# Movie Review Sentiment Classification

## Using Machine Learning Techniques

Dimitar Dimitrov – 0922289

University of Guelph

November 25, 2019

# Table of Contents

# Introduction

Since the rise of the internet and web 2.0 people around the world have generated and continue to generate an enormous amount of content online every day. Unfortunately, to absorb all the unique information on the web is very difficult for a human to do but it can be considerably easier for a machine. Having a machine interpret and summarize results for a particular topic is useful to a person who just wants to get the gist of a particular piece of content. Examples of this include assessing the quality of a product by searching online reviews and getting informed about a particular event through news articles. This report focuses on the problem of movie review sentiment classification (positive or negative). By classifying many reviews for a particular movie in this way, we can correctly summarize peoples' opinions on the movie. The end result is an informed reader who can use this information to decide whether or not to see the movie. For this task I used the movie review dataset previously released by Cornell University in 2004. These reviews are labeled and balanced with 1000 positive and 1000 negative reviews. I used a variety of feature sizes in combination with chi-square testing and POS tagging to determine which parts of speech were more valuable to us when classifying reviews by sentiment. I then performed supervised machine learning classification using naïve bayes multinomial, support vector machine, and k-nearest neighbours to determine the best classifier for this task. Overall the best performance in terms of f-measure was achieved by naïve bayes multinomial with a feature size of 3000 and accounting only for adjectives and adverbs. The worst classifier for this job was the distance-based k-nearest neighbors. One other classifier that performed almost as well as naïve bayes was support vector machine.

# Description of Techniques Used

For this comparative study I used two different feature selection techniques at different feature sizes then found the best performing machine learning classifier at each size for the given feature.

## Feature Selection

Feature selection is often useful for many text classification tasks. By identifying a subset of discriminative features, we can potentially improve classification performance and reduce computational cost in memory and running time (Song, 2019). One common feature selection technique is the chi-square ranking. Using the chi-square test we may determine whether the output variable in question (feature) is dependent on the input variable (label). If we get a low chi-square test statistic we may interpret that the two variables are independent in which case the feature may be irrelevant to the problem and discarded (Brownlee, 2019). In the area of sentiment analysis not all words are created equal. Different parts of speech are better able to express sentiment. Adjectives, adverbs, verbs, and nouns are all important for sentiment analysis but in varying degrees (Song, 2019). With this thought in mind the two feature selection methods chosen differ in the types of speech permitted for the corpus. To determine which words belong to each type we can use various third-party libraries to analyze and tag each token in the dataset. For the first feature selection technique we choose all nouns, verbs, adjectives, and adverbs to remain in the data set. For the second feature selection technique we select only adjectives and adverbs to see if these alone can perform better for movie review sentiment analysis.

## Machine Learning Classifiers

For this experiment I used one probabilistic classifier and two distance-based classifiers; naïve bayes multinomial, support vector machine, and k-nearest neighbors respectively. From the

Pang study (2002), we find that better performance can be achieved with unigrams so to train the machine learning models on the POS-filtered movie review data I used the bag of words feature framework and represented each document as a vector of term frequencies.

**Naïve Bayes Multinomial**

The first classical machine learning algorithm used was naïve bayes multinomial. Derived from the bayes theorem, with naïve bayes classifiers we assume that all features are independent of each other. We also include the term frequencies (n) in the estimates.

$$p(w_t|c_j) = \frac{1 + \sum_{i=1}^{|D|} n_{it} f(c_j, d_i)}{|V| + \sum_{s=1}^{|V|} \sum_{i=1}^{|D|} n_{is} f(c_j, d_i)}$$

*Figure 1. Formula for Naive Bayes Multinomial*

I chose naïve bayes for this task since despite its simplicity and its unrealistic independence assumption for features it has been shown to perform quite well for text classification tasks.

**Advantages**:

- NBM is easy to implement and fast to execute.
- Works for both binary and multi-class classifications
- Makes predictions based on probability statistics

**Disadvantages:**

- Assumes that features are all independent of each other (naïve).
- With sparse data, smoothing techniques are often needed.

**Support Vector Machine**

The idea behind the support vector machine learning classifier is that we take all the training data and separate it according to the class it is labeled with. Then with SVMs we can find vectors surrounding each class such that the margin between them is maximized. Once the

training is complete, we can use a decision function to classify a given document based on its position relative to the other (labelled) data points.

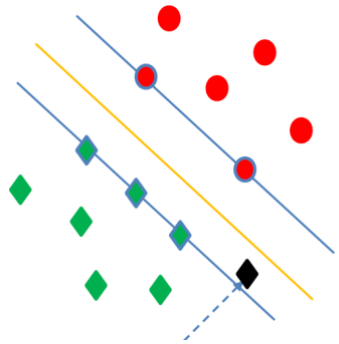

*Figure 2. Example of classification using SVM*

**Advantages**:

- Found to be the most accurate for text classification tasks (better than naïve bayes and decision trees).

**Disadvantages:**

- Optimizing hyper-parameters for performance can be a difficult task in itself.
- Can be slow in execution. Doesn't scale well as data size increases.

**K-Nearest Neighbours**

The second distance-based classifier I used is k-nearest neighbours. With this classifier for every new document that appears we simply make a prediction on its class based on its proximity to other documents. For text classification we can model this operation using an information retrieval mechanism. Given a new document we can search for other documents in the index using the new document as a query. When we find the search results ordered by relevancy we take the top K documents and the class with the majority representation in the K documents becomes the classification of the new document.

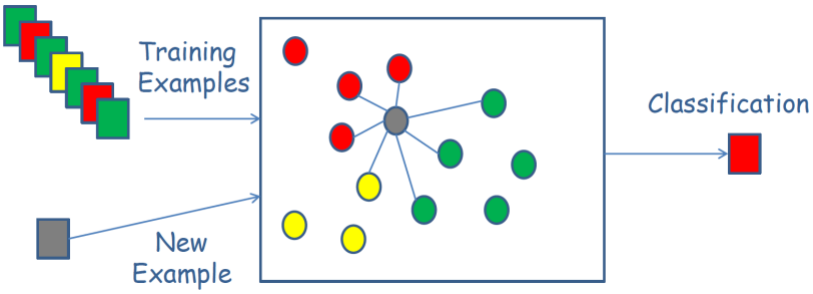


*Figure 3. Example of classification using K-Nearest Neighbors*

**Advantages**:

- Relatively easy to implement.
- Algorithm is versatile for many language modeling tasks.

**Disadvantages:**

- Doesn't seem to perform well for sentiment analysis.

# Implementation Details

## Phase 1: Feature Preprocessing

Each feature selection method requires that we preprocess the movie review data, filtering out all the types of words we don't want to keep based on their POS tags. I found the best way to do this was to implement a separate program that takes the original review data and writes the filtered data to new directories matching the original's structure.

To do this we develop a separate python script called *data_preprocessor.py*. Because it is a time-consuming process to write out all these new files to disk we only need to run this program once. Taking the root folder of the movie reviews as input we generate 2 different directories for data related to each feature selection method (for example, txt_sentoken_feature_1 and txt_sentoken_feature_2). We read the file data for each positive and negative review and write out a copy of this file keeping only the selected parts-of-speech for the given feature. To POS-tag each review we use a popular NLP library for python called nltk. After tagging the review data,

to determine whether or not to keep a token in the file we simply check if its corresponding tag belongs to a manually defined python set for adjectives, adverbs, verbs, and nouns (feature 1), and adjectives and adverbs (feature 2).

## Phase 2: Dataset Splitting

To begin the dataset splitting process we load all the pre-processed review data into memory used the *load_files* method defined in sci-kit. I used this method as it uses the sub-directory names (pos and neg) as data labels, which is convenient for us given how our data is organized. For each feature we split the data into training and validation partitions allocating 15% of the data for development/validation purposes. To do so we make use of the handy sci-kit method *train_test_split* and specify a stratified dataset in the arguments.

After the data is split we take the training data and its corresponding labels and split it into five folds for cross-validation. To do this we make use of the *KFold* module from sci-kit and prepare a list of python dictionaries with each index in the list corresponding to a particular fold. Because the fold is a dictionary we can store and access the training and testing data partitions separately.

## Phase 3: Additional Feature Selection

To complete feature selection I have a method for building pipelines utilizing sci-kit's *Pipeline* module. Each document is first transformed into a vector of counts representing the term frequencies for each word type in the review. After this we use the built-in estimator from sci-kit called *SelectKBest* to score vectors based on their chi-square test results and return the top K features (useful for specifying feature size). At the end of our pipeline is the machine learning classifier used to train the data. In this experiment we use the modules *MultinomialNB,*

*SDGClassifier, and KNeighborsClassifier* provided by sci-kit to perform the training and prediction.

### Phase 4: Cross-Validation

To optimize the model on the three factors of feature selection method, feature size, and machine learning classifier, we start by picking a feature selection method and a machine learning classifier. For this combination we then perform two five-fold cross-validation tests, one using the validation set (held-out set) and one using the test folds (2 x 3 = 6 combinations). The weighted and averaged f-measures (over all folds) are then stored for each feature size; 500, 1000, 1500, 2000, 2500, and 3000 (6 x 6 = 36 combinations). From this iteration we use the validation f-measures to determine the best performing feature size for this feature-classifier combination. Given this information we take the previously computed cross-validation score for the testing folds and use this as our performance for the "best model".

We perform the same tests for the second feature selection method and compare the results.

## Dataset

The dataset for this experiment consisted of 1000 positive-labeled movie reviews and 1000 negative-labeled reviews. Using the *DataAnalyzer.java* program we can find some useful statistics to better understand our data.

### Sentence-Based Statistics

- Minimum document length by number of sentences: 1
- Maximum document length by number of sentences: 112
- Average document length by number of sentences: 32

### Token-Based Statistics

- Minimum document length by number of tokens: 17
- Maximum document length by number of tokens: 2679
- Average document length by number of tokens: 747
- Average sentence length by number of tokens: 23

# Results

## Feature Selection Method 1: Adjectives, Adverbs, Verbs, and Nouns

### Naïve Bayes Multinomial

| Feature Size | Validation Set Score (Weighted F-Measure) |
|---|---|
| 500 | 0.786163 |
| 1000 | 0.780596 |
| 1500 | 0.774002 |
| 2000 | 0.777554 |
| 2500 | 0.772476 |
| 3000 | 0.780475 |

**Best feature size = 500, f-measure with test folds: 0.801266**

### Support Vector Machine

| Feature Size | Validation Set Score (Weighted F-Measure) |
|---|---|
| 500 | 0.734955 |
| 1000 | 0.760027 |
| 1500 | 0.763878 |
| 2000 | 0.773360 |
| 2500 | 0.794737 |
| 3000 | 0.803680 |

**Best feature size = 3000, f-measure with test folds: 0.819279**

### K-Nearest Neighbors (K = 5)

| Feature Size | Validation Set Score (Weighted F-Measure) |
|---|---|
| 500 | 0.554373 |
| 1000 | 0.559829 |
| 1500 | 0.512540 |
| 2000 | 0.514453 |
| 2500 | 0.507191 |

| | |
|---|---|
| 3000 | 0.481041 |

**Best feature size = 1000, f-measure with test folds: 0.574595**

## Feature Selection Method 2: Adjectives and Adverbs Only

### Naïve Bayes Multinomial

| Feature Size | Validation Set Score (Weighted F-Measure) |
|---|---|
| 500 | 0.831515 |
| 1000 | 0.834885 |
| 1500 | 0.835664 |
| 2000 | 0.845073 |
| 2500 | 0.840501 |
| 3000 | 0.845537 |

**Best feature size = 3000, f-measure with test folds: 0.822855**

### Support Vector Machine

| Feature Size | Validation Set Score (Weighted F-Measure) |
|---|---|
| 500 | 0.759916 |
| 1000 | 0.773977 |
| 1500 | 0.743591 |
| 2000 | 0.769640 |
| 2500 | 0.793241 |
| 3000 | 0.794917 |

**Best feature size = 3000, f-measure with test folds: 0.810757**

### K-Nearest Neighbors (K = 5)

| Feature Size | Validation Set Score (Weighted F-Measure) |
|---|---|
| 500 | 0.520709 |
| 1000 | 0.498757 |
| 1500 | 0.505203 |
| 2000 | 0.503232 |
| 2500 | 0.498592 |
| 3000 | 0.508447 |

**Best feature size = 500, f-measure with test folds: 0.477824**

# Analysis

## Machine Learning Classifier and Feature Size

### Naïve Bayes Multinomial

Given this particular run of the experiment we can't seem to find a specific one-size-fits-all feature size for the naïve bayes multinomial classifier. From the validation set we find 500 to be the best feature size for feature selection method 1 while the best feature size for feature selection method 2 was 3000. These values are both at the extremes of the range of feature sizes tested. Perhaps more testing needs to be done to confirm this reality.

### Support Vector Machine

The best feature size for support vector machine was consistent with the two different features selection methods at 3000. From this we can assume that for sentiment analysis support vector machine generally performs better with larger feature sizes.

### K-Nearest Neighbors

The best feature size for k-nearest neighbors when using five neighbors in the decision function lies somewhere in the interval of (500, 1000). However, from the results we can see that differences in f-measure scores are not very big and don't seem to degrade as the feature size increases. K-nearest neighbors seems to scale well for sentiment classification.

## Machine Learning Classifier and Feature Selection Technique

### Nouns, Verbs, Adjectives, and Adverbs

In terms of performance, the best results for the first feature selection method came from support vector machine (0.819) with naïve bayes multinomial close behind (0.801). K-Nearest neighbors performed the worst (0.575).

### Adjectives, and Adverbs

The best results for feature selection method 2 came from naïve bayes with a weighted f-measure of 0.823. Support vector machine followed closed behind with a score of 0.811. Once again k-nearest neighbors performed the worst (0.478).

### Overall

Overall, we can find that K-nearest neighbors has the worst performance out of all the classifiers for sentiment analysis given my configurations for the experiments (see code for details). Naïve bayes and support vector machine had similar overall performances between the two tests. Naïve bayes multinomial performed slightly better for the feature selection method using only adjectives and adverbs while SVM performed slightly better when accounting adjectives, adverbs, nouns, and verbs.

In the end by selecting only adjectives and adverbs for our features, we could deliver better performance overall (0.823) when compared with using adjectives, adverbs, verbs, and nouns (0.819). This may suggest that verbs and nouns are less effective in expressing sentiment.

## Conclusion

From the results above we find conclusions similar to those found by Pang (2002) that naïve bayes classifiers have performances similar to support vector machine in the domain of movie review sentiment analysis. In this study we find that K-nearest neighbors is the worse classifier out of the three for sentiment analysis, severely underperforming the others.

In the Pang study (2002) they found that unigram presence performed better overall when compared to unigram frequency. For another iteration of this experiment one may choose to account solely for presence instead. Instead of preprocessing the data into document vectors of

term frequencies we can simply replace the frequency with 1 if the word appears in the document and 0 if it does not.

Perhaps most useful for sentiment analysis would be to gain the ability to determine when the author is referencing the movie itself. While it is difficult to capture context surrounding a given word in a movie review (seen by the ineffectiveness of bigrams), the effect of the positioning of a word in a review should be explored further. Assuming there's a common structure for a movie review we may see that sentiment for a movie is expressed more frequently in certain parts of the document than others (for example the beginning or very end).

# Build and Installation Guide

## Compilation/Pre-running

Compile all java programs: **javac DataAnalyzer.java Document.java**

Download all python package dependencies: **pipenv install**

## Running all Programs

Data Analyzer: **java DataAnalyzer <positive_review_folder> <negative_review_folder>**

Data Preprocessor: **pipenv run python data_preprocessor.py <review_root_folder>**

Sentiment Analysis Test: **pipenv run python sentiment_analysis_test.py <review_folder_f1> <review_folder_f2>**

Note: Data analyzer program is optional. It is not involved in the testing process.

### Scenario 1: No Preprocessed Documents

1. Run the data preprocessor script, specifying the root folder for the movie reviews:
   **pipenv run python data_preprocessor.py txt_sentoken**
2. Run the sentiment analysis tester specifying the root folders for the first and second feature selection directories respectively.

**pipenv run python sentiment_analysis_test.py txt_sentoken_feature_1 txt_sentoken_feature_2**

**Scenario 2: Preprocessed Documents Ready**

1. Simply run the sentiment analysis tester specifying the root folders for the first and second feature selection directories respectively.

   **pipenv run python sentiment_analysis_test.py txt_sentoken_feature_1 txt_sentoken_feature_2**

# References

Brownlee, J. (2019, October 31). *A Gentle Introduction to the Chi-Squared Test for Machine Learning*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/chi-squared-test-for-machine-learning/

Song, F. (2019). *Distance-Based Classifiers.* Personal Collection of F. Song, University of Guelph, Guelph ON.

Song, F. (2019). *Probability-Based Classifiers.* Personal Collection of F. Song, University of Guelph, Guelph ON.

Song, F. (2019). *Text Classification.* Personal Collection of F. Song, University of Guelph, Guelph ON.