**Sentiment Analysis on Movie Reviews**

**Abstract**

There are many ways of implementing the use of sentiments often found in documents; one of which is the sentiments found on the product or service reviews. It is so important to be able to process and extract textual data from the documents. Therefore, we propose a system that is able to classify sentiments from review documents into two classes: positive sentiment and negative sentiment. We use the Logistic Regression method in this classification system that we build. Most of the time reviews on movies carry sentiment which indicates whether the review is positive or negative. The goal of this project is to predict the sentiments of reviews using basic algorithms. We choose a problem that was taken from the Kaggle competition called “The Bag of Words Meets the Bags of Popcorn”. The classifying accuracy yields an average of 87.5% from five times of accuracy measuring attempts using the aforementioned dataset.

**1. Introduction**

The Sentiment Analysis is a field that evaluates the emotions and feelings in the review texts. In order to calculate the sentiment score of the review, each piece of text can be examined separately or in combination with others. In this manner, after calculating the sentiment scores of all the pieces of text in the review some aggregation technique is used to calculate the overall sentiment of the review. One of the simplest ways of combining the total score of the review is summing all the scores of all the pieces in that review. Certainly, it is our choice to select how big those pieces will be. We can choose every word, n subsequent words, sentence, and/or the whole review to represent a feature. However, due to the fact that there is a little chance that there will be repeating reviews, it is more reasonable to have smaller pieces of text as representations of features. Therefore, we will apply the Logistic Regression Model, which will have every word to represent a feature and predict the sentiment.

**Bag-of-Words Model:**

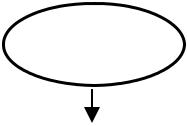
We cannot work with text directly when using machine learning algorithms. Instead, we need to convert the text to numbers. We may want to perform classification of documents, so each document is an “*input*” and a class label is the “*output*” for our predictive algorithm. Algorithms take vectors of numbers as input, therefore we need to convert documents to fixed-length vectors of numbers. A simple and effective model for thinking about text documents in machine learning is called the Bag-of-Words Model, or BoW. The model is simple in that it throws away all of the order information in the words and focuses on the occurrence of words in a document. This can be done by assigning each word a unique number. Then any document we see can be encoded as a fixed-length vector with the length of the vocabulary of known words. The value of each position in the vector could be filled with a count or frequency of each word in the encoded document.

This is the bag of words model, where we are only concerned with encoding schemes that represent what words are present or the degree to which they are present in encoded documents without any information about the order. There are many ways to extend this simple method, both by better clarifying what a “*word*” is and in defining what to encode about each word in the vector.

**1.1 Exploratory Analysis:**

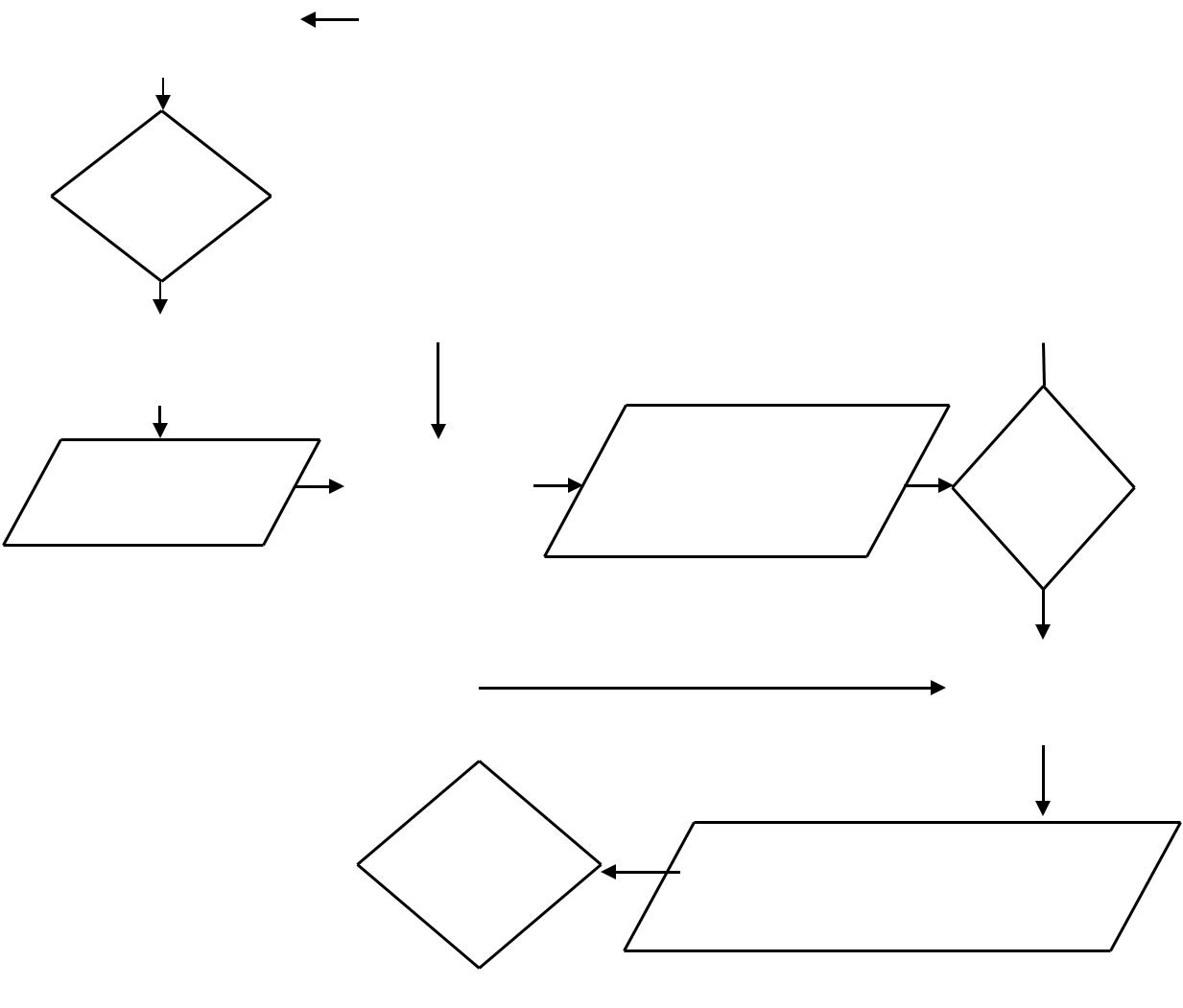
We are given a training and a test set both including 25, 000 user ids and movie reviews. In the training set, we are also given a sentiment of the movie which is 1 if it is positive and 0 otherwise. We are also given 50, 000 ids and movie reviews as unlabeled training set but we will not use that data. Due to a huge number of reviews in training set and, therefore, an enormous number of words we had to discard all non-necessary words by several steps:

* Lowercase of all words
* Delete all the punctuations
* Delete all the stopword such as “the”, “a”, “to”, etc.
* Stem the words( for example, fishing", "fished", and "fisher" to the root word, "fish")
* Discard all words that occur less than or equal to 15 times because there is little change they relate to the sentiment of the review
* Finally, collect all the left unique words and their frequency of occurrence



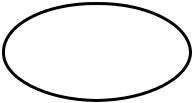
Start

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Add a review in | |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | the dataset | |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | Yes | | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  | Add more | |  |  |  |  |  |  |  |  |  |  |  |
|  | Review? |  |  |  |  |  |  |  |  |  |  |  |  |
|  | No | |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | Document | |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | Yes | |  |
|  | Extraction Process | |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  | Table Number of | Train | |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | System | | |  | Occurrences of Terms |  |
|  | Bag of Words | |  |  |  |  | the | |  |
|  |  |  |  |  | and Likelihood of Each |  |
|  |  |  |  | Training | | |  |  |
|  |  |  |  |  | system | |  |
|  |  |  |  |  |  |  | Term |  |
|  |  |  |  |  |  |  |  |  |  | No | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | Accuracy | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | Yes | |  | Calculation | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |



|  |  |  |
| --- | --- | --- |
| Calculate | The Results of The Sentiment |  |
| the accuracy |  |
| Analysis of Each Review in the |  |
| again? |  |
| Test Data and Confusion Matrix |  |
|  |  |

No



Finish

Overview of Movie Review Sentiment Analysis System

**2. Predictive Task**

After the data analysis that is done in the previous section, our goal is to make a prediction of the review sentiment based only on review text.

**(𝑟𝑒𝑣𝑖𝑒𝑤 𝑡𝑒𝑥𝑡 𝑓𝑒𝑎𝑡𝑢𝑟𝑒𝑠)− > 𝑠𝑒𝑛𝑡𝑖𝑚𝑒𝑛[0 𝑜𝑟 1]**

One of our variables will be the review text features themselves. We will estimate the sentiment score based on the processed text. We will also switch the frequency threshold that will be responsible for considering only those features whose positive/negative scores are higher than the threshold’s value. Then we will compute the score based on features. Finally, we will use the Logistic Regression Model to predict the sentiment. Because by default Logistic Regression uses all the features given and in our case those features are words, we will test how Logistic Regression model will work if the features are selected more carefully. By choosing only those words that appear the most in positive or the most in negative, it is natural to assume that our prediction accuracy will improve. We use CountVectorizer and Tfidf for tokenization.

**2.1 CountVectorizer:**

The CountVectorizer provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary.

You can use it as follows:

* Create an instance of the CountVectorizer class.
* Call the fit() function in order to learn a vocabulary from one or more documents.
* Call the transform() function on one or more documents as needed to encode each as a vector.

An encoded vector is returned with a length of the entire vocabulary and an integer count for the number of times each word appeared in the document. Because these vectors will contain a lot of zeros, we call them sparse. Python provides an efficient way of handling sparse vectors in the **scipy.sparse** package. The vectors returned from a call to transform() will be sparse vectors, and you can transform them back to **numpy** arrays to look and better understand what is going on by calling the array() function.

**2.2 Word Frequencies with TfidfVectorizer:**

Word counts are a good starting point but are very basic. One issue with simple counts is that some words like “the” will appear many times and their large counts will not be very meaningful in the encoded vectors. An alternative is to calculate word frequencies, and by far the most popular method is called TF-IDF. This is an acronym that stands for “Term Frequency-Inverse Document” Frequency which are the components of the resulting scores assigned to each word.

**Term Frequency**: This summarizes how often a given word appears in a document.

**Inverse Document Frequency**: This downscales words that appear a lot across documents.

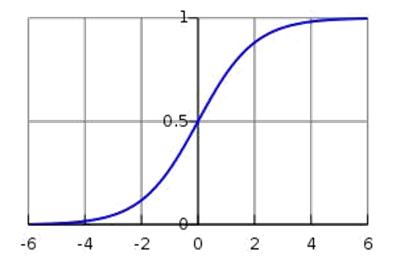
Without going into the math, TF-IDF is word frequency scores that try to highlight words that are more interesting, e.g. frequent in a document but not across documents. The TfidfVectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents. Alternately, if you already have a learned CountVectorizer, you can use it with a TfidfTransformer to just calculate the inverse document frequencies and start encoding documents.

The same create, fit, and transform process is used as with the CountVectorizer.

**2.3 Logistic Regression:**

Despite its name, it is a linear model for classification rather than regression. It is also known in the literature as logit regression, log-linear classifier and maximum-entropy classification (MaxEnt). In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function. A logistic function or logistic curve is a common S shape (sigmoid curve), with the equation:





Performance has been evaluated using holdout approach (80% -training, 20%- testing)

**3. Analysis**

We generate a Confusion matrix for summarizing the performance of a classification algorithm. Calculating a confusion matrix can give you a better idea of what your classification model is getting right and what types of errors it is making.

Classification accuracy is the ratio of correct predictions to total predictions made.

**classification accuracy = correct predictions / total predictions**

It is often presented as a percentage by multiplying the result by 100.

**classification accuracy = correct predictions / total predictions \* 100**

Below is the process for calculating a confusion Matrix.

* You need a test dataset or a validation dataset with expected outcome values.
* Make a prediction for each row in your test dataset.
* From the expected outcomes and predictions count:
* The number of correct predictions for each class.
* The number of incorrect predictions for each class, organized by the class that was predicted.

These numbers are then organized into a table or a matrix as follows:

**Expected down the side:** Each row of the matrix corresponds to a predicted class.

**Predicted across the top:** Each column of the matrix corresponds to an actual class.

The counts of correct and incorrect classification are then filled into the table.

This gives us:

“true positive” for correctly predicted event values.

“false positive” for incorrectly predicted event values.

“true negative” for correctly predicted no-event values.

“false negative” for incorrectly predicted no-event values.

We can summarize this in the confusion matrix as follows:

**event no-event**

**event** true positive false positive

**no-event** false negative true negative

**Confusion Matrix for this project :**

**4. Conclusions**

In this project, we use basic machine learning techniques and explore how useful they can be in predicting the sentiment of movie reviews. Using simple approaches to train our model we can make a quite accurate prediction of the text’s sentiment. The Logistic Regression Model performed relatively well on our data set. By choosing what features our model must consider we can increase the accuracy of our results. The basic methods such as Logistic Regression Model consider every feature to have the same weight, however, as we have seen in this project, to have more accurate results they all must be treated differently.

**5. References**

1. [**https://www.analyticsvidhya.com/blog/2015/11/beginners-guide-on-logistic-regression-in-r/**](https://www.analyticsvidhya.com/blog/2015/11/beginners-guide-on-logistic-regression-in-r/)
2. [**https://machinelearningmastery.com**](https://machinelearningmastery.com)
3. [**http://scikit-learn.org/stable/modules/feature\_extraction.html#the-bag-of-words-representation**](http://scikit-learn.org/stable/modules/feature_extraction.html#the-bag-of-words-representation)