KHULNA UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Project Name: Person Evaluation with Social Mining

Submitted to: Prof. Dr. K. M. Azharul Hasan

Submitted by:

Ishrak Islam Zarif Roll: 1307025

Mahmudul Hasan Roll: 1307049

Introduction: Our Software will do sentiment analysis on Gmail and classify the Gmail if the Gmail contains Positive text or Negative text. Our System can be divided into three major phases.

1. Data Preprocessing and making training and test sets
2. Calculating the Predictive Model
3. Visualization of Result

Data Preprocessing and Making Training and Test Sets: First, to train the classifier we need data. To gain the dataset we used python's nltk library's movie review corpus. Following are snapshots of positive and negative movie texts.

Positive:

```
In [9]: movie_reviews.sents(categories='pos')

Out[9]: [['films', 'adapted', 'from', 'comic', 'books', 'have', 'had', 'plenty', 'of', 'success', ',', 'whether', 'they',
         "'", 're', 'about', 'superheroes', '(', 'batman', ',', 'superman', ',', 'spawn', ')', ',', 'or', 'geared', 'towar
        d', 'kids', '(', 'casper', ')', 'or', 'the', 'arthouse', 'crowd', '(', 'ghost', 'world', ')', ',', 'but', 'there',
         "'", 's', 'never', 'really', 'been', 'a', 'comic', 'book', 'like', 'from', 'hell', 'before', '.'], ['for', 'starte
        rs', ',', 'it', 'was', 'created', 'by', 'alan', 'moore', '(', 'and', 'eddie', 'campbell', ')', ',', 'who', 'brough
        t', 'the', 'medium', 'to', 'a', 'whole', 'new', 'level', 'in', 'the', 'mid', "'", '80s', 'with', 'a', '12', '-', 'p
        art', 'series', 'called', 'the', 'watchmen', '.'], ...]
```

Negative:

```
In [14]: movie_reviews.sents(categories='neg')

Out[14]: [['plot', ':', 'two', 'teen', 'couples', 'go', 'to', 'a', 'church', 'party', ',', 'drink', 'and', 'then', 'drive',
          '.'], ['they', 'get', 'into', 'an', 'accident', '.'], ...]
```

Here to read these data we used categorized corpus reader. By using categorized corpus reader, we can read updated data and also other labeled data too.

Next Step is finding out the high info words from these corpus of text. A high

information word is a word that is strongly biased towards a single classification label. The low information words are words that are common to all labels. It may be counter-intuitive, but eliminating these words from the training data can actually improve accuracy, precision, and recall of our model. To calculate the high information words first we need to know how often each word occurs for each label. We have calculated score of a word for each label using Bigram association measures phi square function which takes following four parameters:

- n_ii: Frequency of the word for the label
- n_ix: Total Frequency of the word across all labels
- n_xi: Total Frequency of all words that occurred for the label
- n_xx: Total Frequency for all words in all labels

The correlation between the labels and each words are measured with Pearson's Phi Coefficient which is described below:

In statistics, the **phi coefficient** (also referred to as the "**mean square contingency coefficient**" and denoted by φ (or r_φ)) is a measure of association for two binary variables. Introduced by Karl Pearson, this measure is similar to the Pearson correlation coefficient in its interpretation. In fact, a Pearson correlation coefficient estimated for two binary variables will return the phi coefficient.

$$\phi^2 = \frac{\chi^2}{n}$$

where *n* is the total number of observations. Two binary variables are considered positively associated if most of the data falls along the diagonal cells. In contrast, two

binary variables are considered negatively associated if most of the data falls off the diagonal. If we have a 2*2 table for two random variables *x* and *y*

	$y = 1$	$y = 0$	total
$x = 1$	n_{11}	n_{10}	$n_{1\bullet}$
$x = 0$	n_{01}	n_{00}	$n_{0\bullet}$
total	$n_{\bullet 1}$	$n_{\bullet 0}$	n

where n_{11}, n_{10}, n_{01}, n_{00}, are non-negative counts of number of observations that sum to n, the total number of observations. The phi coefficient that describes the association of x and y is

$$\phi = \frac{n_{11}n_{00} - n_{10}n_{01}}{\sqrt{n_{1\bullet}n_{0\bullet}n_{\bullet 0}n_{\bullet 1}}}$$

Once we have the scores from phi square for each word in each label, we can filter out all words whose score is below the min_score threshold. We keep the words that meet or exceed the threshold and return all high scoring words in each label.

After getting the high info words and corpus we have to process the features to feed our classifiers. For these we need two models

1. Bag of words: It takes tokenized words as parameters and returns tuple containing (word, True). Ex:

```
In [4]: bag_of_words(['the', 'quick', 'brown', 'fox'])
Out[4]: {'brown': True, 'fox': True, 'quick': True, 'the': True}
```

2. Bag of words in set: it takes tokenized words and high info words as parameters and return tuple containing the (common words, tuple). Ex:

```
In [8]: bag_of_words_in_set(['the','quick','brown','fox'],['fox'])
Out[8]: {'fox': True}
```

After finding the features we have to make training and test sets from it. We have used 25% of our data as test set and 75% of our data for training set.

Calculating the predictive model: We have used three machine learning classifier models to predict the positive and negative Gmail. So, these classifiers work as binary classifiers but with the help of one vs rest method these classifiers can be extended to multiple label classifiers. The three classifiers are following:

1. Naive Bayes
2. Support Vector Machine
3. Decision Tree

1. Naive Bayes: Naive Bayes classifier is one of the simplest and popular probabilistic machine learning techniques. Naive Bayes classifier works based on Bayesian Theorem.

According to Bayes theorem,

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

To use this theorem in machine learning, let's define a classification problem: Given, a data point d, and a set of classes C = c 1, c 2, c 3, ..., c n, the likelihood of d being belongs to the any of the classes is P(C|d).

To expand this using Bayes theorem, we have:

$$P(C|d) = \frac{P(d|C)P(C)}{P(d)}$$

P(d) is same for all of the classes. Therefore, we can omit the this and we have following equation:

$$P(C|d) = P(d|C)P(C)$$

To decide on appropriate class, we have to take class for which P(C|d) is maximum. So the problem can be reduced to the following form:

$$argmax_C P(C|d)$$
$$= argmax_C P(d|C)P(C)$$

Let's split the document in its constituent tokens such as

$d = w_1 w_2 w_3 w_4 w_5 .. w_n$

$$= argmax_C P(d = w_1 w_2 w_3 w_4 w_5 .. w_n | C) P(C)$$

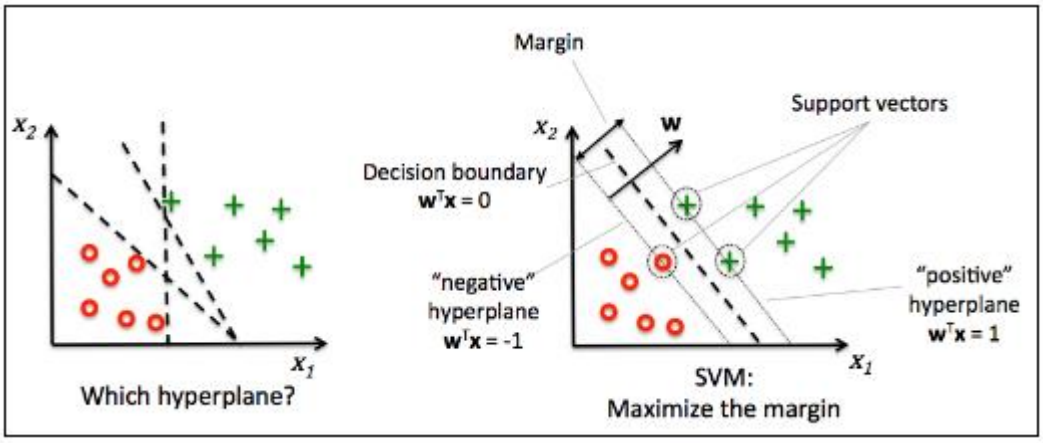
Assuming terms are independent, we have:

$$= argmax_C P(C) P(w_1|C) P(w_2|C) P(w_3|C) .. P(w_n|C)$$
$$= argmax_C P(C) \prod_{w_i \in V} P(w_i|C) \text{ , V is the Vocabulary}$$
$$= argmax_C \log P(C) + \sum_{w_i \in V} \log P(w_i|C) \text{ , V is the Vocabulary}$$

We can compute the class probability using following formula:

$$P(C) = \frac{N_c}{N}$$
$$P(w_i|C) = \frac{count(w_i, c) + 1}{V + \sum_{w_i \in V} count(w_i, C)}$$

2. Support Vector Machine: In SVMs, our optimization objective is to maximize the margin. The margin is defined as the distance between the separating hyperplane (decision boundary) and the training samples that are closest to this hyperplane, which are the so-called support vectors. This is illustrated in the following figure:



The rationale behind having decision boundaries with large margins is that they tend to have a lower generalization error whereas models with small margins are more prone to overfitting. To get an intuition for the margin maximization, let's take a closer look at those positive and negative hyperplanes that are parallel to the decision boundary, which can be expressed as follows:

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}_{pos} = 1 \qquad (1)$$

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}_{neg} = -1 \qquad (2)$$

If we subtract those two linear equations (1) and (2) from each other, we get:

$$\Rightarrow \boldsymbol{w}^T \left(\boldsymbol{x}_{pos} - \boldsymbol{x}_{neg} \right) = 2$$

We can normalize this by the length of the vector w, which is defined as follows:

$$\|\boldsymbol{w}\| = \sqrt{\sum_{j=1}^{m} w_j^2}$$

So we arrive at the following equation:

$$\frac{\boldsymbol{w}^T \left(\boldsymbol{x}_{pos} - \boldsymbol{x}_{neg} \right)}{\|\boldsymbol{w}\|} = \frac{2}{\|\boldsymbol{w}\|}$$

The left side of the preceding equation can then be interpreted as the distance between the positive and negative hyperplane, which is the so-called margin that we want to maximize.

Now the objective function of the SVM becomes the maximization of this margin $\frac{2}{\|\boldsymbol{w}\|}$ by maximizing w under the constraint that the samples are classified correctly, which can be written as follows:

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} \geq 1 \; if \; y^{(i)} = 1$$

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} < -1 \; if \; y^{(i)} = -1$$

These two equations basically say that all negative samples should fall on one side of the negative hyperplane, whereas all the positive samples should fall behind the positive hyperplane. This can also be written more compactly as follows:

$$y^{(i)}\left(w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)}\right) \geq 1 \ \forall_i$$

Now, let's briefly mention the slack variable ξ. The motivation for introducing the slack variable ξ was that the linear constraints need to be relaxed for nonlinearly separable data to allow convergence of the optimization in the presence of misclassifications under the appropriate cost penalization.

The positive-values slack variable is simply added to the linear constraints:

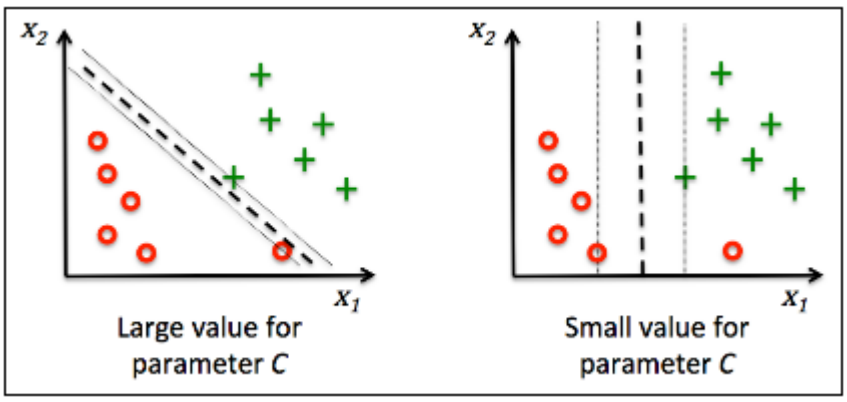
$$\boldsymbol{w}^T \boldsymbol{x}^{(i)} \geq 1 \ \textit{if} \ \ y^{(i)} = 1 - \xi^{(i)}$$

$$\boldsymbol{w}^T \boldsymbol{x}^{(i)} < -1 \ \textit{if} \ \ y^{(i)} = 1 + \xi^{(i)}$$

So the new objective to be minimized (subject to the preceding constraints) becomes:

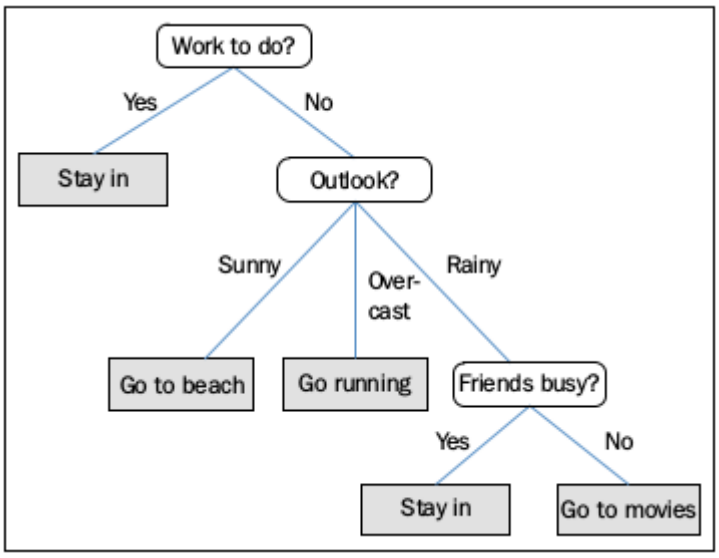
$$\frac{1}{2}\|\boldsymbol{w}\|^2 + C\left(\sum_i \xi^{(i)}\right)$$

Using the variable C, we can then control the penalty for misclassification. Large values of C correspond to large error penalties whereas we are less strict about misclassification errors if we choose smaller values for C. We can then we use the parameter C to control the width of the margin and therefore tune the bias-variance trade-off as illustrated in the following figure:



3. Decision Tree: Decision tree classifiers are attractive models if we care about interpretability. Like the name decision tree suggests, we can think of this model as breaking down our data by making decisions based on asking a series of questions.

Let's consider the following example where we use a decision tree to decide upon an activity on a particular day:



Based on the features in our training set, the decision tree model learns a series of questions to infer the class labels of the samples. Although the preceding figure illustrated the concept of a decision tree based on categorical variables, the same concept applies if our features. This also works if our features are real numbers like in the Iris dataset. For example, we could simply define a cut-off value along the sepal width feature axis and ask a binary question "sepal width ≥ 2.8 cm?"

Using the decision algorithm, we start at the tree root and split the data on the feature that results in the largest information gain (IG), which will be explained in more detail in the following section. In an iterative process, we can then repeat this splitting procedure at each child node until the leaves are pure. This means that the samples at each node all belong to the same class. In practice, this can result in a very

deep tree with many nodes, which can easily lead to overfitting. Thus, we typically want to prune the tree by setting a limit for the maximal depth of the tree.

In order to split the nodes at the most informative features, we need to define an objective function that we want to optimize via the tree learning algorithm. Here, our objective function is to maximize the information gain at each split, which we define as follows:

$$IG(D_p, f) = I(D_p) - \sum_{j=1}^{m} \frac{N_j}{N_p} I(D_j)$$

Here, f is the feature to perform the split, D p and D j are the dataset of the parent and jth child node, I is our impurity measure, N p is the total number of samples at the parent node, and N j is the number of samples in the jth child node. As we can see, the information gain is simply the difference between the impurity of the parent node and the sum of the child node impurities—the lower the impurity of the child nodes, the larger the information gain.

However, for simplicity and to reduce the combinatorial search space, most libraries (including scikit-learn) implement binary decision trees. This means that each parent node is split into two child nodes, D left and D right:

$$IG(D_p, a) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$$

Now, the three impurity measures or splitting criteria that are commonly used in binary decision trees are Gini index (I G), entropy (I H), and the classification error (I E). Let's start with the definition of entropy for all non-empty classes (p (i | t) ≠ 0):

$$I_H(t) = -\sum_{i=1}^{c} p(i|t) \log_2 p(i|t)$$

Here, p (i | t) is the proportion of the samples that belongs to class c for a particular node t. The entropy is therefore 0 if all samples at a node belong to the same class, and the entropy is maximal if we have a uniform class distribution. For example, in

a binary class setting, the entropy is 0 if p (i = 1| t) = 1 or p (i = 0 | t) = 0. If the classes are distributed uniformly with p (i = 1| t) = 0.5 and p (i = 0 | t) = 0.5, the entropy is 1. Therefore, we can say that the entropy criterion attempts to maximize the mutual information in the tree.

Intuitively, the Gini index can be understood as a criterion to minimize the probability of misclassification:

$$I_G(t) = \sum_{i=1}^{c} p(i|t)\left(-p(i|t)\right) = 1 - \sum_{i=1}^{c} p(i|t)^2$$

Similar to entropy, the Gini index is maximal if the classes are perfectly mixed, for example, in a binary class setting (c = 2):

$$1 - \sum_{i=1}^{c} 0.5^2 = 0.5$$

However, in practice both the Gini index and entropy typically yield very similar results and it is often not worth spending much time on evaluating trees using different impurity criteria rather than experimenting with different pruning cut-offs. Another impurity measure is the classification error:

$$I_E = 1 - \max\left\{p(i|t)\right\}$$

This is a useful criterion for pruning but not recommended for growing a decision tree, since it is less sensitive to changes in the class probabilities of the nodes. We can illustrate this by looking at the two possible splitting scenarios shown in the following figure:

We start with a dataset D p at the parent node D p that consists of 40 samples from class 1 and 40 samples from class 2 that we split into two datasets D left and D right, respectively. The information gain using the classification error as a splitting

criterion would be the same (IG E = 0.25) in both scenarios A and B:

$$I_E\left(D_p\right)=1-0.5=0.5$$

$$A: I_E\left(D_{left}\right)=1-\frac{3}{4}=0.25$$

$$A: I_E\left(D_{right}\right)=1-\frac{3}{4}=0.25$$

$$A: IG_E=0.5-\frac{4}{8}0.25-\frac{4}{8}0.25=0.25$$

$$B: I_E\left(D_{left}\right)=1-\frac{4}{6}=\frac{1}{3}$$

$$B: I_E\left(D_{right}\right)=1-1=0$$

$$B: IG_E=0.5-\frac{6}{8}\times\frac{1}{3}-0=0.25$$

However, the Gini index would favor the split in scenario B (IG G = 0.16) over scenario A (IG G = 0.125), which is indeed more pure:

$$I_H\left(D_p\right)=-\left(0.5\ \log_2\left(0.5\right)+0.5\ \log_2\left(0.5\right)\right)=1$$

$$A: I_H\left(D_{left}\right)=-\left(\frac{3}{4}\log_2\left(\frac{3}{4}\right)+\frac{1}{4}\log_2\left(\frac{1}{4}\right)\right)=0.81$$

$$A: I_H\left(D_{right}\right)=-\left(\frac{1}{4}\log_2\left(\frac{1}{4}\right)+\frac{3}{4}\log_2\left(\frac{3}{4}\right)\right)=0.81$$

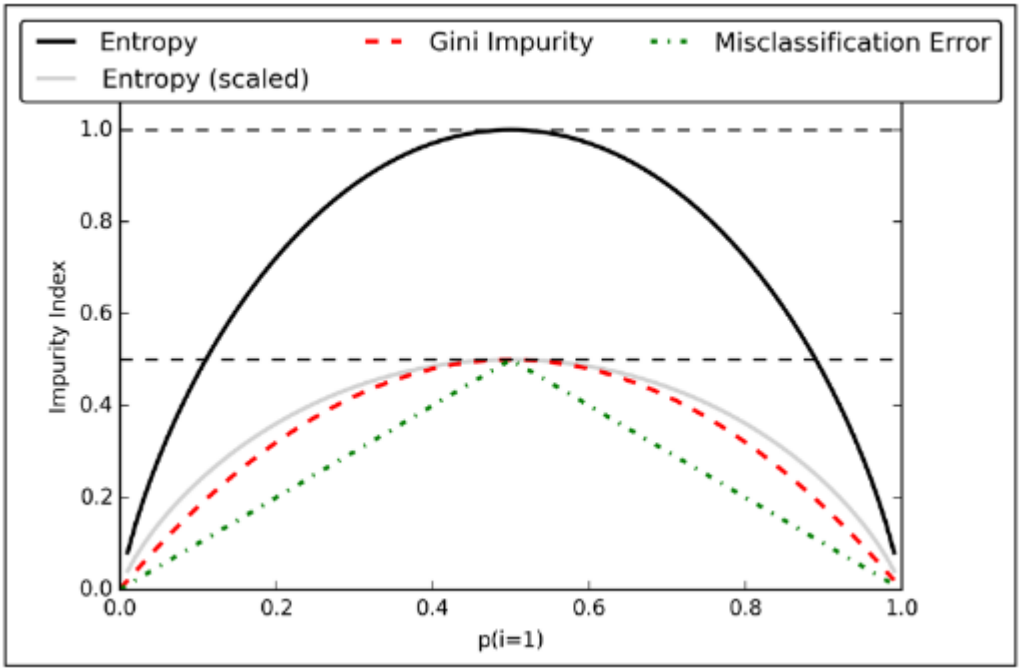
$$A: IG_H=1-\frac{4}{8}0.81-\frac{4}{8}0.81=0.19$$

$$B: I_H\left(D_{left}\right)=-\left(\frac{2}{6}\log_2+\left(\frac{2}{6}\right)+\frac{4}{6}\log_2+\left(\frac{4}{6}\right)\right)=0.92$$

$$B: I_H\left(D_{right}\right)=0$$

$$B: IG_H=1-\frac{6}{8}0.92-0=0.31$$

For a more visual comparison of the three different impurity criteria that we discussed previously, the plot of the impurity indices for the probability range [0, 1] for class 1 is given below.



So, after training the classifiers with the features collected from movie review corpus we can predict the Gmail if it is positive or negative. The same technique can be applied on twitter, Facebook and other social medias. So with the help of this we can build multiple label such as if the Gmail is crime based, erotic, funny or religious. By finding these labels we can find out a person's motive. For crime investigation and also parental monitoring of their kids it will be helpful. Specific application can be if any kid is involved in ISIS or another crime organization and if they communicate through the Gmail or other social medias then we can detect the crime based texts and parents or regarding authority can take proper actions.

Data Visualization: For visualizing the data we have used live plot graph and pie-chart. Here's sample of output of our application which was used against the Gmail id taher9abu@gmail.com.

```
86   I have to knit some dust bunnies for a charity bazaar.
87   I'm having my baby shoes bronzed.
88   I have to go to court for kitty littering.
89   I'm going to count the bristles in my toothbrush.
90   I have to thaw some karate chops for dinner.
91   Having fun gives me prickly heat.
92   I'm going to the Missing Persons Bureau to see if anyone is looking
     for me.
93   I have to jog my memory.
94   My palm reader advised against it.
95   My Dress For Obscurity class meets then.
96   I have to stay home and see if I snore.
97   I prefer to remain an enigma.
98   I think you want the OTHER  [your name].
99   I have to sit up with a sick ant.
100  I'm trying to cut down.

pos
```

```
OPTION B: You do the same as above, but you use gasoline instead, this
produces a fire in many cases afterwords. Great for mailboxes!

WICKS: This is a pretty good way to make wicks.
Ok here is the stuff:
               piece of string, not yarn (howeverlong you want it)
               gunpowder
               a plate or something flat
               LITTLE bit of water

Here we go, get the string and roll it in the gunpowder, then LIGHTLY wet the
gunpowder, roll the sting in the several times. Make sure you have the string
pretty well coated, but not too much, or else it will burn too quickly.
OPTION A: If you are about to use the wick in a couple of secs, use gasoline
instead of water, gasoline will work better.
OPTION B: Use wax to coat the string first, get it so the wax is not too hot,
but cool enough to roll gunpowder in. Coat the string once again.

neg
```

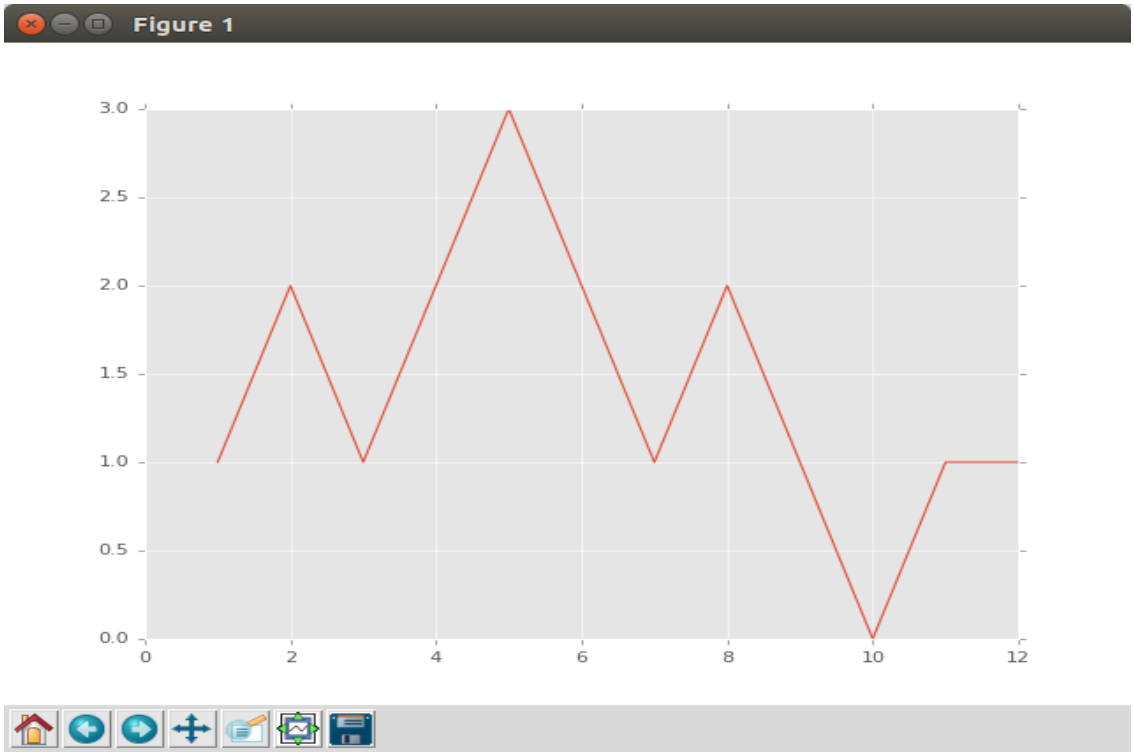


Figure 1
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8
10
12

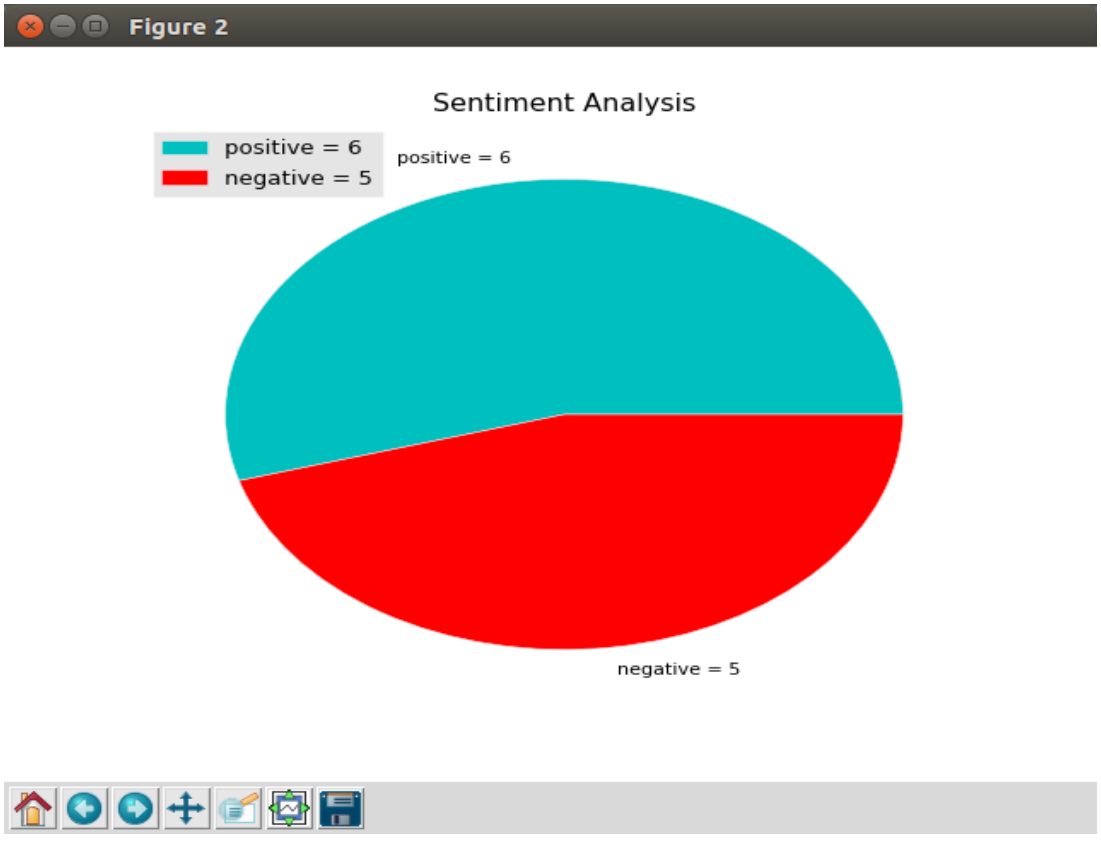


Figure 2
Sentiment Analysis
positive = 6
negative = 5
positive = 6
negative = 5

References:

Python3 Text Processing with NLTK 3 Cookbook by Jacob Perkins

Python Machine Learning by Sebastian Raschka

Stanford Machine Learning Course by Prof. Andrew Ng

https://en.wikipedia.org/wiki/Phi_coefficient

https://pythonprogramming.net/