# Sentiment Analysis

## MACHINE LEARNING FOR NATURAL LANGUAGE PROCESSING

Vicky Mohammad
0895381
University of Guelph
mohammav@uoguelph.ca

# Introduction

**Abstract**

The underlying process of classifying through the overall sentiment can be difficult due to its multiple variables and its complex process of contextualizing and understanding of language documents. Finding the correct classifiers with relative of different feature selection, feature size, cross validation and ways of preprocessing data can drastically change accuracy and efficiency of the model. We will be using different classifiers and combination to establish an efficient model, best fitted for determining a positive and negative data.

**Problems and major applications**

The biggest obstacle in sentiment analysis is examining the context of the document data. For instance, multiple complications such as sarcasm, navel gazing, relative sentiment, conditional sentiment, ambiguous negative words, contain biases and multiple perception to understand the sentiment of the data. For example, obstacle such as sarcasm is one of the most difficult sentiments interpretation of whether it is a positive or negative response due to its inverse context meaning such as “it was awesome for the week that it worked.”

**Major techniques and data sets**

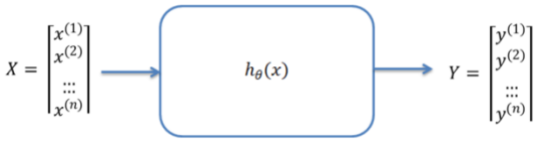
The data sets we will be using are 1000 positive and 1000 negative document of movie reviews from IMDb. We will be using three different classifier, logistic regression, random forest classifier, and

linear support vector classifier to train the data sets. We will be using different combination of feature size from 500 to 3000 in an increment of 500 sizes for each test with 5 and 10 stratified k-fold cross validation to make sure that the model is not under-fitting or over-fitting when it is creating an assumption.

# Technique Descriptions

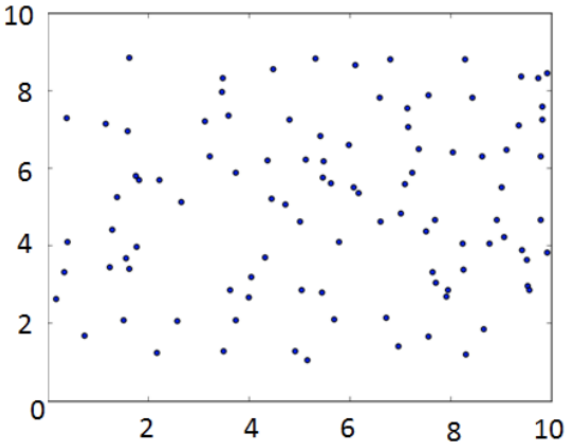
**Logistic Regression**

Regression analysis is a statistical model that in basic form use a logistic function to model a binary dependent variable. When there are one or more than one independent variable exists, it can be used to determine output or result from the multiple variables in the binary form of 0 or 1. The goal is to find the best a function which best correlates with the dataset. For instance, if we have a data sets containing *n* number of datapoints where $X = (x_1, x_2, x_3, \ldots x_n)$, in each of the inputs there exist a correspond output of *y-value,* the independent variables are *x-datapoints* results, and the dependent variable is *y*. Thus, *y* depends on *x* value. The objective is the to find the best regression function of $f(x)$ that best correlates $X$ with $Y$ called hypothesis function:

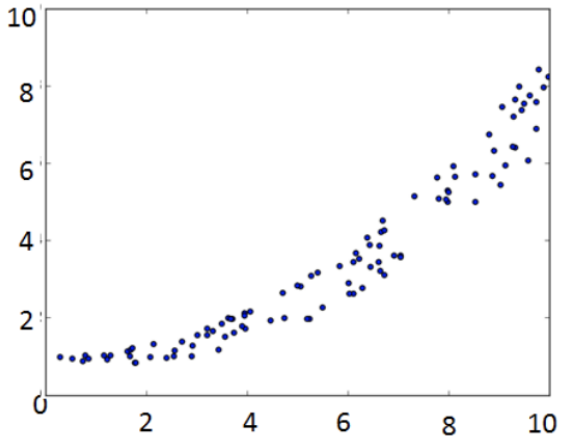



**(Figure 1)**

Furthermore, if we are trying to predict model for the success of a positive feedbacks in reviews. We would have *Y* datasets containing final feedback rating of *n* students and the independent value would be the *X* datasets:




**(Figure 2)**



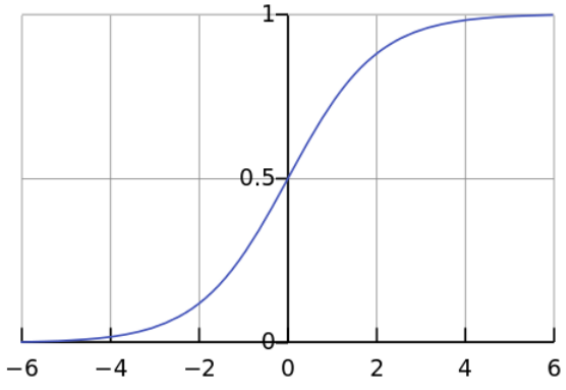

**(Figure 3)**

The *figure 2* shows inconsistent distribution, which shows no correlation between *X* and *Y*. The *figure 3* is the ideal distribution where there is a strong correlation occurred, this is where we want to see.

A logistic regression is a regression but with the motive of classifications where you must iteratively apply gradient descent of the cost function to approximate the value of the parameter. Cost function is given by the squared error of difference between the y and *hypothesis function*, with the form also known as logistic function:

$$h_\theta = \frac{1}{1+\exp(-z)} = \frac{1}{1+\exp(-\theta x)}.$$

With the function you can bound the y-direction by 1 and 0. It will strict the value to prevent beyond or below of 1 and 0. For example if *z* is negative value the function will go towards reaching 0 but will not pass below it, same goes with positive value reaching towards 1 as shown *figure 4* below, however, we have to create a decision boundary, which separate the positive and negative class:




**(Figure 4)**

The advantage of logistic regression is that it is very efficient and does not requires large computation. Hence, it is widely used technique, where the method can be interpreted easily, easy to regularize, does not need input features to scale, no tuning required and decent predication with probabilities.

Due to its simplicity, where it is easy and quick to implement, the disadvantage is that it is hard to solve a non-linear problem. It is also not the most powerful algorithm, there are other more complex algorithm that can easily outperformed it. Logistical regression is known with weakness of being overfitting.

**Multinomial Naïve Bayes**

Multinomial Naïve Bayes is a baseline classification solution for most cases for sentiment analysis. The methods of Multinomial Naïve Bayes are based on a Naïve Bayes technique by finding probability of classes through a joint probability of the word terms and its classes. Given the dependent vector $C_k = (x_1, x_2, x_3, \ldots x_n)$ the theorem stated with mathematical relation:

$$P(C_k \mid x_1, \ldots, x_n) = \frac{P(C_k)P(x_1, \ldots, x_n \mid C_k)}{P(x_1, \ldots, x_n)}$$

It is assumed that given class $C_k$, each vector feature of $x_1$ is conditionally independent of each and every other feature.

$$P(C_k \mid x_1, \ldots, x_n) = \frac{P(C_k)\prod_{i=1}^{n} P(x_i \mid C_k)}{P(x_1, \ldots, x_n)}$$

However, to avoid underflow, we can use the sum of logs:

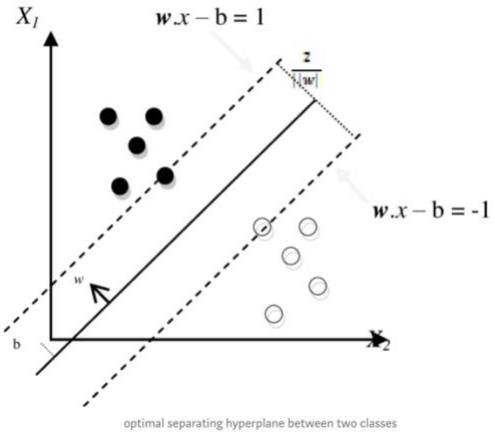
$$\hat{y} = \arg\max_k (\ln P(C_k) + \sum_{i=1}^{n} \ln P(x_i \mid C_k))$$

Although it is not a simple approach, the advantage of using Naïve Bayes is that it is fast method for prediction, it has low cost of computation, perform wells in discrete response variable in contrast to the continuous variable. Can be used for multi-class perditions, and lastly, in some cases if the independents hold the assumptions in most cases it performs better in comparison to other baseline classifier like logistic regression.

However, in most cases it is very hard that the model will have a set of entirely independent predictors, hence it can perform very poorly. Moreover, if there is no training of the correlated pair of class can hinder a great low probability, making it hard to construct proper predictions, which is known problem called the *zero probability/frequency problem*.

**Linear support vector classifier**

Linear support vector classifier, also known as SVM is a classification solution of a machine learning algorithm that is capable of solving regression or classification problems. SVM performs classifications by finding the hyper-plane where it can distinguish classes in *n-dimensional* space. The classifier is only trained using the support vectors as *figure 5* shown:




**(Figure 5)**

Like any other classifying methods, support vector machine has advantages and disadvantages. One of the advantages of support vector machine is that the methods is very effective in the higher dimension. It is very effective when the number of features is more than the training example. The classifier is also best when the classes are separable. Another pro is that, it is suited for extreme case binary classification. Moreover, the hyperplane is only affected by only the support vectors, hence outliers has minimal impact on the classifier, hence it does pretty good job classifying models that may contain unwanted outliers.

Though there are pros in support vector machine classifier, there is always a contrast when dealing with how to train and best fit the model. One of the disadvantages of using vector machine is that it the datasets are large it will need a large amount of time to process the data. It also does not perform well in the cases of when classes are overlapped. Selecting appropriate hyperparameter for the support vector machine will allow enough general performance and can be tricky to find the appropriate kernel functions.

# Implementation Highlight

**Data structure and models**

There are three main files in the project. In the *sk_learn.py* the *SkLearn* class is created for the process of training and validating as well as prototyping and testing different parameters and variables to find the best fit and accuracy of the models. The other class in the *sk_learn.py* contains *Classifiers* class used to test different classifier methods for the project. The *data_analysis.py* file contains *DataAnalysis* class which processes and prints all the data analysis result on the command line. During the process, of the data analysis, the class fist preprocesses the data to remove the stop words and stemmed the data to prevent any biases on the token and sentence analysis. This gives us an idea of what it is being processed before the training happens. Moreover, it will also prevent biases on the terms for the training model because each token will be stemmed to the root word.

**The Implementation of the natural language processing.**

After the preprocessing of the raw data that was obtained. The preprocessed data was used for the process of creating the ideal model for the sentiment analysis. The first step is to create a 15% validation set and

85% training sets from the preprocess data acquired. The 85% of the training sets will be used to train our models while the other 15% will be used for validation. A first, models were created to test to determine which potential machine learning classifiers might be good for the sentiment analysis such as:

- *Multi-layer perception*
- *Multinomial Naïve Bayes*
- *K-Nearest neighbors*
- *Random forest classifier*
- *Logistic regression*
- *Linear support vector classifier*

Multiple classifiers were tried, and three potential classifiers was selected through the analysis. The three selected classifiers are *logistic regression, linear support vector classifier* and the *multinomial naïve Bayes.* These three were picked because of the potential capability of what we can do when we optimize and tuned them due to having high accuracy scores. However, we cannot determine whether these scores are valid due to the known underfitting and overfitting problem. Hence, we process with a cross validation method later on. After the selection, two different selection were used; *TFxIDF* and *trigram models* vectorizer.

**Cross validation process**

Later, the process with the combination of three machine learning classifiers, two feature selection and size different sizes. Providing the 36 results. Through the process we use *stratified cross validation* process so that to ensure that each model was not underfitting or overfitting through 5 and 10 k-fold process. Stratified k-fold were used because stratification is generally the optimal scheme in term of biases and variance in comparison to the traditional k-fold. It is sought to ensure that each fold is representative of all strata data. Which ensures that each class is equally presented throughout each test fold. Once it is all processed, we average them then used the 15% validation sets to confirm that the model is acceptable.

# Data results and analysis

**Token and sentence data analysis**

| **Positive data** | **Sentences** | **Tokens** |
|---|---|---|
| Min sentences: | 5 | 114 |
| Max sentences: | 112 | 2303 |
| Avg sentences: | 32.937 | 668.087 |
| Total Sentences: | 32937 | 668087 |

| **Negative data** | **Sentences** | **Tokens** |
|---|---|---|
| Min sentences: | 1 | 16 |
| Max sentences: | 112 | 1799 |
| Avg sentences: | 31.783 | 596.265 |
| Total Sentences: | 31783 | 596265 |

When we preprocess the data, the result shown that after the preprocessing, the average, maximum and minimum sentence are the same as the positive and negative data. However, the positive average tokens were a bit more than the negative data by an average of 72 tokens *(figure 8).* This alludes that we have a small skewed uneven

distribution of more positive term frequency data then the negative one which may the training process. But since, it is very small it may not be as significant.

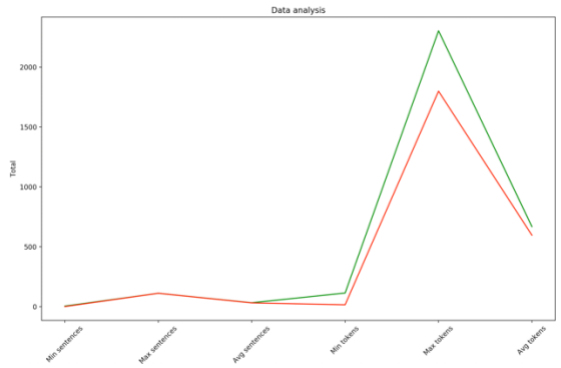



**Figure 8**

**Final model analysis**

After finding the best model. We determine that *linear support vector classifier* is our final model. From the combination, we found out that the best result are the *linear support vector classifier* and the *logistic regression* using the TFxIDF with the feature size of 2000 and stratified 10 k-folds cross validation result both with an accuracy around 85%:

Final accuracy score: 0.8534
Final f-measure: 0.8540

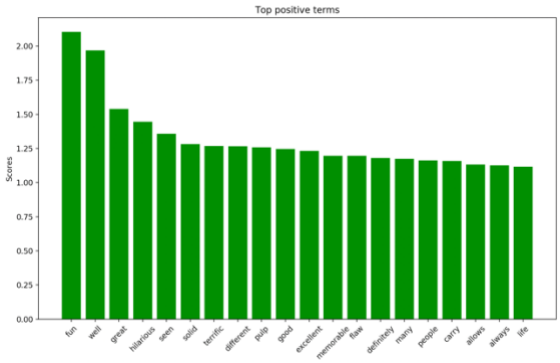



**Figure 6 LinearSVM**

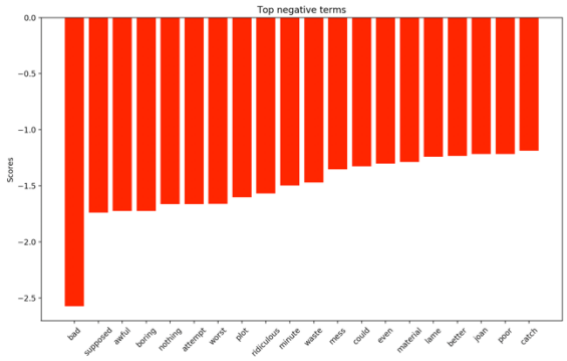



**Figure 7 LinearSVM**

As the feature size increase the accuracy also increase, however after certain size, the accuracy starts to decrease in a skewed parabolic form. After we have found the best model, we listed the top positive terms *(Figure 6)* and negative terms *(Figure 7)* to be able to see the top 20 scores through our perspective. Through our perspective we can see that the word “fun”, “well”, “terrific”, excellence and “good” shows a positive term throughout the final model. However, there are terms that are collected into the top 20 positive scores that may not be as positive term as it seems, such as the word “many” which can still be used as a negative connotation. Similar to negative terms in *figure 6*, we can see that there exist a small number of ambiguous words that can still be used. For example, the negative word “Ridiculous” can still be used as a positive connotation such as “That was ridiculously awesome”.

# Conclusions and Future work

The final result of the model was a success, although it is not the best model, there are parts where we can still improve upon. For example, when we examine the data, some words can still be improved such a better preprocessed data. Due to the sentiment complexity, it hard to detect convoluted problems mentioned in previously in the paper such as the such as detecting sarcasm where double meaning occurred, or the meaning has been inversed previously exampled. Nevertheless, the model is still capable of determine whether the review is a positive or negative sentiment with an 85% accuracy.

# Build and installation guide

**Specification and requirement**

The project was coded using *python* 3.7 and *Sci-kit learn* version 0.21 on Mac, Catalina. A *"makefile"* in the main project directory is already provided which contains all commands for the running, build, and the required installation for the dependencies.

First thing is to make sure that your computer has the capability of calling the *"make"* command on terminal. If not, you can install *"make"* using package manager like *"brew"* for mac or *"sudo-apt"* for Linux.

**Installing dependencies**

Once you have the correct specification and is capable of calling *"make"* on your terminal. On the main project directory, type *"make dependencies"* to install all required dependencies for this project.

**Running the program**

In the main project directory, simply type *'make'* to run the program, this will run the data analysis and training process for the selected models.

# References


https://www.adweek.com/digital/37705-sentiment-analysis/

https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6

https://datascience.stackexchange.com/questions/52632/cross-validation-vs-train-validate-test

https://scikit-learn.org/stable/tutorial/statistical_inference/model_selection.html

https://scikit-learn.org/stable/modules/cross_validation.html