CONTENTS

List of Figures

1. ABSTRACT

Emotions are a part of human communication. Mood, feelings and personality of any human being can be recognized through his emotions. People have always found music significant in their lives. Music is a language of an emotion. It frequently expresses emotional qualities and qualities of human personality such as happy, sadness, aggressiveness, tenderness etc. It has a central role in human society because it so strongly evokes feelings and affects social activities and interactions.

Music mood recognition is the process wherein the emotions of a musical piece are identified through various means including the analysis of audio and lyrical text . Most of the research on music classification is based on features obtained by audio signals . However, as shown in [1] the semantic (lyrics) and harmonic (tunes) information are processed independently by the brain, even when these information sources are closely related to each other. So, the exploration of lyrics alone as a source of information can be relevant in music.

Lyrics provide high-level information about a song. We can gather this meta-information such as the genre, sentiment, and theme of a song simply by reading its lyrics. This project aims to automate the same. The Song Lyrics Dataset needs to be cleaned, structured and preprocessed using the Natural Language Processing approach. Further the classification into particular mood needs to be governed by Machine Learning Algorithms.

The goal of this project is to investigate which features extraction method is the most promising and efficient for mood classification and to build a classifier that is able to predict whether a song is happy, sad, relaxed or angry based on its lyrics. Such a classifier can easily find it’s use in the music industry for example it can be used to classify and form playlists automatically in music players.

2. NATURAL LANGUAGE PREPROCESSING (NLP)

2.1 Introduction

Natural-language processing (NLP) is an area of computer science and artificial intelligence concerned with the processing of human natural languages by computers. It deals with how to program computers to fruitfully process large amounts of natural language data. Here we attempt to build programs that can understand human natural language. These are then used to process huge bulks of data.

Natural language processing is a field with a lot of applications. Listed below are a few of them.

- **Machine Translation:** Machine translation is the problem of converting a source text in one language to another language.
- **Sentiment analysis:** The goal of sentiment analysis is to identify sentiment of a given text where emotion is not always explicitly expressed.
- **Spam filtering:** The task here is given a E- mail classify it as spam or not.
- **Caption Generation**: Caption generation is the problem of describing the contents of an image. Given a digital image, such as a photo, generate a textual description of the contents of the image.
- **Document Summarization:** Document summarization is the task where a short description of a text document is created.
- **Question Answering:** Question answering is the problem where given a subject, such as a document of text, answers specific questions about the subject.
- **Text Prediction (Auto-Complete):** This is commonly seen in chat applications. The task here is given a few words predict the next (one or few words).

Natural Language processing is a very wide field of study with a lot of topics. Here we will briefly introduce only the topics we are going to use in our project.

2.2 Text Tokenisation

The goal of this stage is to divide the text into relevant units such as **individual words (tokens)** or **sequences of words** (n-grams). In particular, tokenisation refers to the task of segmenting a text into individual words or word-like units.

The word_tokenizer and regexp_tokenizer functions from NLTK library are used for tokenisation in our project. It is worth highlighting the importance of stop-words removal. Stop-words should be removed from a text because they make the text look heavier and less important for analysis. Removing stop words reduces the dimensionality of term space too. Hence it increases both speed and accuracy. We are removing all words with less than or equal to two letters in them in addition to all nltkstopwords.

2.3 Stemming

Stemming is the process of reducing inflected or derived words to their word stem, base or root form—generally a written word form. The stem need not be identical to the morphological root of the word, it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid word.

For example, the words develop, developer, development, developing all can be stemmed to the word "develop". The goal of this method is to remove various suffixes. This reduces the number of words and also would related word as one single word which is a smart idea since the related words would mean same thing anyway.

The following are a few **stemming algorithms**:

1. **Porter stemmer** is introduced by M.F. Porter it works by treating complex suffixes as compounds made up of simple suffixes, and removing the simple suffixes in a number of steps.
2. **Lancaster stemmer** is the fastest but it's aggressive method.
3. **Snowball stemmer** is an updated and more efficient version of the original Porter stemmer. It provides the possibility of stemming with different languages and also the stop-words.

All these algorithms are provided by NLTK library. However since our data-set is small we would only be using Porter and Snowball stemmers.

2.4 Lemmatisation

This is a similar process to stemming. It is also used for reducing inflected or derived words to their word base or root form (here called lemma). However lemmatisation unlike stemming gives dictionary form base word and uses word-net to find relation between words. For some words lemmatisation works wonders which stemmers just can't do.

For example: *good*, *better* and *best* are lemmatised to *good*, *good* and *good* respectively.

The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

However, the two differ in their flavour. Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma.

2.5 Bag of Words

A bag-of-words model, or BoW for short, is a way of extracting features from text for use in modelling, such as with machine learning algorithms. The approach is very simple and flexible, and can be used in a myriad of ways for extracting features from documents.

A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:

- ❖ A vocabulary of known words.
- ❖ A measure of the presence of known words.

It is called a “*bag*” of words, because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document.

2.6 N-Grams

N-gram is a contiguous sequence of n items from a given sequence of text. Instead of tokenizing into single words (unigram), we can tokenize words into sets of two (bigram) or three (trigram) words. This helps because most of the time it is the occurrence of a phrase (or collection of words) that tells us the sentiment of the song more than single words occurring in isolation.

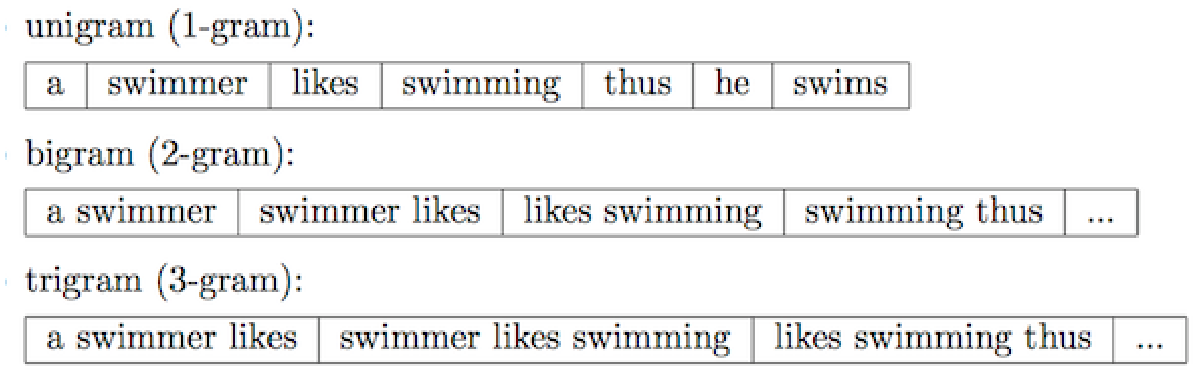

unigram (1-gram):
a | swimmer | likes | swimming | thus | he | swims
bigram (2-gram):
a swimmer | swimmer likes | likes swimming | swimming thus | ...
trigram (3-gram):
a swimmer likes | swimmer likes swimming | likes swimming thus | ...

Fig 2.1 N-grams : Unigram, Bigram and Trigram

2.7 Term Frequency-Inverse Document Frequency (TF-IDF)

A problem with scoring word frequency is that highly frequent words start to dominate in the document (e.g. larger score), but may not contain as much “informational content” to the model as rarer but perhaps domain specific words.

One approach is to rescale the frequency of words by how often they appear in all documents, so that the scores for frequent words like “the” that are also frequent across all documents are penalized.

This approach to scoring is called Term Frequency – Inverse Document Frequency, or TF-IDF for short, where:

- **Term Frequency**: The term frequency of a term ‘t’ in a document ‘d’ is the number of time the term ‘t’ occurs in the document ‘d’ by number of terms in the document.
- **Inverse Document Frequency**: is a scoring of how rare the word is across documents.

The scores are a weighting where not all words are equally as important or interesting.

The scores have the effect of highlighting words that are distinct (contain useful information) in a given document.

IDF of a term is calculated as the number of documents in document set by number of documents where the term occurs. This raw IDF is normally taken in log or taken as it is or in square or even as cube depending on number of documents in document-set.

Together **TF-IDF gives** the **importance of a word in a document**. The TF-IDF score is calculated as TF*IDF.

TF-IDF Score

$$TF - IDF\ Score = TF_{x,y} * IDF = TF_{x,y} *\ log\frac{N}{df} \ldots\ldots.(1)$$

, where $TF_{x,y}$ *is the frequency of keyphrase X in the article Y,*

N is the total number of documents in the corpus.

df is the number of documents containing keyphrase X

Fig 2.2 TF-IDF Score

3. MACHINE LEARNING (ML)

3.1 Introduction

Machine learning is a field of computer science that uses statistical techniques to give computer systems the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed.

“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.” - formal definition provided by Tom M. Mitchell.

Machine learning tasks are typically classified into two broad categories:

- **Supervised Learning:** The computer is presented with example inputs and their desired outputs and the goal is to learn a general rule that maps the given inputs to outputs.
- **Unsupervised Learning:** No labels are given to the learning algorithm, leaving it on its own to find structure in its input. The goal in unsupervised learning is to discover hidden patterns in data

Like NLP Machine Learning is also a very vast field of study. So we will briefly introduce only the topics we are going to use in our project.

3.2 Classification

Classification is a type of supervised learning. It is a learning procedure based on the statistical learning theory. A classifier is a system that performs a mapping from a feature space X to a set of labels (also called classes) Y. A classifier assigns a pre-defined class label to a sample. In a supervised-learning context classes are pre-defined and samples from these classes are given. The classifier goal is then to model the observed data (called ground truth) to classify new instances with the highest accuracy possible. The classifier aims at discovering relationships between descriptors of samples from the ground truth and the class labels. Classifiers are trained on positive (and sometimes negative) examples of the to-be-learned class and tested on new unknown data.

Given the dataset composed by features vectors, the goal is to train a classifier in order to be able to predict whether a song is angry, happy, sad or relaxed based on its lyrics. Now we will discuss the three classification algorithms that are used in our project.

3.2.1 Naïve Bayes

Definition:

Naïve Bayes algorithm is based on Bayes' theorem with the assumption of independence between every pair of features or among the predictors. In simple terms, a Naïve Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'. This classifier works well in many real-world situations such as document classification and spam filtering.

The algorithm performed on the dataset is **Multinomial Naïve Bayes.** It is used for discrete counts. Here we consider Bernoulli trials which is one step further and instead of "word occurring in the document", we have "count how often word occurs in the document", i.e. the number of times outcome number x_i is observed over the n trials.

Advantages:

- Naïve Bayes classifiers are extremely easy and fast compared to more sophisticated methods.
- When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.
- It performs well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).

Disadvantages:

- If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as "Zero Frequency". To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.
- Naïve Bayes is is known to be a bad estimator.
- Another limitation of Naïve Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

Formula:

$$\underbrace{P(c\,|\,x)}_{\text{Posterior Probability}} = \frac{\overbrace{P(x\,|\,c)}^{\text{Likelihood}}\,\overbrace{P(c)}^{\text{Class Prior Probability}}}{\underbrace{P(x)}_{\text{Predictor Prior Probability}}}$$

$$P(c\,|\,X) = P(x_1\,|\,c) \times P(x_2\,|\,c) \times \cdots \times P(x_n\,|\,c) \times P(c)$$

Fig 3.1 Class (c, target) , Predictor (x, attributes) for Naïve Bayes

3.2.2 Support Vector Machine:

Definition: “Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well

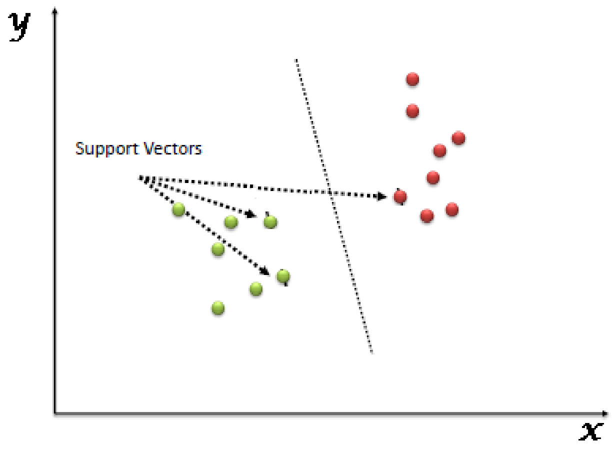


Fig 3.2 Hyper-plane differentiating two classes

Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line).

How does it work?

The burning question is "How can we identify the right hyper-plane?".

- ❖ **Identify the right hyper-plane (Scenario-1):** Here, three hyper-planes (A, B and C) are provided. To identify the right hyper-plane to classify star and circle, the thumb rule is : "Select the hyper-plane which segregates the two classes better". In this scenario, hyper-plane "B" has excellently performed this job.

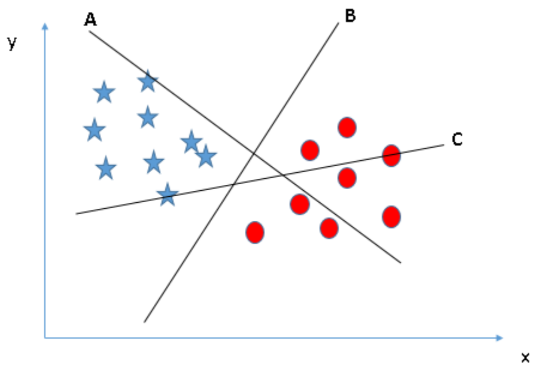


Fig 3.3 Hyper-plane Scenario-1

- ❖ **Identify the right hyper-plane (Scenario-2):** Here, the three hyper-planes (A, B and C), all are segregating the classes well. Maximizing the distances between nearest data point (either class) and hyper-plane will decide the right hyper-plane. This distance is

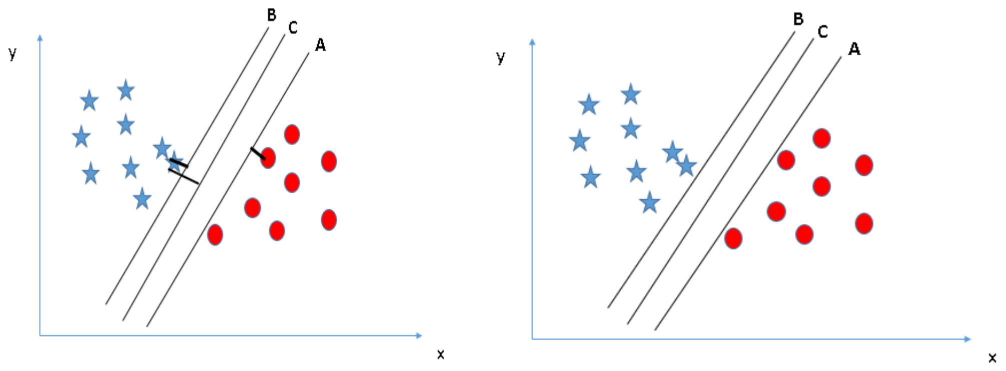


Fig 3.4 Hyper-plane Scenario-2

called as **Margin**. Above, you can see that the margin for hyper-plane C is high as compared to both A and B. Name the right hyper-plane as C. Another lightning reason for selecting the hyper-plane with higher margin is robustness. If we select a hyper-plane having low margin then there is high chance of miss-classification.

- ❖ **Identify the right hyper-plane (Scenario-3)**

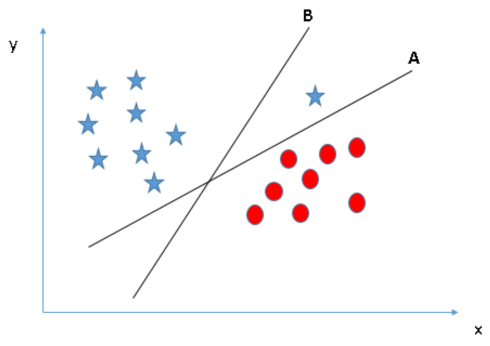


Fig 3.5 Hyper-plane Scenario-3

SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin. Here, hyper-plane B has a classification error and A has classified all correctly. Therefore, the right hyper-plane is **A.**

❖ **Can we classify two classes (Scenario-4)?:** Below, Unable to segregate the two classes using a straight line, as one of star lies in the territory of other(circle) class as an outlier.?

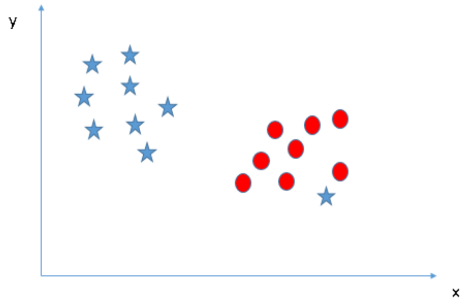


Fig 3.6 Hyper-plane Scenario-4

One star at other end is like an outlier for star class. SVM has a feature to ignore outliers and find the hyper-plane that has maximum margin. Hence, SVM is robust to outliers.

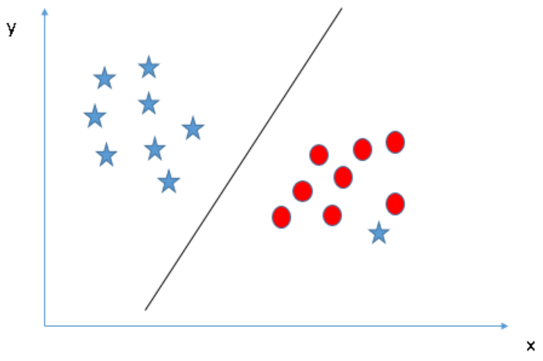


Fig 3.7 Hyper-plane Scenario-4 (Ignore Outliers)

❖ **Find the hyper-plane to segregate to classes (Scenario-5):**

In the scenario below, we can't have linear hyper-plane between the two classes, so how does SVM classify these two classes?

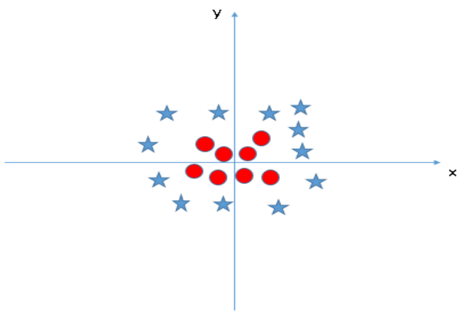


Fig 3.8 Hyper-plane Scenario-5 I

SVM can solve this problem. Easily! It solves this problem by introducing additional feature. Here, we will add a new feature z=x^2+y^2. Now, let's plot the data points on axis x and z:

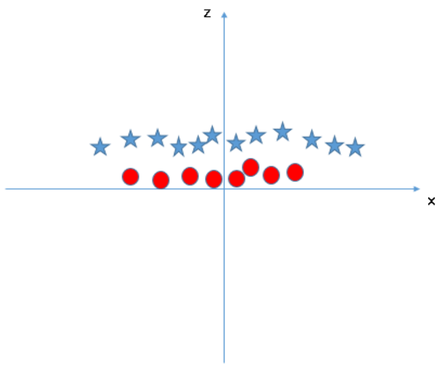


Fig 3.9 Hyper-plane Scenario-5 II

In above plot, points to consider are:

- All values for z would be positive always because z is the squared sum of both x and y
- In the original plot, red circles appear close to the origin of x and y axes, leading to lower value of z and star relatively away from the origin result to higher value of z.

In SVM, it is easy to have a linear hyper-plane between these two classes. But, another burning question which arises is, should we need to add this feature manually to have a hyper-plane. No, SVM has a technique called the **kernel trick**. These are functions which takes low dimensional input space and transform it to a higher dimensional space i.e. it converts not separable problem to separable problem, these functions are called kernels. It is mostly useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then find out the process to separate the data based on the labels or outputs you've defined.

When we look at the hyper-plane in original input space it looks like a circle:

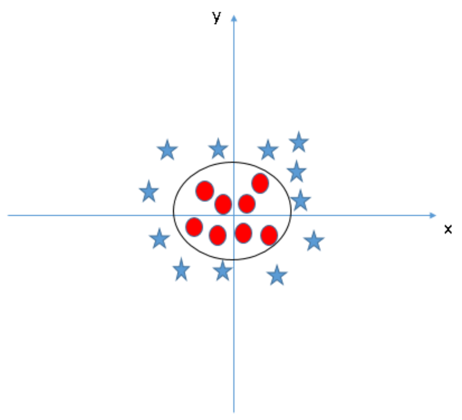


Fig 3.10 Hyper-plane Scenario-5 III

Advantages:

- Effective in high dimensional spaces and uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Works well with clear margin of separation
- Effective in cases where number of dimensions is greater than the number of samples.

Disadvantages:

- The algorithm does not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.
- It doesn't perform well, when we have large data set because the required training time is higher as well as when the data set has more noise i.e. target classes are overlapping.

3.2.3 Logistic Regression:

Definition: Logistic Regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function. The dependent variable is the target class variable to be predicted while the independent variables are the features or attributes that are used to predict the target class. It is usually implemented for binary classification.

If the logistic regression algorithm is used for the multi-classification task, then the same logistic regression algorithm called a Multinomial Logistic Regression.

In the logistic regression, the logistic function which takes the input features and calculates the probabilities of the possible two outcomes is the Sigmoid Function. Later the high probabilities target class is the final predicted class from the logistic regression classifier. In multinomial logistic regression ,the Softmax Function is used as the logistic function.

$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

Fig 3.11 Softmax Function

Advantages:

- Works well with diagonal (feature) decision boundaries
- Multicollinearity is not really an issue and can be countered with L2 regularization to an extent.
- Lots of ways to regularize the model, hence not much effect of the features being correlated
- Low Variance

Disadvantages:

- Doesn’t perform well when feature space is too large.
- Doesn’t handle large number of categorical features/variables well.
- Relies on transformations for non-linear features.
- High Bias

4. TECHNOLOGIES USED

Our project would be built on python3 programming language. Python is an interpreted high-level programming language for general-purpose programming. Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library. Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations.

We would be using third party libraries Pandas, NumPy, SciPy, Scikit-Learn and NLTK for different functionalities.

4.1 Pandas

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

Pandas allows us to focus more on research and less on programming. Pandas is easy to learn, easy to use, and easy to maintain. The bottom line is that it has increased our productivity.

Python has long been great for data munging and preparation, but less so for data analysis and modelling. Pandas helps fill this gap, enabling you to carry out your entire data analysis workflow in Python without having to switch to a more domain specific language like R.

The best way to get pandas is via either conda or pip:

- conda install pandas
- pip3 install pandas

4.2 Numpy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

The best way to get pandas is via pip:

- pip install numpy

4.3 SciPy

SciPy is an open-source Python library used for scientific computing and technical computing.

SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.

The best way to get SciPy is via pip:

- pip install scipy

4.4 Scikit-learn

Scikit-learn (formerly scikits.learn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Scikit-learn requires:

- Python (>= 3.3),
- NumPy (>= 1.8.2),
- SciPy (>= 0.13.3).

If you already have a working installation of numpy and scipy, the easiest way to install scikit-learn is using pip:

- Pip install scikit-learn

4.5 NLTK:

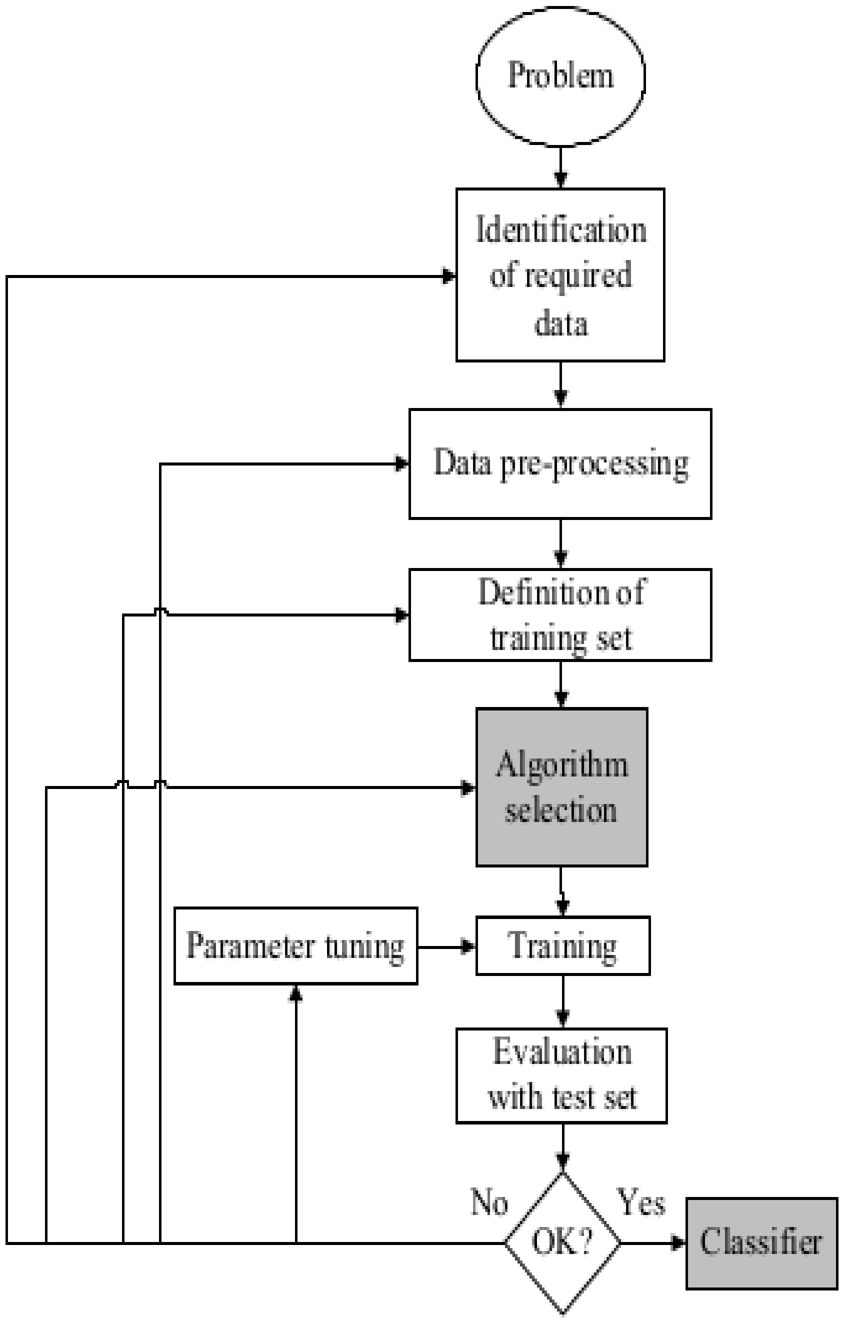
NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, semantic reasoning and wrappers for industrial-strength NLP libraries.

The easiest way to install nltk is using pip:

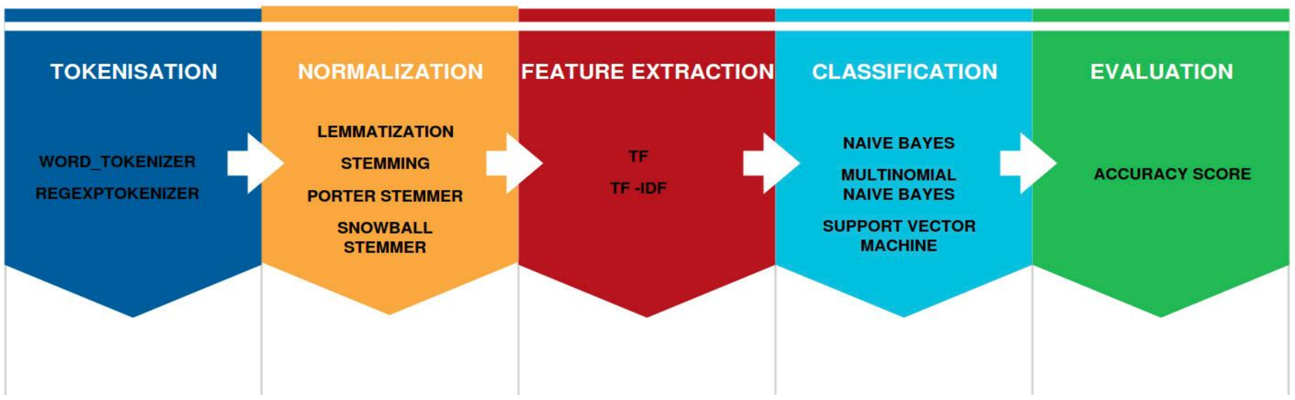
- Pip install nltk

5. RESEARCH METHODOLOGY

5.1 FLOWCHART



5.2 STEPS TO BE FOLLOWED



Tokenizers Used:

1.Word-Tokenizer(nltk)
2.RegexpTokenizer(nltk)

Stop Words:

1.nltkstopwords
2.scikit-learn stopwords
3.single letter words

Stemmers/Lemmatizers:

1.Porter Stemmer
2.Snowball Stemmer
3.WordnetLemmatizer

Vectorizers:

1.Count (Term-Frequency) Vectorizer
2.TF-IDF Vectorizer
3.TF-IDF Vectorizer + POS(parts of speech)Selection

Classification Models:

1. Naïve Bayes Classifier
2. Support Vector Machine(SVM) Classifier
3. Logistic Regression Classifier

5.3. DATASET

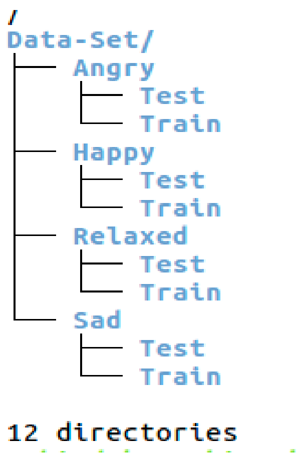
The dataset consists of 400 music clips of 4 mood categories - angry, happy, relaxed and sad. From which 300 clips are used as the training set and the other 100 ones are used for testing. The distribution of clip samples in 4 mood categories are shown in the following table:

Mood Category	**Training Samples**	**Testing Samples**
angry	75	25
happy	75	25
relaxed	75	25
sad	75	25

For each music clip in the dataset, a plain text (.txt) file is provided consisting of every sentences of music lyrics, along with the time tags (i.e. the time offset [hour:minute.second] of the sentence relative to the start of the music).
Further, an info.txt file is maintained for each training and testing set comprising of the following :

1. S.No (Match for the Corresponding Lyrics File)
2. Song Name
3. Artist's Name
4. Duration of Music (in seconds)

5.3.1 STRUCTURE OF DATASET



```
/
Data-Set/
├── Angry
│   ├── Test
│   └── Train
├── Happy
│   ├── Test
│   └── Train
├── Relaxed
│   ├── Test
│   └── Train
└── Sad
    ├── Test
    └── Train

12 directories
```

Each Test and Train data here has several .txt files for lyrics and one info.txt that give info of the songs in that folder.

For fast and efficient data reading, the training and testing samples were combined to form csv files - **train.csv** and **test.csv** respectively.

5.4 PREPROCESSING OF DATA

Since, text is the most unstructured form of all the available data, various types of noise are present in it and hence the collected lyrics in the dataset cannot be analyzed in their raw form. The entire process of cleaning and standardization of text, making it noise-free and ready for analysis is known as text preprocessing.
Data preprocessing methods play a very important role in text mining techniques and applications.

5.5 FEATURE EXTRACTION

The next step is to convert text into features, which give a simpler and more focused view of the text to a machine learning model.

A commonly used model in natural language processing is the so-called bag of words model, where a vocabulary is created: the collection of all different words that occur in the dataset and each word is associated with a count of how it occurs. This vocabulary can be understood as a set V of non-redundant items where the order doesn’t matter. The vocabulary can then be used to construct the d -dimensional feature vectors for the individual documents where the dimensionality is equal to the number of different words in the vocabulary (d = |V |). The

feature vector can be binary (a word occurs/doesn’t occur in a document) or absolute (how often the word occurs in each document).
The following features extraction methods are investigated:

- unigram , bigrams - in the lyrics
- term-frequency (tf) accounts for the number of times a term t occurs in document d. This work analyses only the frequency of unigram
- term frequency – inverse document frequency (tf-idf) combines the tf metric described above with inverse document frequency (idf), which gives higher weights to terms which are rare in the collection. Our work will analyze both unigram and unigram combined with bigram.

5.6 CLASSIFICATION

Classification is a learning procedure based on the statistical learning theory. A classifier is a system that performs a mapping from a feature space X to a set of labels (also called classes) Y . A classifier assigns a pre-defined class label to a sample. In a supervised-learning context classes are pre-defined and samples from these classes are given. The classifier goal is then to model the observed data (called ground truth) to classify new instances with the highest accuracy possible. The classifier aims at discovering relationships between descriptors of samples from the ground truth and the class labels. Classifiers are trained on positive (and sometimes negative) examples of the to-be-learned class and tested on new unknown data.

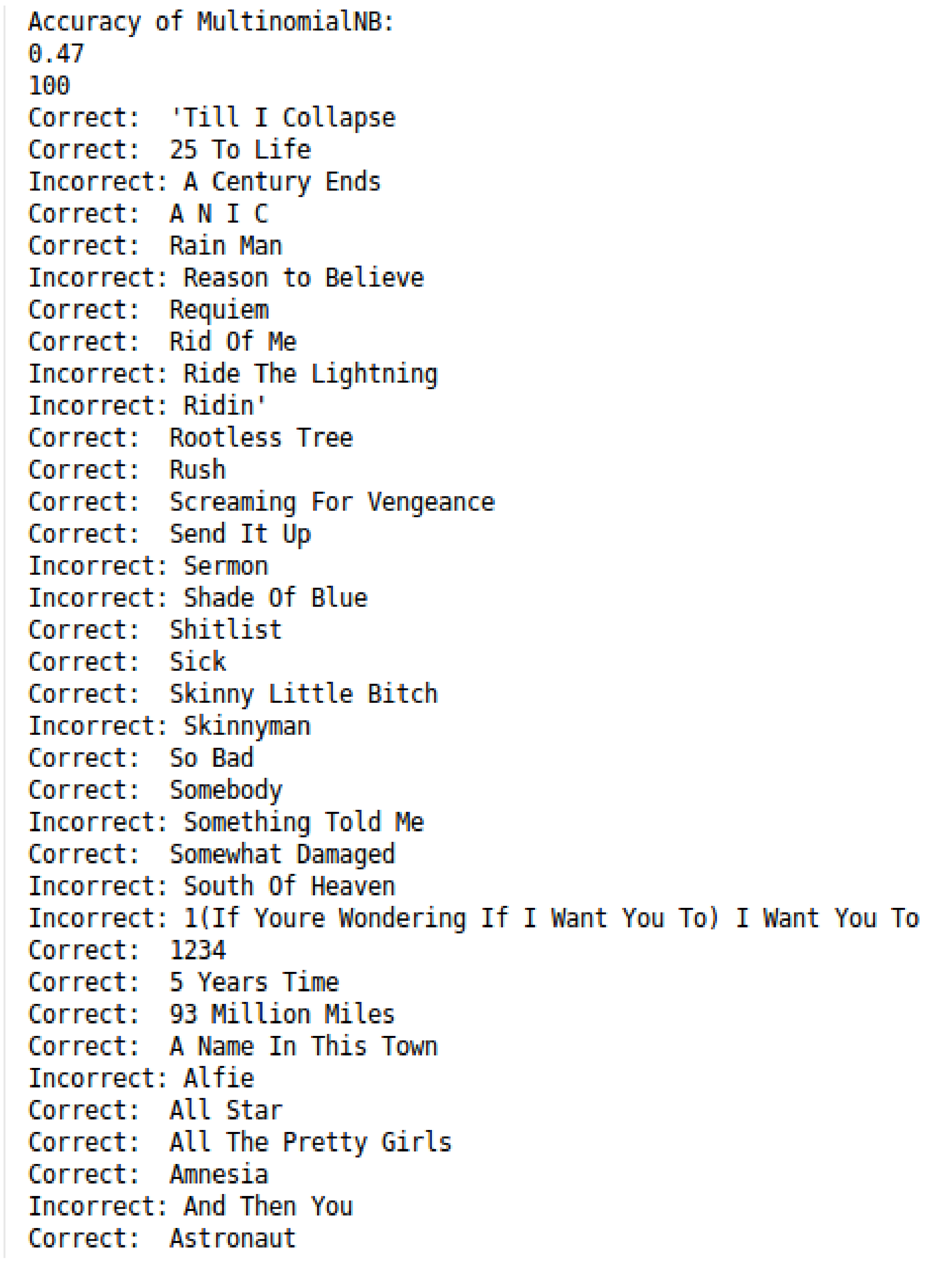
Given the dataset composed by features vectors, the goal is to train a classifier in order to be able to predict whether a song’s lyrics is happy or sad.
The classification algorithms implemented over the dataset are:
NAÏVE BAYES, SVM and LOGISTIC REGRESSION.

6. EVALUATION AND RESULTS

To evaluate the quality of a classifier, we usually compare its prediction with ground truth data.

Accuracy = (Count (Correctly Classified) / Total Tests given)*100

Hence, accuracy is the percentage of correctly classified items.



```
Accuracy of MultinomialNB:
0.47
100
Correct:  'Till I Collapse
Correct:  25 To Life
Incorrect: A Century Ends
Correct:  A N I C
Correct:  Rain Man
Incorrect: Reason to Believe
Correct:  Requiem
Correct:  Rid Of Me
Incorrect: Ride The Lightning
Incorrect: Ridin'
Correct:  Rootless Tree
Correct:  Rush
Correct:  Screaming For Vengeance
Correct:  Send It Up
Incorrect: Sermon
Incorrect: Shade Of Blue
Correct:  Shitlist
Correct:  Sick
Correct:  Skinny Little Bitch
Incorrect: Skinnyman
Correct:  So Bad
Correct:  Somebody
Incorrect: Something Told Me
Correct:  Somewhat Damaged
Incorrect: South Of Heaven
Incorrect: 1(If Youre Wondering If I Want You To) I Want You To
Correct:  1234
Correct:  5 Years Time
Correct:  93 Million Miles
Correct:  A Name In This Town
Incorrect: Alfie
Correct:  All Star
Correct:  All The Pretty Girls
Correct:  Amnesia
Incorrect: And Then You
Correct:  Astronaut
```

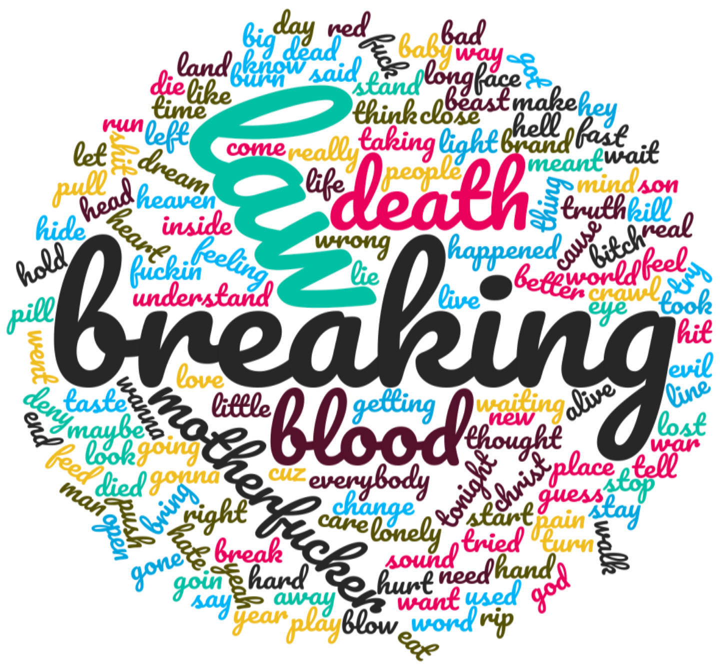
Fig 6.1 Accuracy score of Naïve Bayes Classifier based on the frequency of words present in the created vocabulary

6.1 Summary of Basic Lyric Features

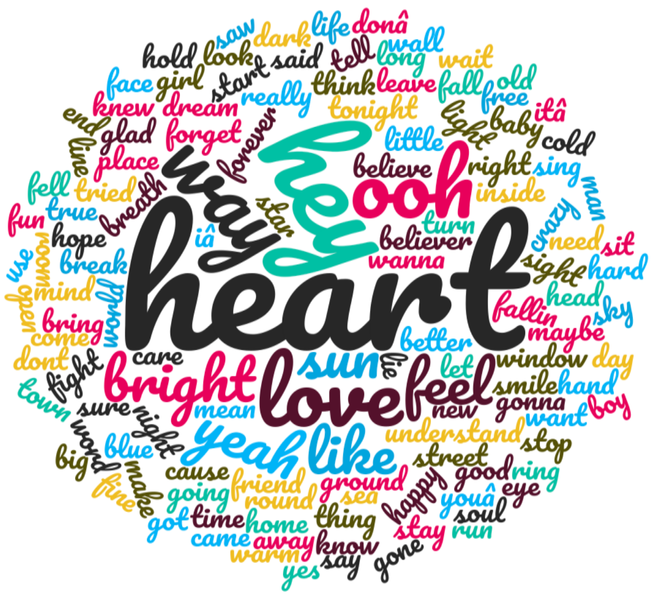
Feature Type	n-grams	No. of Dimensions/ Word Tokenizer	No. of Dimensions/ RegexpTokenizer
Content words	**Unigram**	*5052*	*4856*
	Bigram	*20924*	*19629*
	Trigram	*25427*	*22428*
	Uni + Bigram	*25976*	*24485*
	Uni + Bi + Trigram	*51403*	*46913*

6.2 Wordclouds

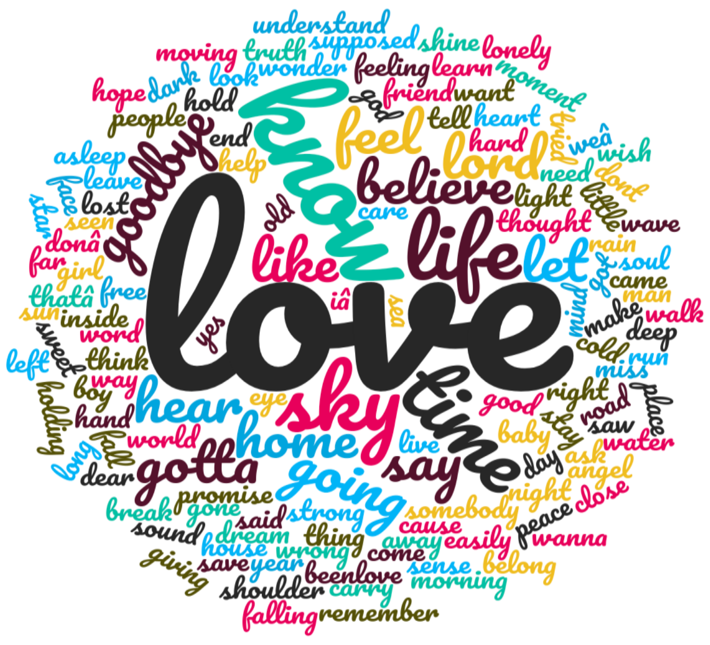
Angry



Happy



Relaxed



Sad

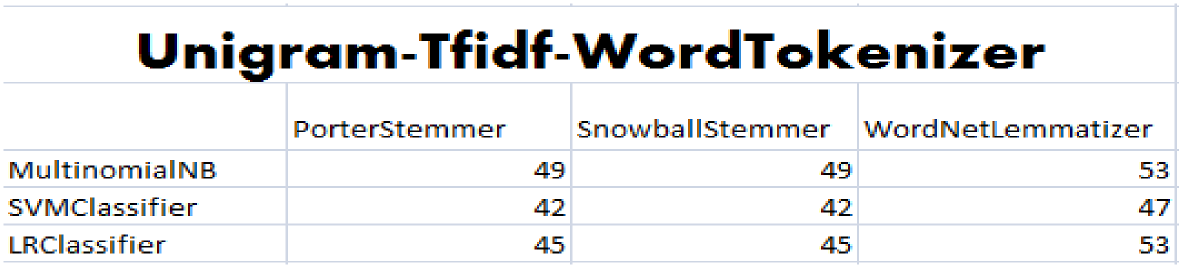
6.3 Based on Term Frequency

Term Frequency			
REGEXP + LEMMATIZER	NAIVE BAYES	SVM	LOGISTIC REGRESSION
Unigram	57	42	46
Uni+Bi	55	44	48
Uni+Bi+Tri	52	43	45

Clearly, **Multinomial Naive Bayes (MNB)** shows highest accuracy based on term frequency data. Further, it shows greater precision for uni + bigram and uni + bi + trigram combinations as compared to individual grams.

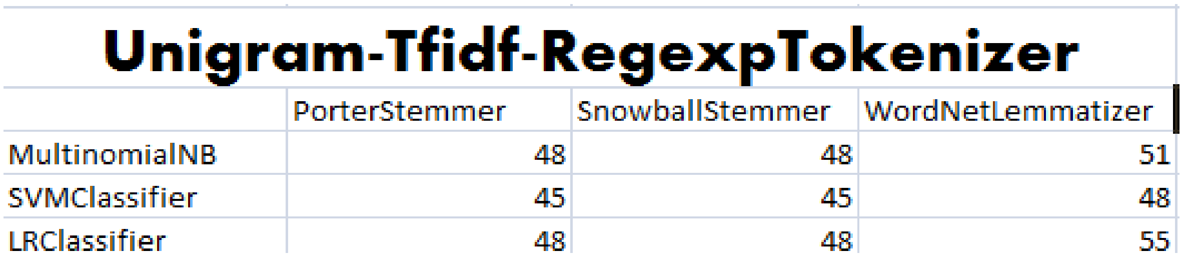
6.4 Based on TF-IDF

Summary of the results followed by the best outputs are shown.



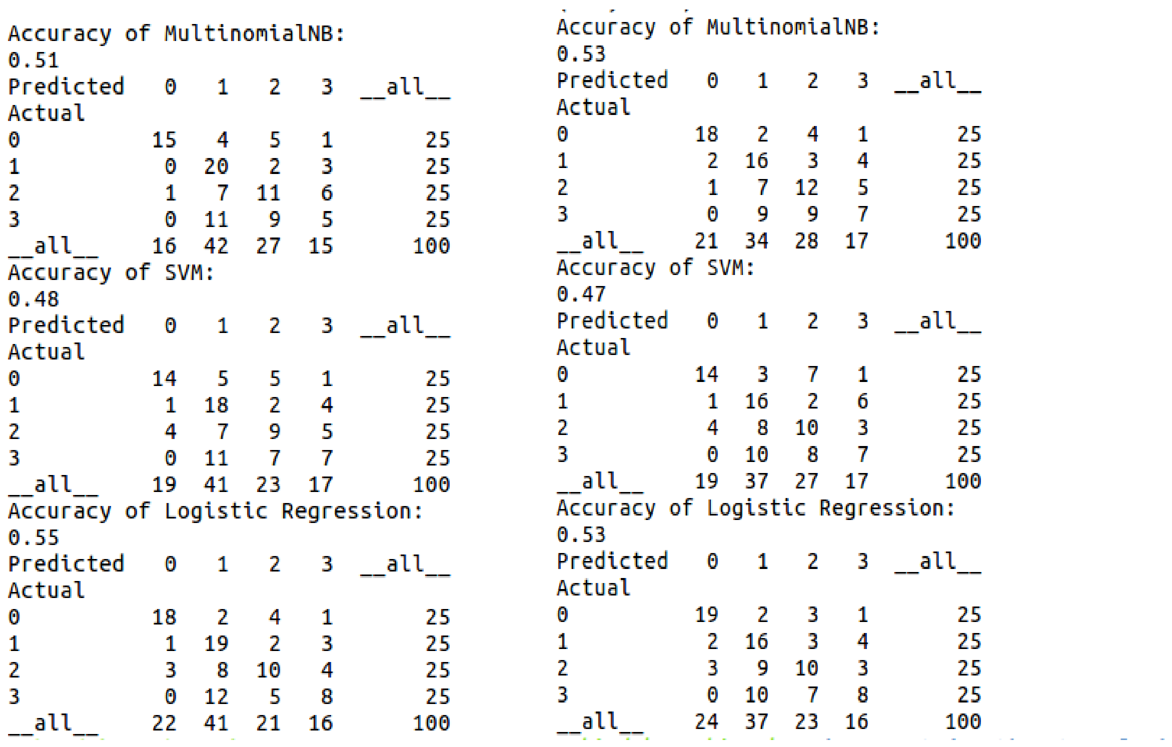
Unigram-Tfidf-WordTokenizer

	PorterStemmer	SnowballStemmer	WordNetLemmatizer
MultinomialNB	49	49	53
SVMClassifier	42	42	47
LRClassifier	45	45	53



Unigram-Tfidf-RegexpTokenizer

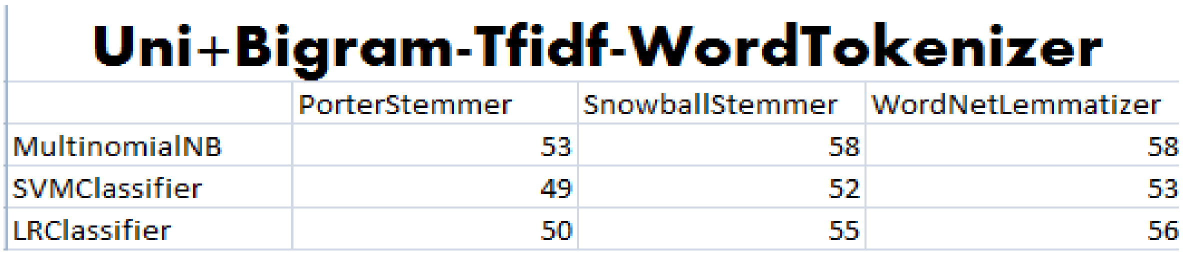
	PorterStemmer	SnowballStemmer	WordNetLemmatizer
MultinomialNB	48	48	51
SVMClassifier	45	45	48
LRClassifier	48	48	55



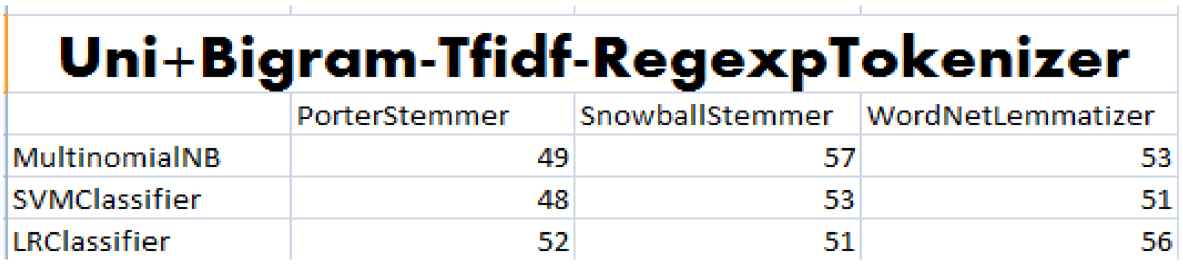
```
Accuracy of MultinomialNB:
0.51
Predicted    0   1   2   3  __all__
Actual
0           15   4   5   1       25
1            0  20   2   3       25
2            1   7  11   6       25
3            0  11   9   5       25
__all__     16  42  27  15      100
Accuracy of SVM:
0.48
Predicted    0   1   2   3  __all__
Actual
0           14   5   5   1       25
1            1  18   2   4       25
2            4   7   9   5       25
3            0  11   7   7       25
__all__     19  41  23  17      100
Accuracy of Logistic Regression:
0.55
Predicted    0   1   2   3  __all__
Actual
0           18   2   4   1       25
1            1  19   2   3       25
2            3   8  10   4       25
3            0  12   5   8       25
__all__     22  41  21  16      100
```

```
Accuracy of MultinomialNB:
0.53
Predicted    0   1   2   3  __all__
Actual
0           18   2   4   1       25
1            2  16   3   4       25
2            1   7  12   5       25
3            0   9   9   7       25
__all__     21  34  28  17      100
Accuracy of SVM:
0.47
Predicted    0   1   2   3  __all__
Actual
0           14   3   7   1       25
1            1  16   2   6       25
2            4   8  10   3       25
3            0  10   8   7       25
__all__     19  37  27  17      100
Accuracy of Logistic Regression:
0.53
Predicted    0   1   2   3  __all__
Actual
0           19   2   3   1       25
1            2  16   3   4       25
2            3   9  10   3       25
3            0  10   7   8       25
__all__     24  37  23  16      100
```

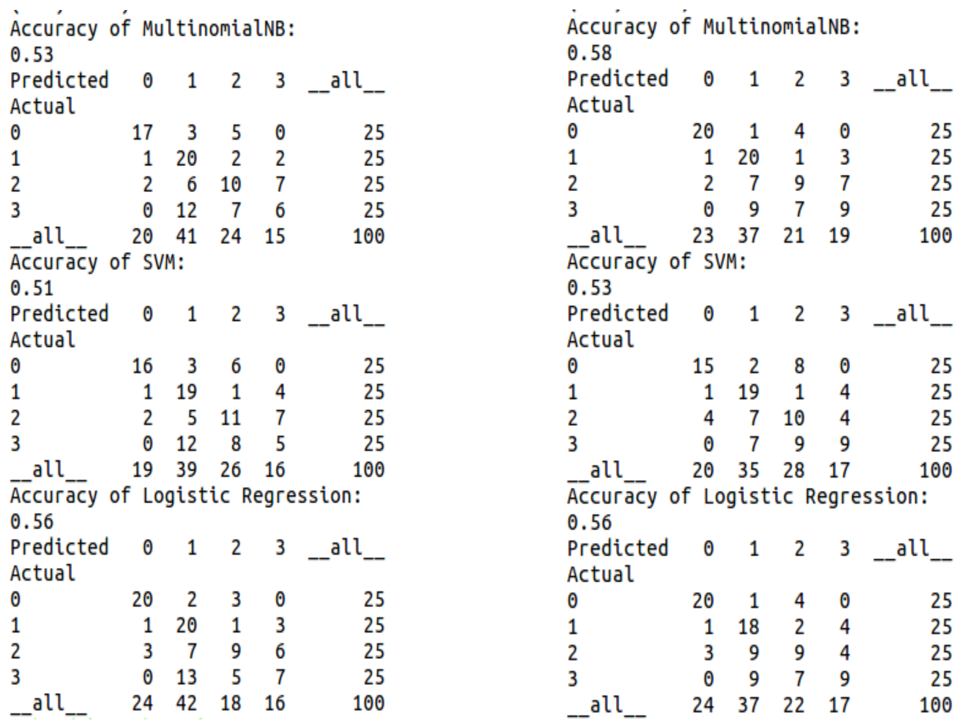
Fig 6.2 Lemmatizer + Regexp Tokenizer (left) /Word Tokenizer(right)-Unigram/TFIDF



Uni+Bigram-Tfidf-WordTokenizer			
	PorterStemmer	SnowballStemmer	WordNetLemmatizer
MultinomialNB	53	58	58
SVMClassifier	49	52	53
LRClassifier	50	55	56



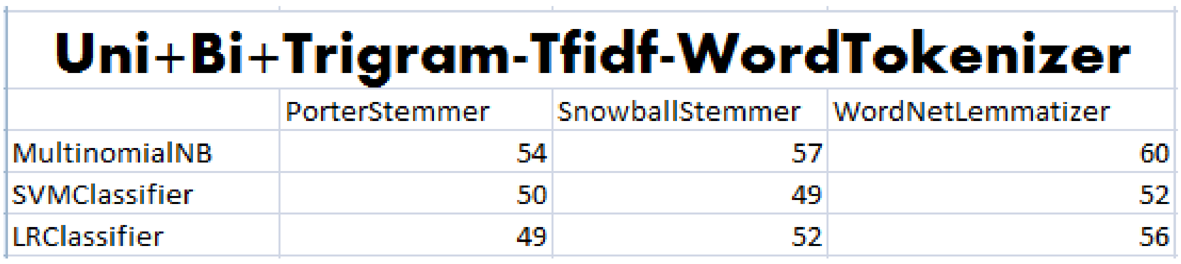
Uni+Bigram-Tfidf-RegexpTokenizer			
	PorterStemmer	SnowballStemmer	WordNetLemmatizer
MultinomialNB	49	57	53
SVMClassifier	48	53	51
LRClassifier	52	51	56



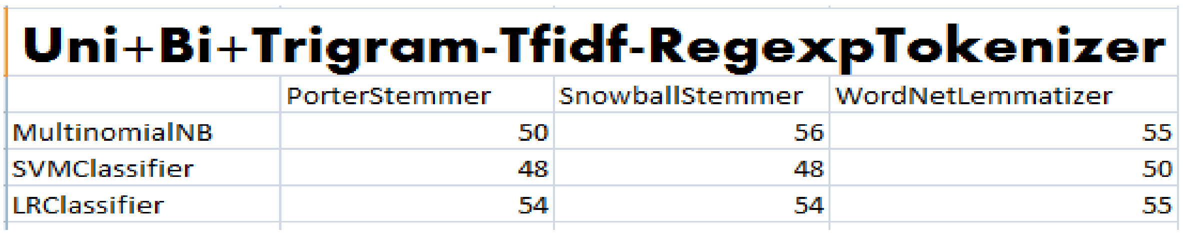
```
Accuracy of MultinomialNB:
0.53
Predicted    0   1   2   3  __all__
Actual
0           17   3   5   0       25
1            1  20   2   2       25
2            2   6  10   7       25
3            0  12   7   6       25
__all__     20  41  24  15      100
Accuracy of SVM:
0.51
Predicted    0   1   2   3  __all__
Actual
0           16   3   6   0       25
1            1  19   1   4       25
2            2   5  11   7       25
3            0  12   8   5       25
__all__     19  39  26  16      100
Accuracy of Logistic Regression:
0.56
Predicted    0   1   2   3  __all__
Actual
0           20   2   3   0       25
1            1  20   1   3       25
2            3   7   9   6       25
3            0  13   5   7       25
__all__     24  42  18  16      100
```

```
Accuracy of MultinomialNB:
0.58
Predicted    0   1   2   3  __all__
Actual
0           20   1   4   0       25
1            1  20   1   3       25
2            2   7   9   7       25
3            0   9   7   9       25
__all__     23  37  21  19      100
Accuracy of SVM:
0.53
Predicted    0   1   2   3  __all__
Actual
0           15   2   8   0       25
1            1  19   1   4       25
2            4   7  10   4       25
3            0   7   9   9       25
__all__     20  35  28  17      100
Accuracy of Logistic Regression:
0.56
Predicted    0   1   2   3  __all__
Actual
0           20   1   4   0       25
1            1  18   2   4       25
2            3   9   9   4       25
3            0   9   7   9       25
__all__     24  37  22  17      100
```

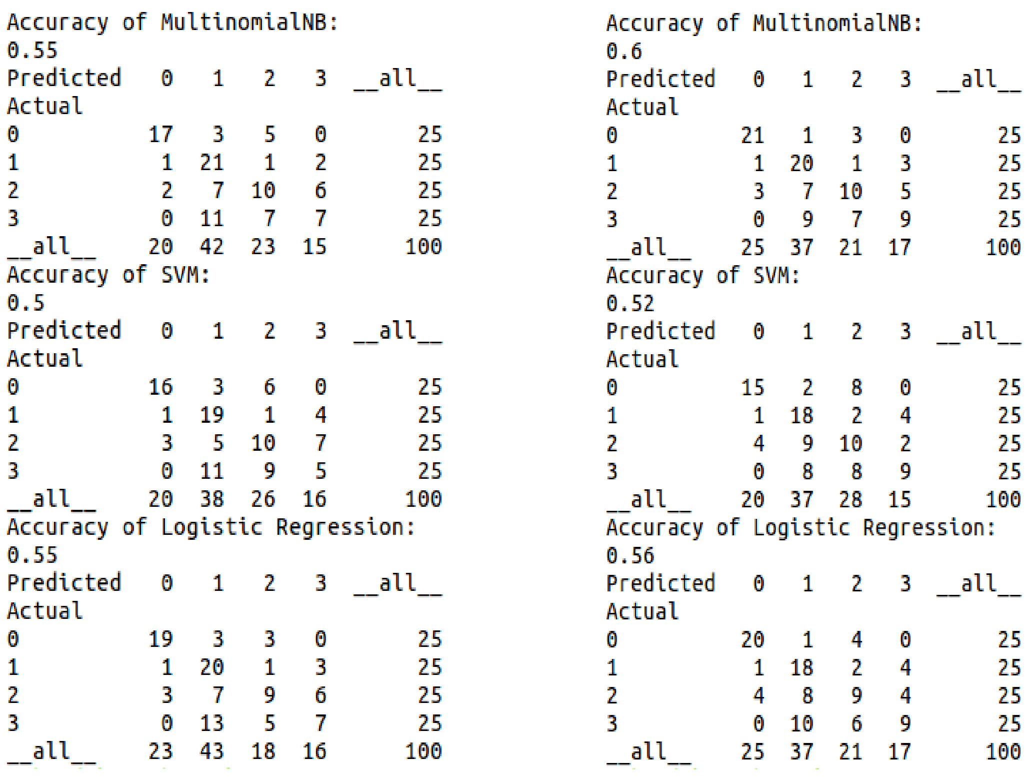
Fig 6.3 Lemmatizer + Regexp Tokenizer (left) /Word Tokenizer(right) – Uni+Bigram/TFIDF



Uni+Bi+Trigram-Tfidf-WordTokenizer			
	PorterStemmer	SnowballStemmer	WordNetLemmatizer
MultinomialNB	54	57	60
SVMClassifier	50	49	52
LRClassifier	49	52	56



Uni+Bi+Trigram-Tfidf-RegexpTokenizer			
	PorterStemmer	SnowballStemmer	WordNetLemmatizer
MultinomialNB	50	56	55
SVMClassifier	48	48	50
LRClassifier	54	54	55

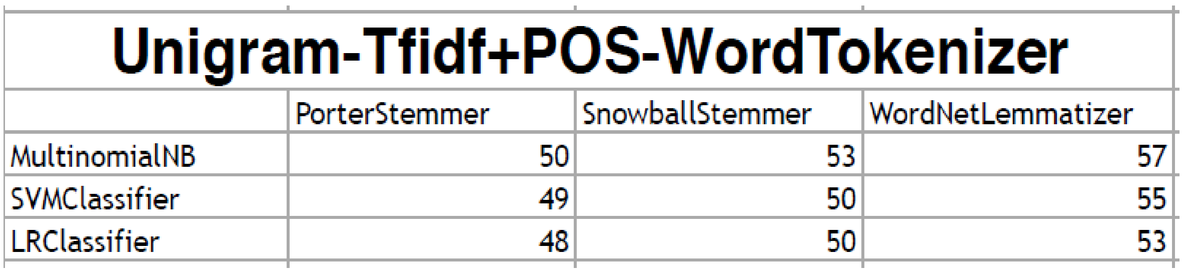


```
Accuracy of MultinomialNB:
0.55
Predicted   0   1   2   3  __all__
Actual
0          17   3   5   0       25
1           1  21   1   2       25
2           2   7  10   6       25
3           0  11   7   7       25
__all__    20  42  23  15      100
Accuracy of SVM:
0.5
Predicted   0   1   2   3  __all__
Actual
0          16   3   6   0       25
1           1  19   1   4       25
2           3   5  10   7       25
3           0  11   9   5       25
__all__    20  38  26  16      100
Accuracy of Logistic Regression:
0.55
Predicted   0   1   2   3  __all__
Actual
0          19   3   3   0       25
1           1  20   1   3       25
2           3   7   9   6       25
3           0  13   5   7       25
__all__    23  43  18  16      100
```

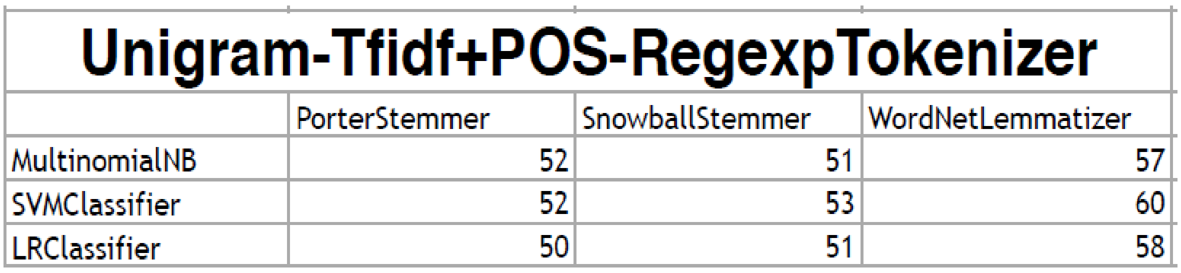
```
Accuracy of MultinomialNB:
0.6
Predicted   0   1   2   3  __all__
Actual
0          21   1   3   0       25
1           1  20   1   3       25
2           3   7  10   5       25
3           0   9   7   9       25
__all__    25  37  21  17      100
Accuracy of SVM:
0.52
Predicted   0   1   2   3  __all__
Actual
0          15   2   8   0       25
1           1  18   2   4       25
2           4   9  10   2       25
3           0   8   8   9       25
__all__    20  37  28  15      100
Accuracy of Logistic Regression:
0.56
Predicted   0   1   2   3  __all__
Actual
0          20   1   4   0       25
1           1  18   2   4       25
2           4   8   9   4       25
3           0  10   6   9       25
__all__    25  37  21  17      100
```

Fig 6.4 Lemmatizer + Regexp Tokenizer (left) /Word Tokenizer(right)-Uni+Bi+Trigram/TFIDF

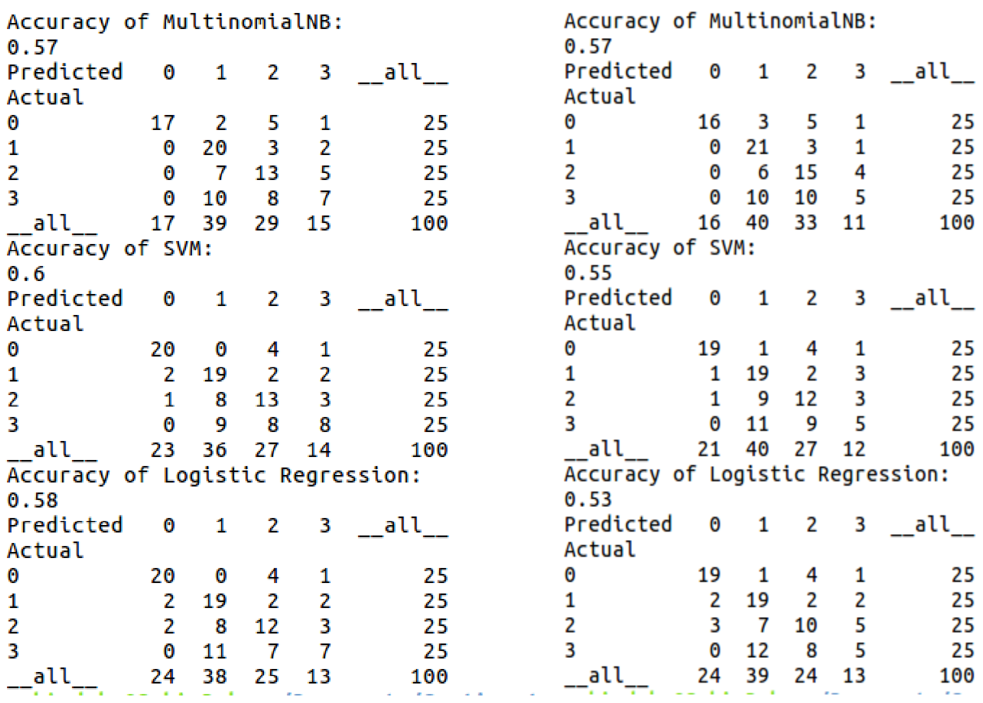
6.5 Based on TF-IDF + POS



Unigram-Tfidf+POS-WordTokenizer			
	PorterStemmer	SnowballStemmer	WordNetLemmatizer
MultinomialNB	50	53	57
SVMClassifier	49	50	55
LRClassifier	48	50	53



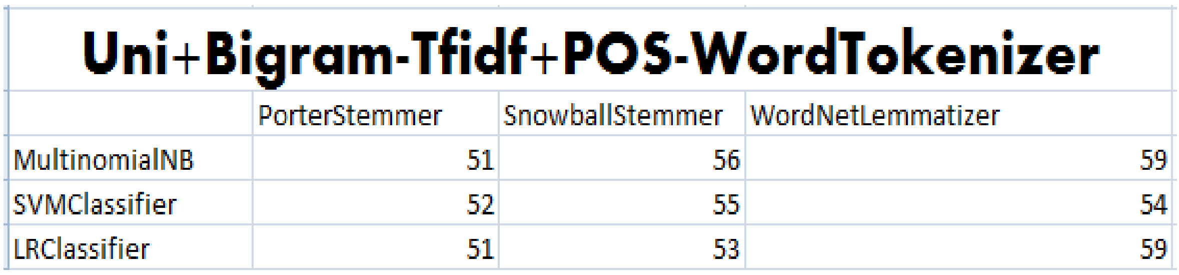
Unigram-Tfidf+POS-RegexpTokenizer			
	PorterStemmer	SnowballStemmer	WordNetLemmatizer
MultinomialNB	52	51	57
SVMClassifier	52	53	60
LRClassifier	50	51	58



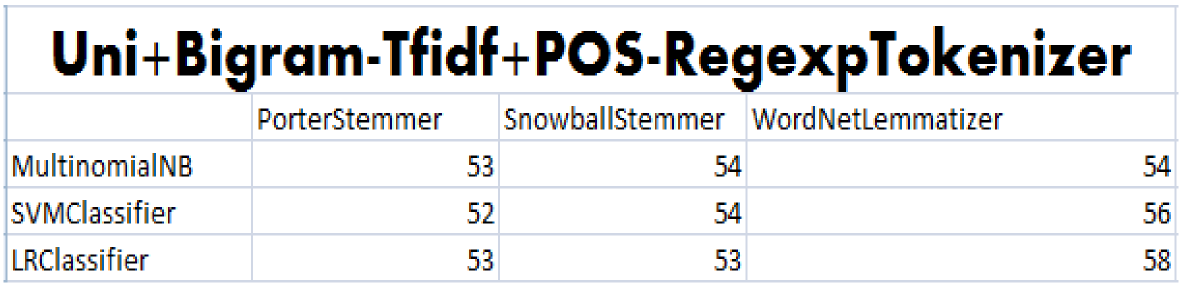
```
Accuracy of MultinomialNB:
0.57
Predicted    0   1   2   3  __all__
Actual
0           17   2   5   1       25
1            0  20   3   2       25
2            0   7  13   5       25
3            0  10   8   7       25
__all__     17  39  29  15      100
Accuracy of SVM:
0.6
Predicted    0   1   2   3  __all__
Actual
0           20   0   4   1       25
1            2  19   2   2       25
2            1   8  13   3       25
3            0   9   8   8       25
__all__     23  36  27  14      100
Accuracy of Logistic Regression:
0.58
Predicted    0   1   2   3  __all__
Actual
0           20   0   4   1       25
1            2  19   2   2       25
2            2   8  12   3       25
3            0  11   7   7       25
__all__     24  38  25  13      100
```

```
Accuracy of MultinomialNB:
0.57
Predicted    0   1   2   3  __all__
Actual
0           16   3   5   1       25
1            0  21   3   1       25
2            0   6  15   4       25
3            0  10  10   5       25
__all__     16  40  33  11      100
Accuracy of SVM:
0.55
Predicted    0   1   2   3  __all__
Actual
0           19   1   4   1       25
1            1  19   2   3       25
2            1   9  12   3       25
3            0  11   9   5       25
__all__     21  40  27  12      100
Accuracy of Logistic Regression:
0.53
Predicted    0   1   2   3  __all__
Actual
0           19   1   4   1       25
1            2  19   2   2       25
2            3   7  10   5       25
3            0  12   8   5       25
__all__     24  39  24  13      100
```

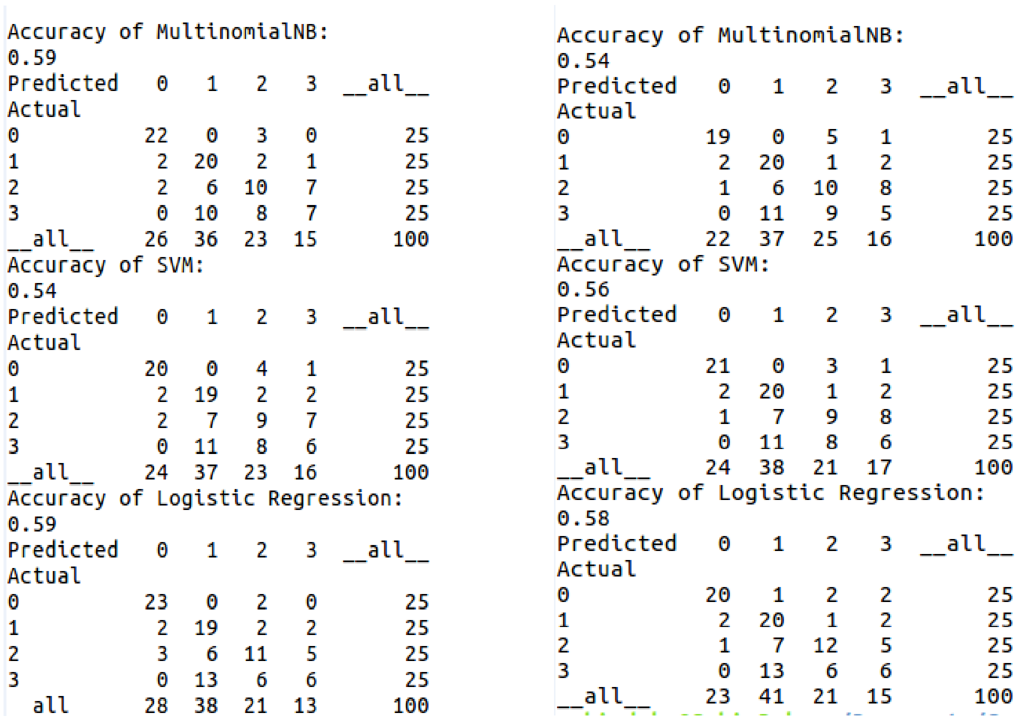
Fig 6.5 Lemmatizer + Regexp Tokenizer (left) /Word Tokenizer(right)-Unigram/TFIDF+POS



Uni+Bigram-Tfidf+POS-WordTokenizer			
	PorterStemmer	SnowballStemmer	WordNetLemmatizer
MultinomialNB	51	56	59
SVMClassifier	52	55	54
LRClassifier	51	53	59



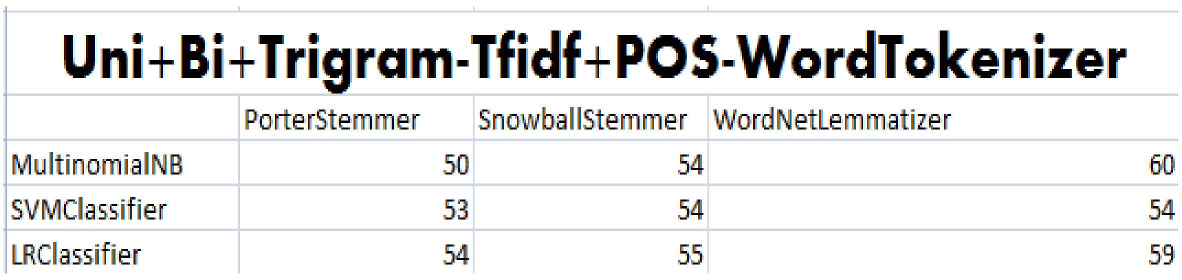
Uni+Bigram-Tfidf+POS-RegexpTokenizer			
	PorterStemmer	SnowballStemmer	WordNetLemmatizer
MultinomialNB	53	54	54
SVMClassifier	52	54	56
LRClassifier	53	53	58



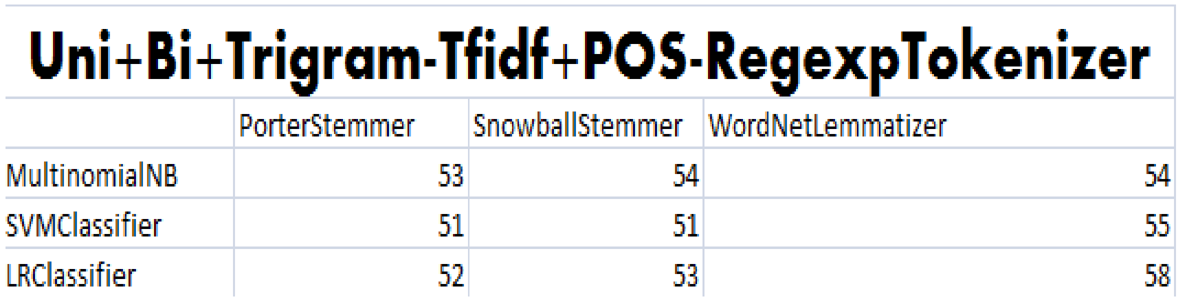
```
Accuracy of MultinomialNB:
0.59
Predicted   0   1   2   3   __all__
Actual
0          22   0   3   0        25
1           2  20   2   1        25
2           2   6  10   7        25
3           0  10   8   7        25
__all__    26  36  23  15       100
Accuracy of SVM:
0.54
Predicted   0   1   2   3   __all__
Actual
0          20   0   4   1        25
1           2  19   2   2        25
2           2   7   9   7        25
3           0  11   8   6        25
__all__    24  37  23  16       100
Accuracy of Logistic Regression:
0.59
Predicted   0   1   2   3   __all__
Actual
0          23   0   2   0        25
1           2  19   2   2        25
2           3   6  11   5        25
3           0  13   6   6        25
  all      28  38  21  13       100
```

```
Accuracy of MultinomialNB:
0.54
Predicted    0    1    2    3   __all__
Actual
0           19    0    5    1        25
1            2   20    1    2        25
2            1    6   10    8        25
3            0   11    9    5        25
__all__     22   37   25   16       100
Accuracy of SVM:
0.56
Predicted    0    1    2    3   __all__
Actual
0           21    0    3    1        25
1            2   20    1    2        25
2            1    7    9    8        25
3            0   11    8    6        25
__all__     24   38   21   17       100
Accuracy of Logistic Regression:
0.58
Predicted    0    1    2    3   __all__
Actual
0           20    1    2    2        25
1            2   20    1    2        25
2            1    7   12    5        25
3            0   13    6    6        25
__all__     23   41   21   15       100
```

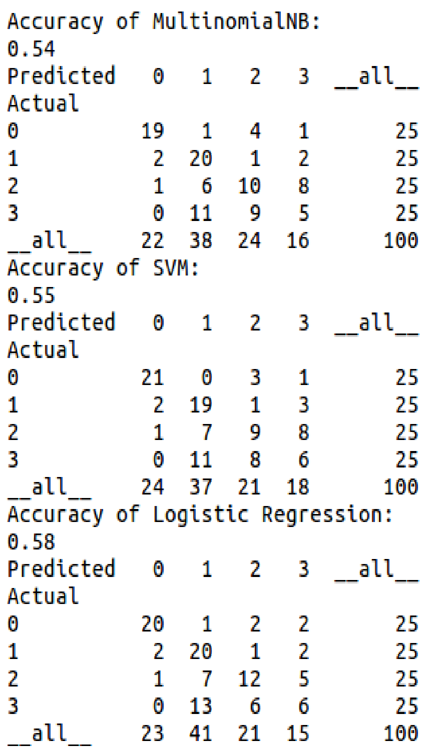
Fig 6.6 Lemmatizer + Regexp Tokenizer (left) /Word Tokenizer(right)-Uni+Bigram/TFIDF+POS



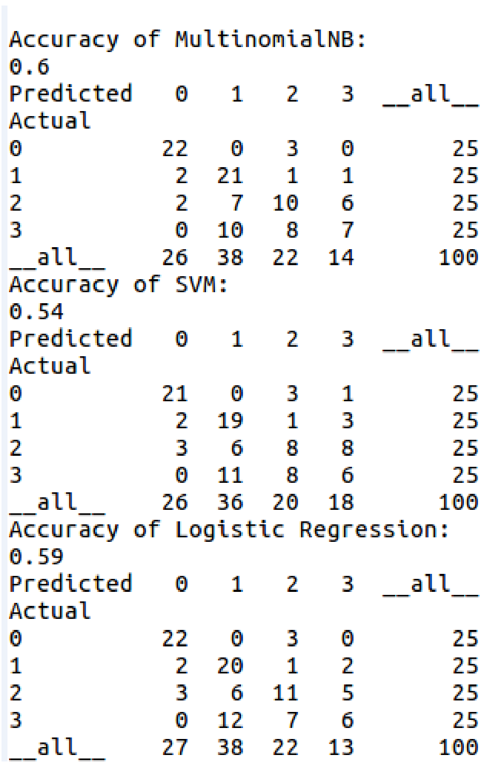
Uni+Bi+Trigram-Tfidf+POS-WordTokenizer			
	PorterStemmer	SnowballStemmer	WordNetLemmatizer
MultinomialNB	50	54	60
SVMClassifier	53	54	54
LRClassifier	54	55	59



Uni+Bi+Trigram-Tfidf+POS-RegexpTokenizer			
	PorterStemmer	SnowballStemmer	WordNetLemmatizer
MultinomialNB	53	54	54
SVMClassifier	51	51	55
LRClassifier	52	53	58



```
Accuracy of MultinomialNB:
0.54
Predicted   0   1   2   3  __all__
Actual
0          19   1   4   1       25
1           2  20   1   2       25
2           1   6  10   8       25
3           0  11   9   5       25
__all__    22  38  24  16      100
Accuracy of SVM:
0.55
Predicted   0   1   2   3  __all__
Actual
0          21   0   3   1       25
1           2  19   1   3       25
2           1   7   9   8       25
3           0  11   8   6       25
__all__    24  37  21  18      100
Accuracy of Logistic Regression:
0.58
Predicted   0   1   2   3  __all__
Actual
0          20   1   2   2       25
1           2  20   1   2       25
2           1   7  12   5       25
3           0  13   6   6       25
__all__    23  41  21  15      100
```



```
Accuracy of MultinomialNB:
0.6
Predicted   0   1   2   3  __all__
Actual
0          22   0   3   0       25
1           2  21   1   1       25
2           2   7  10   6       25
3           0  10   8   7       25
__all__    26  38  22  14      100
Accuracy of SVM:
0.54
Predicted   0   1   2   3  __all__
Actual
0          21   0   3   1       25
1           2  19   1   3       25
2           3   6   8   8       25
3           0  11   8   6       25
__all__    26  36  20  18      100
Accuracy of Logistic Regression:
0.59
Predicted   0   1   2   3  __all__
Actual
0          22   0   3   0       25
1           2  20   1   2       25
2           3   6  11   5       25
3           0  12   7   6       25
__all__    27  38  22  13      100
```

Fig 6.7 Lemmatizer + Regexp Tokenizer (left) /Word Tokenizer(right)-Uni+Bi+Trigram/TFIDF+POS

7.CONCLUSION

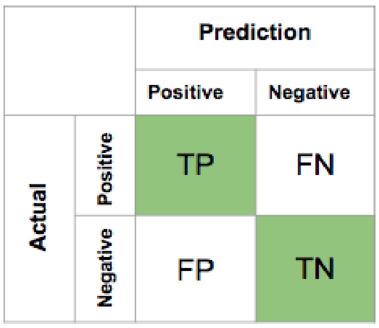
Explanation of Confusion Matrix

0 - Angry

1 - Happy

2 - Sad

3 - Relaxed



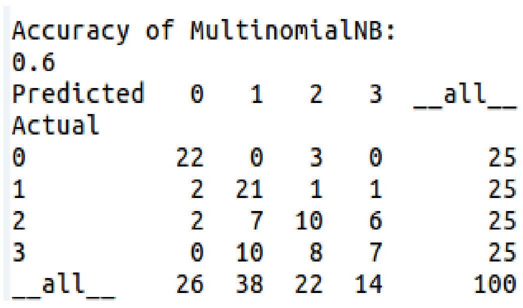
		Prediction	
		Positive	Negative
Actual	Positive	TP	FN
	Negative	FP	TN

Fig 7.1 Confusion Matrix

Max Accuracy of **60%** obtained in

Uni+Bi+Trigram + Parts of Speech Tagging(POS) + WordTokenizer + WordNetLemmatizer

Multinomial Naïve Bayes Classifier



```
Accuracy of MultinomialNB:
0.6
Predicted    0   1   2   3  __all__
Actual
0           22   0   3   0       25
1            2  21   1   1       25
2            2   7  10   6       25
3            0  10   8   7       25
__all__     26  38  22  14      100
```

Fig 7.2 Final Confusion Matrix for the dataset

Accuracy of respective moods:

1.	**Angry**	**: 88%**
2.	**Happy**	**: 84%**
3.	**Sad**	**: 40%**
4.	**Relaxed**	**: 28%**

8. REFERENCES

1. Besson, Mireille, et al. "Singing in the brain: Independence of lyrics and tunes." *Psychological Science* 9.6 (1998): 494-498.

2. Van Zaanen, Menno, and Pieter Kanters. "Automatic Mood Classification Using TF* IDF Based on Lyrics." *ISMIR*. 2010..

3. Raschka, Sebastian. "MusicMood: Predicting the mood of music from song lyrics using machine learning." *arXiv preprint arXiv:1611.00138* (2016).

4. Mood classification of songs based on lyrics - Francesco Cucari , Linköping University

5. Porter Stemmer: https://tartarus.org/martin/PorterStemmer/def.txt

6. Snowball Stemmer: http://snowball.tartarus.org/

7. http://scikit-learn.org/stable/modules/multiclass.html

8. http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

9. http://scikit-learn.org/stable/tutorial/statistical_inference/supervised_learning.html

10. https://www.nltk.org/

11. https://docs.python.org/3/howto/regex.html