vectorizer 1500** | 72.3404255319149 | 77.08333333333333 | 75.44483985765125 |
| **Count Vectorizer 2000** | 71.63120567375887 | 78.62068965517241 | 78.89273356401382 |
| **Count Vectorizer 2500** | 71.48014440433214 | 79.16666666666666 | 77.19298245614034 |
| **Count Vectorizer 3000** | 71.94244604316546 | 78.49829351535836 | 77.66323024054984 |
| **TFIDF Vectorizer 500** | 73.10344827586206 | 74.91638795986621 | 74.50980392156865 |
| **TFIDF Vectorizer 1000** | 71.32867132867133 | 76.35135135135135 | 75.4325259515571 |
| **TFIDF Vectorizer 1500** | 69.47368421052632 | 76.71232876712328 | 77.19298245614034 |
| **TFIDF Vectorizer 2000** | 69.31407942238268 | 77.81569965870307 | 79.31034482758622 |
| **TFIDF Vectorizer 2500** | 70.03610108303249 | 76.8166089965398 | 77.62237762237763 |
| **TFIDF Vectorizer 3000** | 70.28985507246377 | 77.81569965870307 | 80.13937282229965 |

**Results Analysis**

As seen in the tables above, the F-measure scores indicate that the best performance model occurs with a TFIDF Vectorizer feature selection technique, choosing the top 3000 features, using the Linear Support Vector Classification classifier. The results for both the 5-Fold Cross Validation experiments and the "held out data" demonstrate this outcome.

It is not surprising that the TFIDF Vectorizer feature selection technique performed the highest because it takes into account the highest term's weight, not just its frequency. In addition, since the Linear Support Vector Classification is not a probability classifier, it is extremely flexible and scales particularly well with sparse or dense input.

The next highest performing model is the Count Vectorizer feature selection technique, choosing the top 2500 or 3000 features, using the Logistic Regression classifier. It was interesting to find that with the 5-Fold Cross Validation experiments, the optimal top number of features was 3000, while the "held out data" experiment demonstrated that the optimal number of top features was 2500.

The difference in feature selection size likely occurs because the experiment with the "held out" data set contained more data, and thus, less features, meaning less noise (unimportant features).

**Future Improvements**

In the future, the models chosen should be evaluated on various different types of data sets to determine which ones truly perform better using different data. This might include a higher number of positive and negative sentiments, or reviews from different domains such as spam mail.

Further optimization could also be performed on the best models. This includes testing different parameters for each feature selection and classification technique, further research into other techniques and more time spent on analysis.

# README

## Student Information

Student Number: 0954174
Student Name: Miriam Snow
Student Email: msnow01@uoguelph.ca

## Organization of Files

Once the tar file is extracted you will find one directory titled "original" which contains two directories "negData" and "posData" which each contain 1000 files of negative and positive movie reviews, respectively.

You will also find a makefile, split.py, format2CSV.py, dataAnalysis.py and preProcess.py which are used for data analysis and pre processing before any testing can be done. The rest of the files contain implementation code for various feature selection and classifier method combinations as specified below.

Here is a list of codes for how each file is named:

FEATURE:
- CV = Count Vectorizer Feature Selection
- TFIDF = TFIDF Vectorizer Feature Selection

CLASSIFIER:
- MNB = Multinomial Naive Bayes Classifier
- LR = Logistic Regression Classifier
- LSVC = Linear Support Vector Classification Classifier

## Build and Test Instructions

1. In the main directory, type "make" into the command line to…
    a. Create two data analysis files for the negative and positive data collections
    b. Separate the entire data set into validation (15%) and K-Fold Cross Validation (85%) data sets
    c. Format the data into CSV files
    d. Preprocess the data by removing punctuation, performing case normalization and stemming words

2. Perform K-Fold Cross Validation
    a. Using the codes listed above, choose a combination to test.
    b. The files are named KFOLD_<FEATURE>_<CLASSIFIER>.py
    c. Indicate the number of features to be selected as the first command line argument
    d. Indicate the number of folds to be performed as the second command line argument

    For example, if you want to test 2000 features from the Count Vectorizer Feature Selection method with a Multinomial Naive Bayes Classifier and 10 folds, type the following into the command line:

    "python KFOLD_CV_MNB.py 2000 10"

    As another example, if you want to test 500 features from the TFIDF Vectorizer Feature Selection method with a Linear Support Vector Classification Classifier and 5 folds, type the following into the command line:

    "python KFOLD_TFIDF_LSVC.py 500 5"

3. Validate a feature selection and classifier combination (15/85 test/train split)
    a. Using the codes listed above, choose a combination to test.
    b. The files are named <FEATURE>_<CLASSIFIER>.py
    c. Indicate the number of features to be selected as the first command line argument

    For example, if you want to test 1500 features from the Count Vectorizer Feature Selection method with a Logistic Regression Classifier, type the following into the command line:

    "python CV_LR.py 1500"

    As another example, if you want to test 3000 features from the TFIDF Vectorizer Feature Selection method with a Multinomial Naive Bayes Classifier, type the following into the command line:

    "python TFIDF_MNB.py 3000"

4. Type "make clean" into the command line after testing to remove all pre processed data and CSV files

Works Cited

Brownlee, Jason. “A Gentle Introduction to k-Fold Cross-Validation.” *Machine Learning Mastery*, 8 Aug. 2019.

Carrasco, Oscar Contreras. “Support Vector Machines for Classification.” *Medium*, Towards Data Science, 22 Aug. 2019.

Hourrane, Oumaima. “Sentiment Analysis Using Python (Part I - Machine Learning Model Comparison).” *DataScienceToday*, 15 Sept. 2018.

Navlani, Avinash. “Understanding Logistic Regression in Python.” *DataCamp Community*, 7 Sept. 2018.

Shaikh, Raheel. “Cross Validation Explained: Evaluating Estimator Performance.” *Medium*, Towards Data Science, 26 Nov. 2018.

“Sklearn.feature_extraction.Text.CountVectorizer.” *Scikit Learn*, Scikit-Learn Developers, 2019.

“Sklearn.feature_extraction.Text.TfidfVectorizer.” *Scikit Learn*, Scikit-Learn Developers, 2019.

“Sklearn.linear_model.LogisticRegression.” *Scikit Learn*, Scikit-Learn Developers, 2019.

“Sklearn.model_selection.KFold.” *Scikit Learn*, Scikit-Learn Developers, 2019.

“Sklearn.naive_bayes.MultinomialNB.” *Scikit Learn*, Scikit-Learn Developers, 2019.

“Sklearn.svm.LinearSVC.” *Scikit Learn*, Scikit-Learn Developers, 2019.