UNIVERSITY OF NAIROBI

College of Biological and Physical Science

School of Computing and Informatics

COMPARATIVE TWEETER SENTIMENT ALGORITHMS BASED ON PROBABILISTIC AND LINEAR CLASSIFIERS:

By

KIPLAGAT WILFRED KIPRONO

REG.NO. P53/65597/2013

SUPERVISOR: DR. ELISHA A. ABADE

Project report submitted in partial fulfillment of the requirements for the award of a degree in Master of Science in Distributed Computing Technology

DECLARATION

The project in this report is my original work and has not been presented for any other university award.

SIGNATURE _____ DATE _____

Name: KIPLAGAT WILFRED KIPRONO

REG/NO P53/65597/2013

This research report has been submitted in partial fulfillment for the requirement of the Degree in Master of Science in Distributed Computing Technologies of the University of Nairobi with my approval as the university Supervisor.

Signature _____ Date _____

Dr. Elisha A. ABADE
Lecturer
School of Computing and Informatics

ABSTRACT

The transition from web 1.0 to web 2.0 has enabled direct interaction between users and its environment such as social media networks. In this research paper we have analyzed algorithms for sentiment analysis which can be used to utilize this huge information. The goals of this research is to device a way of obtaining social network opinions, extracting features from unstructured text and assign for each feature its associated sentiment in a clear and efficient way. In this project we have applied naïve bayes, support vector machines and maximum entropy for analysis and produced an analytical report of the three qualitatively and quantitatively. We performed the project empirically and analyzed the resulting data using an excel tool so as to obtain comparative analysis of the three algorithms for classification.

DEDICATION

I dedicate my masters of research project to my wife Everline Kosiom, son Brian Rono, and daughter Stacy Rono who supported me unconditionally throughout the course of my studies. To my parents Mrs. Sarah Chelagat and late John Arapkechem who gave me immeasurable support and sacrifices to enable us have an opportunity to pursue our dreams. Only god can truly know the depth of my gratitude.

ACKNOWLEDGEMENTS

The research project process requires a lot of effort and assistance from other people. First and foremost I could like to thank my supervisor DR.Elisha Abade. I would like to express my deepest gratitude to my lecturers and panelists to mention but a few Dr.Andrew Kahonge, Prof.Ayienga and Prof.Omwenga.I would also like thank my fellow colleagues at the school computing and Informatics University of Nairobi for their support and cooperation towards my project. Finally I would like to thank my family and the almighty god for all the opportunities, strength and support offered in the research process.

TABLE OF CONTENTS

LIST OF ABREVIATIONS

POS **:** Parts Of Speech

NB **:** Naïve Bayes

MAX ENT **:** Maximum Entropy

SVM : Support Vector Machines

HTML **:** Hypertext Markup Language

API **:** Application Programming Interface

SQL : Structured Query Language

AMQP : Advanced message queuing protocol

LIST OF FIGURES

CHAPTER ONE:

INTRODUCTION

1.0 Background

The proliferation of web-enabled devices, including desktops, laptops, tablets, and mobile phones, enables people to communicate, participate and collaborate with each other in various web communities, forums, social networks and blogs. Companies and businesses use technology for sales and marketing and the consumers now search for opinions online before, during, and after a purchase. The next step for brands is finding out whether people are talking positively or negatively about their brand, and why. Some online ratings provide a number but not the reasoning behind it, and may only present half of the story, Diana (2011).Facebook and twitter actually welcome and encourage users to support causes for political and/or social change. Many times social media is a voice that provides too much information which the people crave.

Ignoring these opinions on social media and internet use by consumers and not taking the time to understand how adversarial forces are using it to further their causes will not decrease its effectiveness and use. U.S. president barrack Obama has eight million followers and uses his account to update followers on his daily activities and thoughts. David Carr 2008.

The same way that governments and politicians use social media to spread their influence, communicate to supporters, and fundraise, business companies also use social media for the same purposes. Social media applications are a triple-edged sword that can create addictive information-seeking behaviors that break down social-norm behaviors of its users, encourage users to generate and report information that normally would have been kept private, and ultimately provide users with increased access to information that could be used to manipulate the user's perception of a product and the user's environment. Such postings have also mobilized consumers to defect a product or service offered by companies.

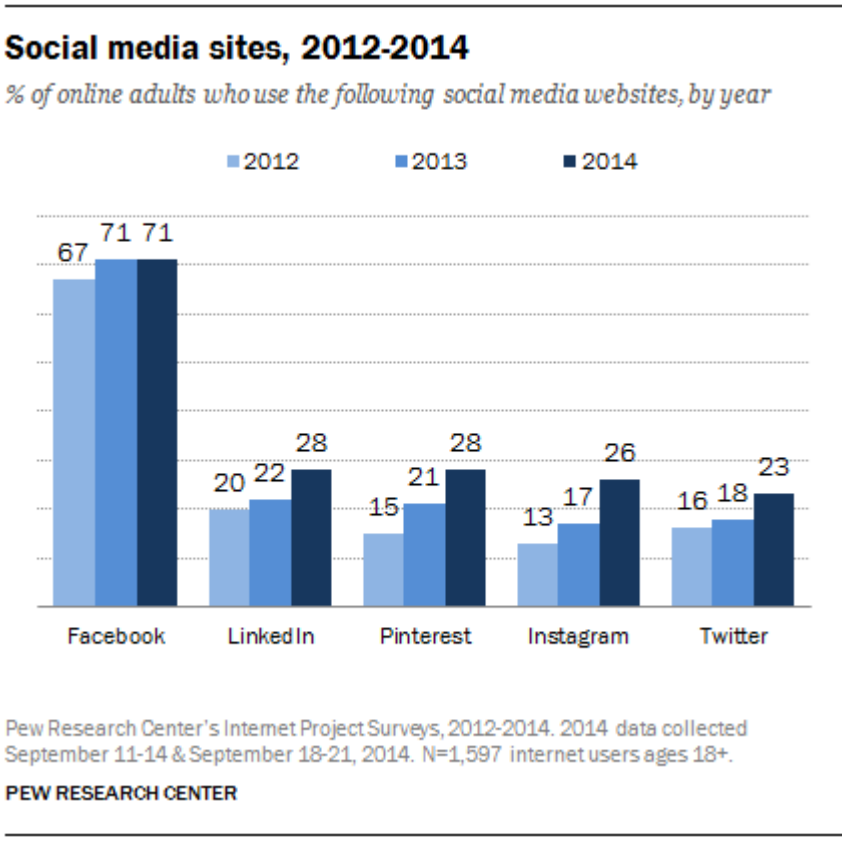


Figure no.1 pew research centers internet surveys 2014

1.1 Problem statement

The vast amount of information posted in the social media has not been utilized by institutions for intelligence purposes on the products and services they offer. They have not employed a mechanism that can effectively discover market intelligence for supporting decision makers by establishing a monitoring system to track external opinions on different aspects of a business in realtime. The challenge is caused by huge amounts of data available, web data is unstructured semi structured and heterogeneous and information about the same product is spread over a large number of websites (Paschke etal 2013). Thus the need to automate this process arises and sentiment analysis is the answer to this need. There is also a challenge for the organizations to acquire evidence accurately from the mass data available for evidence.

1.2 Purpose

The main aim of performing this research is to accurately determine the attitude on opinions by users of social network sites and their relationship to customer preference on products and services to enable organizations improve on their performance.

1.3 Objective of the study

The study aims at enriching the knowledge and understanding on sentiment analysis .specifically the main objectives are:

- To develop a technique to be used by Kenyan businesses to proactively harvest and store tweeter data on the social network.
- To analyze the machine learning algorithms used and evaluate their suitability for classifying tweeter data for sentiment analysis.
- Develop classifiers models using the algorithms adopted in the above to extract features that allow them to classify opinions into the negative, positive or neutral.
- Compare the resulting data and recommend the algorithm with accurate results based on a topic.

1.4 Research questions

In this study the main questions to answer are:

- Do Kenyan businesses have a technique for harvesting data on social networks?
- Is there any analysis technique used to generate the data?
- Are there any classifier for these data generated from social networks use to curb the vice?
- Which technique available can generate an accurate result so that it can be adopted for use by businesses in Kenya?

1.5 Significance of study

The significance of this project research will contribute towards ensuring that business institutions transcend from simple document retrieval to useful knowledge discovery from the

huge amount of textual data in order to assist in identifying comments on product and services for improvement and value addition to enable them to be competitive.

1.6 assumptions and limitations

- The sentiment analysis process was not able to recognize sarcastic opinions or those reported ironically
- The project was only handling opinions based in the English and Kiswahili languages.

1.7 Scope

In this study we intend to undertake this task by considering various views and preferences of consumers from a telecommunications company in Kenya. The research was only seeking to extract opinions from tweets to the company based on their tariffs, data bundles and any other comment which may be negative or positive to the company products,

1.8 Research justification

- Accurately and precisely organize feedback on the social media sites for decision making by business institutions.
- Incorporate all Kenyan languages (English and Kiswahili).

CHAPTER TWO.

LITERATURE REVIEW

2.0 Social Networks

Social networking site refers to web-based tools and services that allow users to create and share content and information. These tools are ‘social’ in the sense that they are created in ways that enable users to share and communicate with one another (bohler-muller & merwe, 2011. Facebook started in 2004 and now connects over five hundred million users worldwide. Approximately three hundred and fifty million of these users are outside the United States; two hundred and fifty million users access facebook from their mobile devices in sixty countries. YouTube was founded in 2005, and today users from around the world upload over thirty-five hours of video every minute. Twitter was founded in 2006, and an organization or person with a twitter account can immediately connect to millions of followers, sending short messages of one hundred and forty characters or less (called tweets). This service rapidly gained worldwide popularity, with more than 100 million users who in 2012 posted 340 million tweets per day. The service also handled 1.6 billion search queries per day. In 2013 twitter was one of the ten most-visited websites, and has been described as "the sms of the internet. As of may 2015, twitter has more than 500 million users, out of which more than 302 million are active users. With the use of smart cell phones and other mobile computing and internet-capable devices, people have the ability to access online content and send/receive instant messages anytime and anywhere there is an internet connection or cell phone signal.

2.1 Machine learning

Machine learning focuses on the development of computer programs that can teach themselves to grow and change when exposed to new data. The process of machine learning is similar to that of data mining. The documents can be classified by three ways, un-supervised, supervised and semi supervised methods. Many techniques and algorithms have been proposed recently for the clustering and classification of electronic documents. The

automatic classification of documents into predefined categories has observed as an active attention, as the internet usage rate has quickly enlarged.

The task of automatic text classification have been extensively studied and rapid progress seems in this area, including the machine learning approaches such as bayesian classifier, decision tree, k-nearest neighbor(KNN), support vector machines(SVMS), neural networks, latent semantic analysis, rocchio's algorithm, fuzzy correlation and genetic algorithms. Normally supervised learning techniques are used for automatic text classification, where pre-defined category labels are assigned to documents based on the likelihood suggested by a training set of labeled documents.

2.1.1 Supervised learning

Supervised learning is the type of machine learning that takes place when the correct output results (or target variables) for the training instances to be input are known. The objective of training a machine learning algorithm is to find the model (that is, a rule or function) that maps the inputs into the known output values (hidden markov 2008). Once the learning process is complete and we have a workable model, it can be applied to new input data to predict the expected output where, unlike the training dataset, the target value are not known in advance. In supervised learning the variables under investigation can be split into two groups: explanatory variables and dependent variables. The values of the dependent variable must be known for a sufficiently large part of the dataset. Supervised learning is also the most common technique for training neural networks and decision trees:

2.2.2 Unsupervised learning

The model is not provided with the "correct results" for a dataset on which to train. Since unlabeled examples are given to the learner, there is no feedback - neither error nor reward - to evaluate a potential solution. The goal is to have the computer learn how to do something even though we don't explicitly tell it how to accomplish that task. In unsupervised learning situations, all variables are treated in the same way. There is no distinction between explanatory variables and dependent variables. However, there is still some objective to achieve, which might be a general objective, such as data reduction, or a more specific goal like finding clusters.

2.2.3 Reinforcement learning

Reinforcement learning is concerned with how an agent ought to take actions in an environment so as to maximize some notion of long-term reward. Reinforcement learning algorithms attempt to find a policy that maps states of the world to the actions the agent ought to take in those states. Reinforcement learning differs from the supervised learning problem in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected.

2.2 Text Classification Algorithms

Classification is a data mining (machine learning) technique used to predict group membership for data instances. For example, you may wish to use classification to predict whether the weather on a particular day will be “sunny”, “rainy” or “cloudy”. Popular classification techniques include decision trees and neural networks.

2.2.1 Clustering

In this type of learning, the goal is to find similarities in the training data and to partition the dataset into subsets that are demarcated by these similarities. The expectation that the most significant clusters discovered by these data-driven procedures are consistent with our intuitive classification is often, but not always, satisfied hu and liu (2004). Although the clustering algorithm won't assign appropriate names to these clusters, it can produce them and then use them to anticipate similarities expected in new examples by classifying them into the most appropriate cluster. This data-driven approach can work well when sufficient data is available. For instance, social information filtering algorithms, such as those used by amazon.com to recommend books, are based on finding similar groups of people and then assigning new users to these groups for the purpose of making recommendations. K-means is one of the simplest unsupervised clustering algorithms that solve well known clustering problems. The k means algorithm clusters data by trying to separate samples into n groups of equal variance, minimizing the "inertia" or "within-cluster sum-of-squares" criterion. This algorithm requires the number of clusters to be specified. K-means can scale to a large number of samples and has been used in a wide range of application areas across many different fields.

2.2.2 Decision tree learning

Decision tree learning is the construction of a decision tree from class-labeled training tuples (b. Pang and l. Lee 2008). The decision tree rebuilds the manual categorization of training documents by constructing well-defined true/false-queries in the form of a tree structure. In a decision tree structure, leaves represent the corresponding category of documents and branches represent conjunctions of features that lead to those categories. The well-organized decision tree can easily classify a document by putting it in the root node of the tree and let it run through the query structure until it reaches a certain leaf, which represents the goal for the classification of the document. The main advantage of decision tree is its simplicity in understanding and interpreting, even for non-expert users. However, when there are a small number of structured attributes, the performance, simplicity and understandability of decision trees for content-based models are all advantages. The major risk of implementing a decision tree is that it over fits the training data with the occurrence of an alternative tree that categorizes the training data worse but would categorize the documents to be categorized better.

2.2.3 Decision rules classification

Decision rules classification method uses the rule-based inference to classify documents to their annotated categories. The algorithms construct a rule set that describe the profile for each category. Rules are typically constructed in the format of "if condition then conclusion", where the condition portion is filled by features of the category, and the conclusion portion is represented with the category's name or another rule to be tested. The rule set for a particular category is then constructed by combining every separate rule from the same category with logical operator, typically use "and" and "or". During the classification tasks, not necessarily every rule in the rule set needs to be satisfied. In the case of handling a dataset with large number of features for each category, heuristics implementation is recommended to reduce the size of rules set without affecting the performance of the classification. . Besides, the learning and updating of decision rule methods need extensive involvement of human experts to construct or update the rule sets.

2.2.4 Artificial neural networks

A neural network usually involves a large number of processors operating in parallel, each with its own small sphere of knowledge and access to data in its local memory. Typically, a neural network is initially "trained" or fed large amounts of data and rules about data relationships .a program can then tell the network how to behave in response to an external stimulus (for example, to input from a computer user who is interacting with the network) or can initiate activity on its own (within the limits of its access to the external world) (mukkamala, 2003).

Neural networks, whose elementary structures are far more complicated than the mathematical models used for artificial neural networks. The dependent variable, y, is the target variable that we are trying to understand, classify or generalize. The vector x is composed of the input variables, x_1, x_2, x_3 etc., that are used for that task. Neural networks have been used both in anomaly intrusion detection as well as in misuse intrusion detection. In the first approach of neural networks (Debar, 1992) for intrusion detection, the system learns to predict the next command based on a sequence of previous commands by a user.

2.2.5 Fuzzy correlation

Fuzzy correlation deals with fuzzy information or in-complete data, and also converts the property value into fuzzy sets for multiple document classification. The challenges of multiclass text categorization using one-against-one fuzzy support vector machine with Reuter's news as the example data, and shows better results using one-against-one fuzzy support vector machine as a new technique when compare with one-against-one support vector machine. Presented the improvement of decision rule and design a new algorithm of F-K-NN (fuzzy K-NN) to improve categorization performance when the class distribution is uneven, and show that the new method is more effective.

2.2.6 Genetic algorithm

Genetic algorithm finds optimum characteristic parameters using the mechanisms of genetic evolution and survival of the fittest in natural selection. Genetic algorithms make it possible to remove misleading judgments in the algorithms and improve the accuracy of document classification. This is an adaptive probability global optimization algorithm, which simulated

in a natural environment of biological and genetic evolution, and is widely used for their simplicity and strength. Several researchers have used this method for the improvement of the text classification process. In the experimental analysis, they show that the improved method is feasible and effective for text classification.

2.3 sentiment analysis

Sentiment analysis is a machine learning approach in which machines analyze and classify the human's sentiments, emotions, opinions etc about some topic which are expressed in the form of either text or speech. The textual data available in the web is increasing day by day. In order to enhance the sales of a product and to improve the customer satisfaction, most of the on-line shopping sites provide the opportunity to customers to write reviews about products. These reviews are large in number and to mine the overall sentiment or opinion polarity from all of them, sentiment analysis can be used. Manual analysis of such large number of reviews is practically impossible. Therefore automated approach of a machine has significant role in solving this hard problem.

2.3.1 Feature extraction in sentiment analysis

Since most of sentiment analysis approaches use or depend on machine learning techniques, the salient features of text or documents are represented as feature vector. The following are the features used in sentiment analysis.

Term presence or term frequency: in standard information retrieval and text classification, term frequency is preferred over term presence. However, pang et al. (2002), in sentiment analysis for movie reviews, show that this is not the case in sentiment analysis. Pang et al. Claim that this is one indicator that sentiment analysis is different from standard text classification where term frequency is taken to b e a go o d indicator of a topic. Ironically, another study by yang et al. (2006) shows that words that app ear only once in a given corpus are good indicators of high-precision subjectivity. Term can be unigrams, bi-grams or other higher-order n-grams. Whether unigrams or higher-order n-grams give better results is not clear. Pang et al.(2002) claim that unigrams outperform bi-grams in movie review sentiment analysis, but Dave et al. (2003) report that bigrams and trigrams give better product-review polarity classification.

Pos (part of speech) tags:

Parts of speech are used to disambiguate sense which in turn is used to guide feature selection (pang and lee, 2008). For example, with pos tags, we can identify adjectives and adverbs which are usually used as sentiment indicators (Turney, 2002). Turney himself found that adjectives performed worse than the same number of unigrams selected on the basis of frequency.

Syntax and negation:

Collocations and other syntactic features can be employed to enhance performance. In some short text (sentence-level) classification tasks, algorithms using syntactic features and algorithms using n-gram features were found to give same performance (pang and lee, 2008).negation is also an important feature to take into account since it has the potential of reversing a sentiment (pang and lee, 2008).There are attempts to model negation for better performance (Das and Chen, 2001, Na et al., 2004).Na et al. (2004) report 3% accuracy improvement for electronics product reviews by handling negation. Note also that negation can b e expressed in more subtle ways such as sarcasm, irony and other polarity reversers.

2.4 Text classification techniques

2.4.0 Support vector machine (SVM)

Support vector machines (SVMS) are one of the discriminative classification methods which are commonly recognized to be more accurate. The SVM classification method is based on the structural risk minimization principle from computational learning theory. The idea of this principle is to find a hypothesis to guarantee the lowest true error. Besides, the SVM are well-founded that very open to theoretical understanding and analysis. The SVM need both positive and negative training set which are uncommon for other classification methods. These positive and negative training set are needed for the SVM to seek for the decision surface that best separates the positive from the negative data in the n-dimensional space. The document representatives which are closest to the decision surface are called the support vector. The performance of the SVM classification remains unchanged if documents that do not belong to the support vectors are removed from the set of training data. The SVM classification method is outstanding from the others with its outstanding classification

effectiveness. However, the major drawback of the SVM is their relatively complex training and categorizing algorithms and also the high time and memory consumptions during training stage and classifying stage. Besides, confusions occur during the classification tasks due to the documents could be a notated to several categories because of the similarity is typically calculated individually for each category.

2.4.1 Naïve bayes algorithm

Naïve bayes classifier is a simple probabilistic classifier based on applying bayes' theorem with strong independence assumptions. A more descriptive term for the underlying probability model would be independent feature model. These independence assumptions of features make the features order is irrelevant and consequently that the presence of one feature does not affect other features in classification tasks. These assumptions make the computation of bayesian classification approach more efficient, but this assumption severely limits its applicability. Depending on the precise nature of the probability model, the naïve bayes classifiers can be trained very efficiently by requiring a relatively small amount of training data to estimate the parameters necessary for classification. Because independent variables are assumed, only the variances of the variables for each class need to be determined and not the entire covariance matrix.

Maximum entropy

The maxent classifier (known as a conditional exponential classifier) converts labeled feature sets to vectors using encoding. This encoded vector is then used to calculate weights for each feature that can then be combined to determine the most likely label for a feature set. Kaufmann Etal (2012) argued that a classifier is parameterized by a set of x {weights}, which is used to combine the joint features that are generated from a feature-set by an x {encoding}. In particular, the encoding maps each c {(feature set, label)} pair to a vector. The probability of each label is then computed using the following equation:

$$P(fs \backslash label) = \frac{dot\ prod\ (weights,\ encode\ (fs,\ label))}{Sum\ (dot\ prod\ (weights, encode(fs,l))forlinlabels)}$$

2.4 Previous Sentiment Analysis Related Research

Kuat yessenov and sasa misailovic **(**2009) analyzed the sentiment of social network comments on articles from digg as text corpora. He evaluated the fitness of different feature selection and learning algorithms (supervised and unsupervised) on the classification of comments according to their subjectivity (subjective/objective) and their polarity (positive/negative). The results showed that simple bag-of-words model can perform relatively well, and it can be further refined by the choice of features based on syntactic and semantic information from the text.

Bo pang and lillian lee (2008) classified documents not by topic, but by overall sentiment, e.g., determining whether a review is positive or negative. Using movie reviews as data, the results produced via machine learning techniques are quite good in comparison to the human generated baselines. In terms of relative performance they reported that, naive bayes tends to do the worst and svms tend to do the best, although the differences aren't very large. On the other hand, they were not able to achieve accuracies on the sentiment classification problem comparable to those reported for standard topic based categorization, despite the several different types of features they tried. Unigram presence information turned out to be the most effective; in fact, none of the alternative features they employed provided consistently better performance once unigram presence was incorporated. Interestingly, though, the superiority of presence information in comparison to frequency information in their setting contradicted previous observations made in topic classification work (Mccallumand Nigam, 1998).

Barbosa and Feng (2010) used a two-phased approach to twitter sentiment analysis. The two phases are: 1) classifying the dataset into objective and subjective classes (subjectivity detection) and 2) classifying subjective sentences into positive and negative classes (polarity detection). Suspecting that the use of n-grams for twitter sentiment analysis might not be a good strategy since twitter messages are short, they use two other features of tweets: Meta information about tweets and syntax of tweets. For meta-info, they use pos tags (some tags are likely to show sentiment, e.g. Adjectives and interjections) and mapping words to prior subjectivity (strong and weak), and prior polarity (negative, positive and neutral). The prior polarity is reversed when a negative expression precedes the word.

Apart from real-life applications, many application-oriented research papers have also been published. For example, in (liu et al., 2007), a sentiment model was proposed to predict sales

performance. In (Mc Glohon, Glance and Reiter, 2010), reviews were used to rank products and merchants. In (Tumasjan et al., 2010), twitter sentiment was also applied to predict election results. In (Chen et al., 2010), the authors studied political standpoints. In (Yano and smith, 2010), a method was reported for predicting comment volumes of political blogs. In (Asur and Huberman, 2010; Joshi et al., 2010;

CHAPTER THREE.

3.0 METHODOLOGY

3.1 Introduction.

The research approach in this project is an empirical study of social network analysis of tweeter data. These study closely examined the accuracy of analyzed data available and gather clues to what is occurring and why. The study also performed analysis and discussion on how to improve guidance to the business organizations. The data collection method was a combination of both quantitative and qualitative .The project was conducted using a survey of sample random tweets based on a subject.

3.2 Sources of Data

Primary data was used to get facts on the subject. Primary data included data collected from actual tweeter pages using a tweeter API which enabled us to pull data in real time observing analyzed data from the social network. The process was conducted in three rounds on a subject topic using the same sentiment opinions obtained from the tweets to ensure that the algorithms do not produce varying results when subjected to the same data.

3.3 Data Analysis Methods

The aim of data analysis is to examine and organize data in a way that provides answers to research question and ensures that the research objectives are met. This process involved Data analysis of the information obtained in the social network by using the algorithm to generate an accurate result of the information required for use.

3.4 Data Analysis Tools

The analysis process of the data was done by using Microsoft office excel tool whereby all the data obtained from the experiment will be entered and reports in terms of graphs ,tables and pie charts will be generated.

3.5 Development Approach

Iterative development model

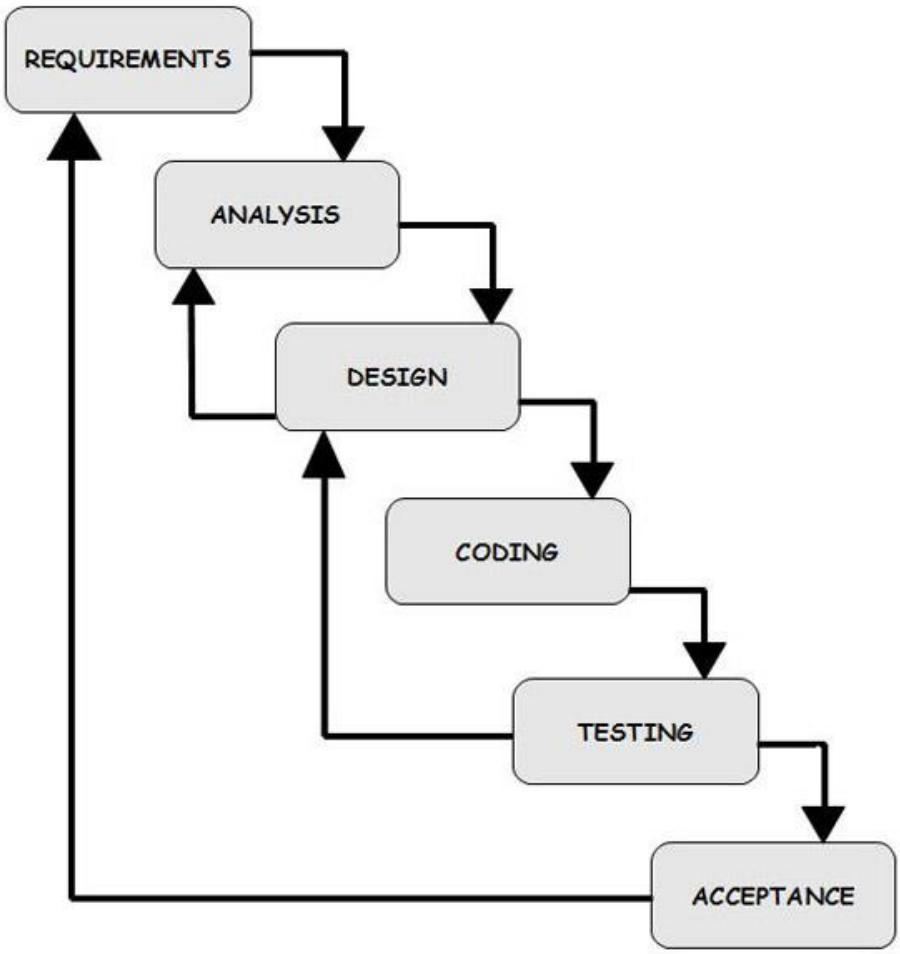


Figure No 2. Diagram of iterative design model for software development

In incremental model the whole requirements is divided into various builds. During each iteration, the development module will go through the requirements, design, implementation and testing phases. Each subsequent release of the module adds function to the previous release. The process continues till the complete system is ready as per the requirements.

The key to successful use of an iterative software development lifecycle is rigorous validation of requirements, and verification & testing of each version of the software against those requirements within each cycle of the model. As the software evolves through successive cycles, tests have to be repeated and extended to verify each version of the software.

Architectural Design

- Extraction of posts from social media using an extraction script. Twitter API was used to collect tweets and then stored in a MSQL database.
- Preprocessing and cleaning of the data.
- The data is then divided into 75% for training and 25% for test data set.
- Training the data so as to come up with a model that can be used to classify new and pure tweets.
- Using the model generated to classify posts which extract features from the tweets collected and classifies them into the three polarities i.e. negative, positive and neutral.
- Results analysis is achieved from the classifiers developed and the conclusions drawn.

Validation of the Prototype

- Is the technique able to collect data from the social media?
- Can the classifier be able to train the data collected?
- Are the features selected for classification and training ideal?
- Are the results accurate and unfavorably biased towards one sentiment?
- Can the application developed generate a visual analysis of its performance graphically or in charts based on naïve bayes, support vector machines and maximum entropy?

3.6 Proposed conceptual model

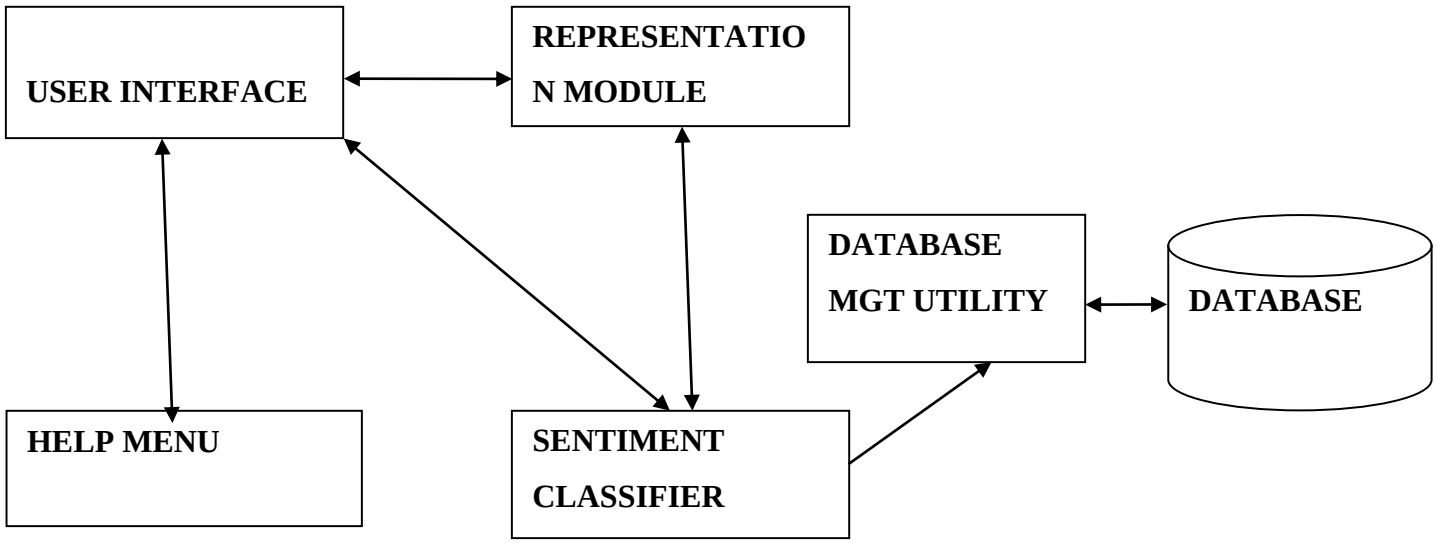


Figure no.3. Conceptual model of the proposed system

User interface: This part in the system handle the entire information a client wishes to visits in order to acquire and input data into the system. The menu bar, task bar and all other system needs will be displayed in an easy to navigate way.

Help module: This module will enable the user to obtain any assistance required to navigate the system

Sentiment classifier: This module enables the client to classify the data based on the subject topic and also to assign a negative or positive sign.

Representation module: This part will enable the vector representation of the text acquired from the tweets

Database .This is the module where the data acquired from the tweets will be stored for analysis in the system

CHAPTER 4.

4.0 SYSTEM ANALYSIS DESIGN AND SPECIFICATIONS

4.1 SOFTWARE REQUIREMENTS AND SPECIFICATIONS

4.2 Functional requirements

In order to meet the objectives the application development should be able to do the following

- Extract posts from twitter and store them in a database for purposes of preprocessing and analysis
- Process the data to remove the low information gain features
- Train the naïve bayes, maximum entropy and support vector machines to come up with a model that can be used to classify new reports
- Classify the posts using the models developed by extracting the relevant features into the three polarities negative, positive, and neutral.
- Provide visual analytics on the results obtained

4.20 User Interfaces

User interface includes various forms and windows. The main window consists of the main search bar and a main menu bar with file, edit, view, tools and help. The interface will visualize the features and functionalities listed in this document for this prototype as the included below not limited to:

- Drop down menu for various option selections
- Selection list for filtering results
- Push buttons for user's feedback and reclassifying tweets
- Visual graphs to show results
- Help button

4.21 Retrieving input

The software retrieves inputs in form of libraries, analysis session duration and tweets.

4.22 Real-time processing

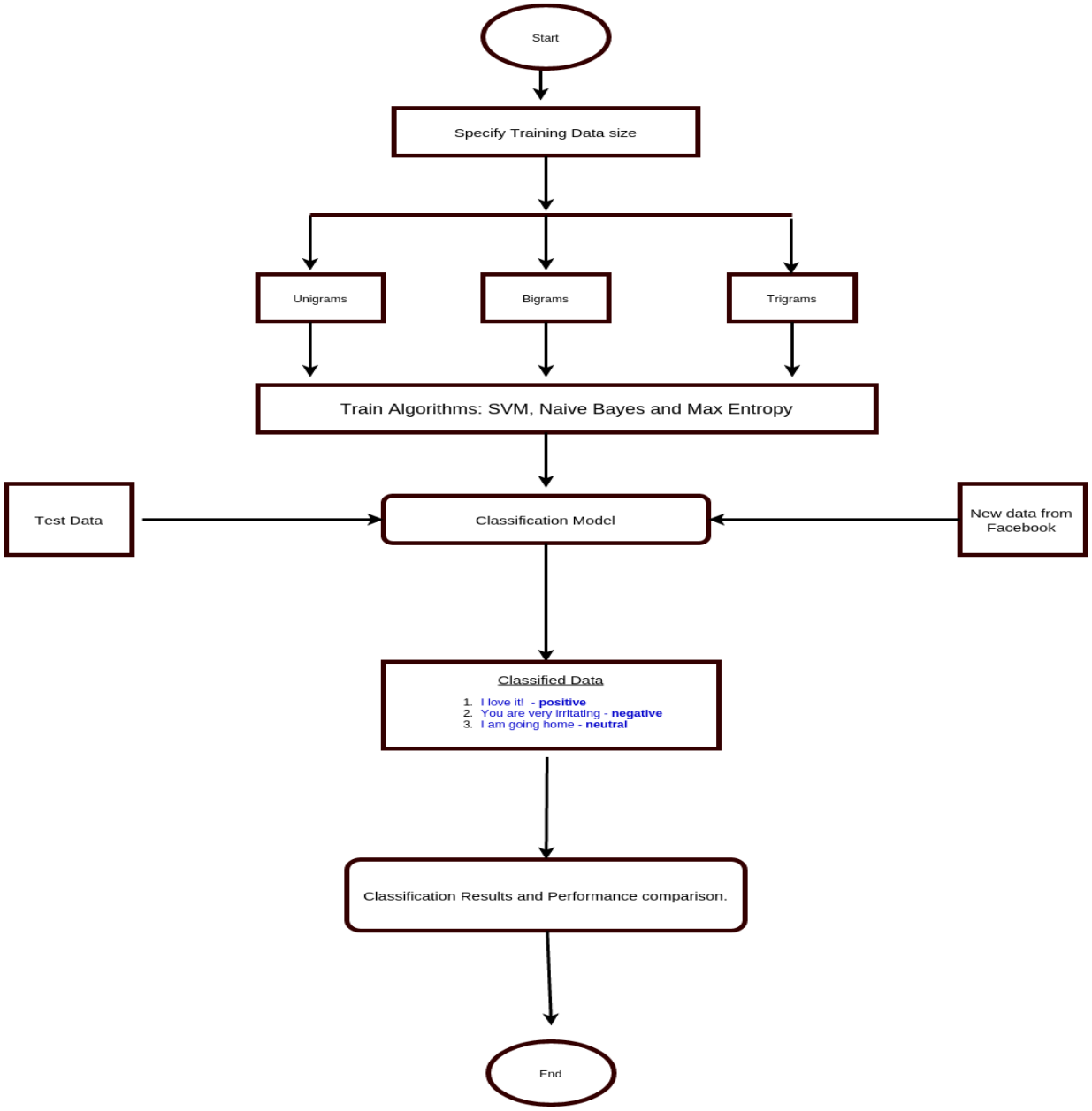
The software takes input, process data and display output in real-time. This ensures the data provided by tweet is a current view of the tweeter community mood.

4.23 Sentiment analysis

This is performed on the keywords within the tweet to determine the overall mood of the tweets relative to the topic. The sentiment analysis provides a negative or positive numeric sentiment value

4.24 Output

The software must output real-time data in the form of simple charts and histograms. In addition, the software may output additional statistics pertaining to a topic.



4.3 NON FUNCTIONAL REQUIREMENTS

4.30 Hardware Interfaces

The solution makes extensive use of several hardware devices. These devices include;

- MySQL database server with intensive use of memory space.
- PHP server with high performance and intensive use for CPU usage.
- Windows and Linux users' computers.

4.31 Communications Interfaces

Internet connection and a web browser are required in order to make use of several functions and to be executed such as searching, viewing and downloading.

4.32 Software Interfaces

The prototype launches the portal over the internet and other than the hardware specified in the hardware interface section, the software requirements are to support windows operating system with support to MySQL, apache and PHP servers.

4.33 Performance

The twitter API provide up-to-date information; limited only by the rate of twitter input. The software provides prompt analysis of the data using the various software packages available to it. The application should be capable of operating in the background should the user wish to utilize other applications.

4.34 Availability

The software is available at all times on the user's device desktop or laptop, as long as the device is in proper working order. The functionality of the software will depend on any external services such as internet access that are required. If those services are unavailable, the user should be alerted.

4.35 Security

The software should never disclose any personal information of twitter users, and should collect no personal information from its own users. The use of passwords and API keys will ensure private use of the twitter API. The programmes will be performed on a password protected laptop and desktop to ensure maximum security.

4.36 Maintainability

The software should be written clearly and concisely. The code is well documented. Particular care will be taken to design the software modularly to ensure that maintenance is easy.

4.4 USER CLASSES AND CHARACTERISTICS

This part is to identify various user classes that we anticipate will use the web application. User classes will be differentiated based on the use, product functions and features, technical expertise, security and privilege levels and educational level. The solution is intended to be used by three main different user classes; system administrators, system operators and customers or regular users. No special knowledge or skills should be assumed for the part of the regular users. Users are not expected to learn or remember a set of commands in order to start using the application. The prototype application will be only a web based and then for the product versions there will be desktop versions. The following clearly describes a visionary role for each participant.

Users:

Users with no particular knowledge needed, users who are interested to use the tool looking for knowing people's thoughts about a desired topic.

4.40 Advanced end users:

Advanced users are those who have valuable input and feedbacks. Users who are more familiar with informative sites and can use our features efficiently. These valuable feeds will lead to enhancement of users' satisfaction.

4.41 System Operators:

- Maintenance for the functional interface of the application and troubleshooting issues
- Suggest possible updates and identifying renewal application needs
- Coordinate with service providers and infrastructure vendors
- Coordinate and communicate with system administrators

4.42 System Administrators:

- Develop and maintain installation and configuration procedures and operational requirements
- Perform weekly/monthly backup operations, ensuring all required files and data are successfully backed up
- Repair and recover from hardware or software failures
- Coordinate and communicate with system operators

4.5 DATA COLLECTION FROM THE TWEETER

For the data gathering, twitter is the only source and using Streaming API that offers high throughput. Using this API is perfect because we can retrieve real time information and also this continuous stream will be retrieved with no end and capturing all the messages in the stream without missing any information.

NO.	NAME	TYPE	CONTENTS
1.	Tweeter .domains	Array of string	List of domains from links mentioned in this Tweet.
2.	Twitter.Geo	Geo	The location from which this Tweet was sent.
3.	Twitter.In_reply_to_screen_name	String	The Twitter username of the user this Tweet is replying to if it is a reply.
4.	Twitter. Links	Array of string	List of links mentioned in Tweet.
5.	Twitter.mentions	Array of string	List of Twitter usernames mentioned in this tweet.
6.	Twitter. Source	String	The source of the Tweet. For example, "web"
7.	Twitter.text	String	The text of the Tweet

Figure 5. Basic information about a single tweet

4.6 DESIGN CONSTRAINTS

Twitter API has some limitations such as twitter API can only return a fixed maximum amount of tweets (1500). The return of a maximum number of tweets may not be met sometimes as there are not enough tweets for the particular keyword.

4.7 LOGICAL DATABASE REQUIREMENTS

The tweets taken from Twitter was stored on an excel spreadsheet. Excel is an excellent programme for storing large amounts of data as well as being easy to upload the data .The data has two columns, column one contains the score of the tweets (positive , negative and neutral), column two will store the actual tweet content. Each row will represent an individual tweet.

4.8 DATASETS

A Twitter API application was used to pull tweets from Twitter's public timeline in real-time. A dataset is created using twitter tweets from a topic that was dominating twitter at the time of data collection. A sentence level sentiment analysis is performed on tweets as many were full of slang words and misspellings. This is done in three phases. In the first phase of a sentence level sentiment analysis pre-processing is done. Secondly a feature vector is created using relevant features. A publicly available sentiment lexicon which consists of around 6800 words in a list of positive and negative opinion words or sentiment words for English was used to separate the tweets. This list was compiled over many years by Liu and Hu (2004) finally tweets are classified into positive and negative classes using different classifiers.

The final sentiment was based on the number of tweets in each class using several sentiment analysis methodologies; the bag-of-words approach, which uses available lexical resources as seen in Turney (2002) sentiment analysis. Machine learning approaches are also used where the tweets dataset was split in two Training and testing. We had a total of 2500 tweets, of these we chose to use 2450 of the data set for training and the remaining 50 tweets to be used for testing. These tweets were then used for training and testing so to conduct a Naive Bayes classifier, maximum entropy and support vector machine classification.

4.9 SYSTEM DESIGN

Architectural design

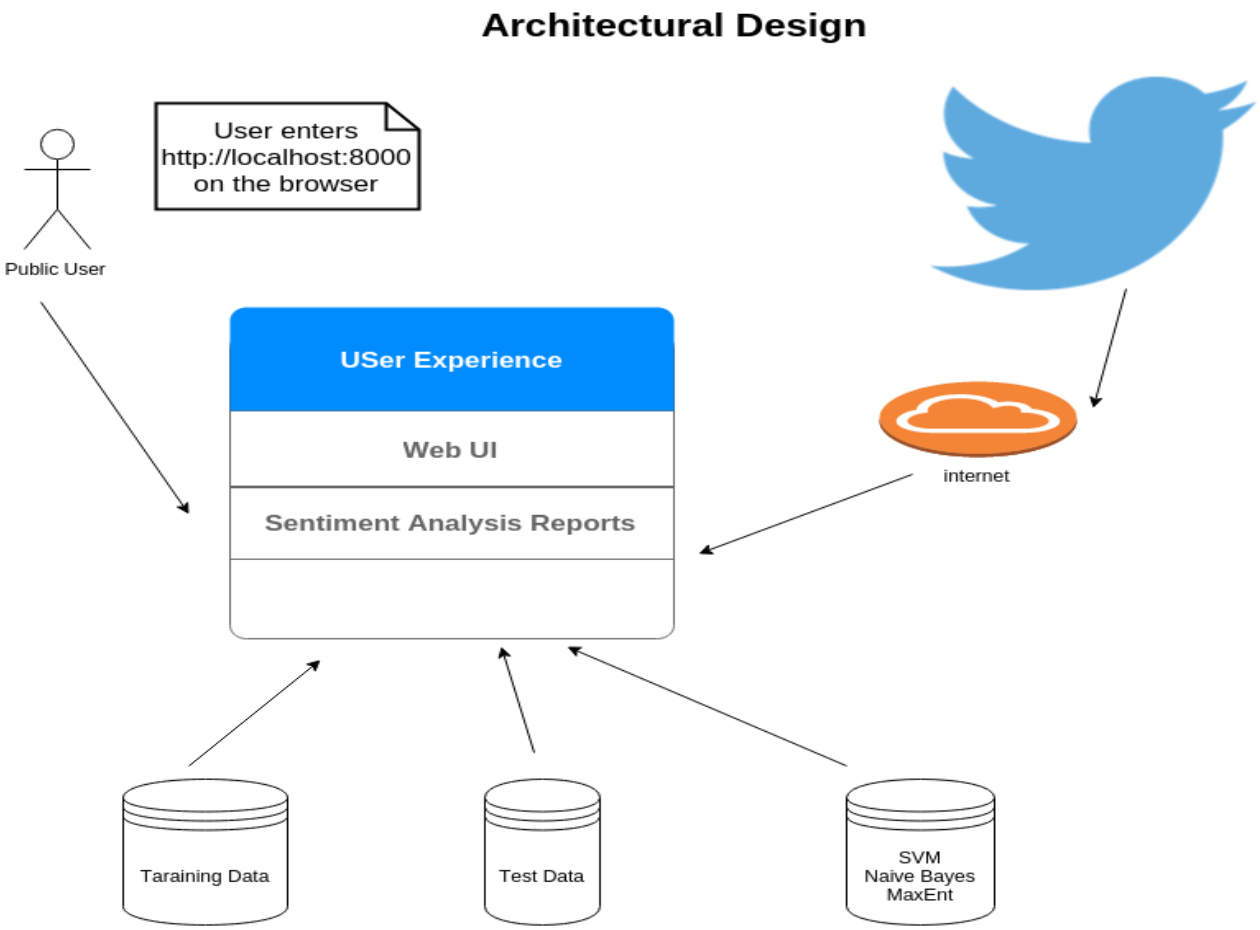


Figure 2.System architectural design

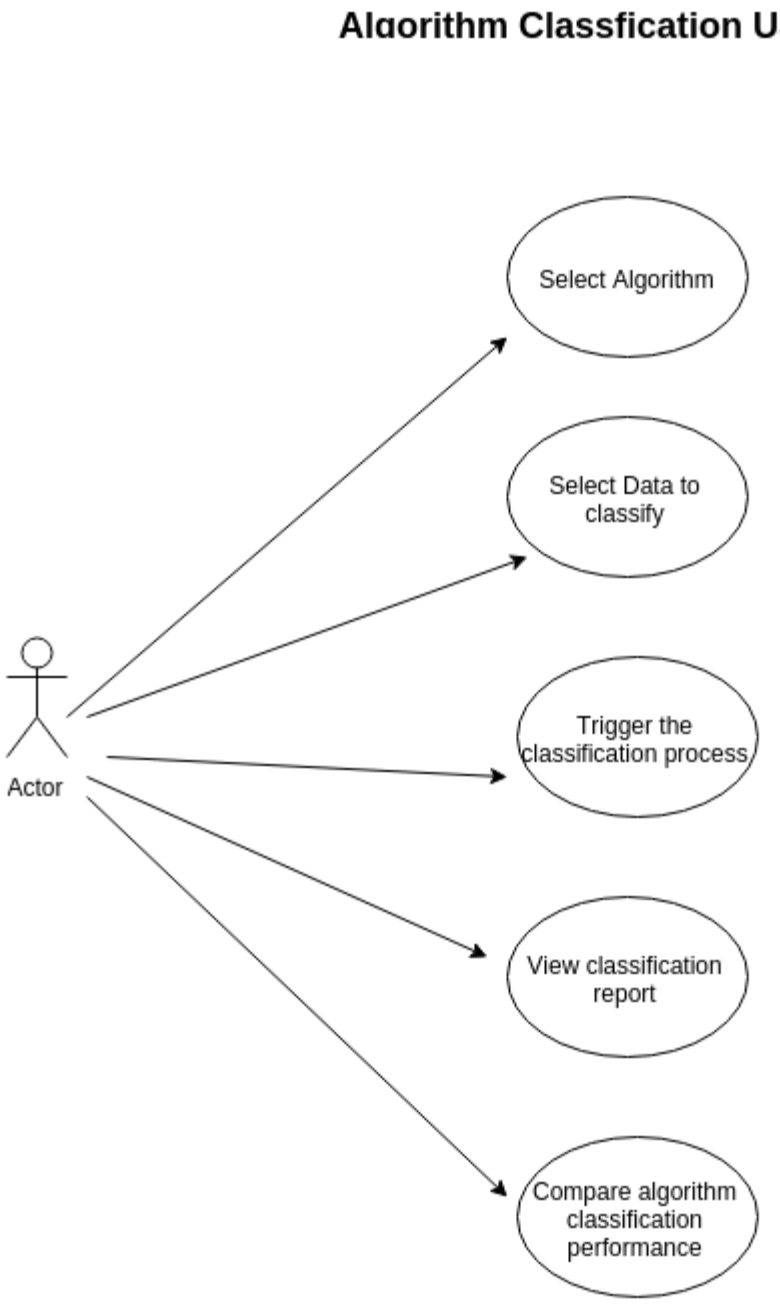


Figure 6.Algorithm classification use case

The user selects the algorithm e.g. naive bayes, select the data to classify, enable the classification process, check report and compare the results based on the same data sample.

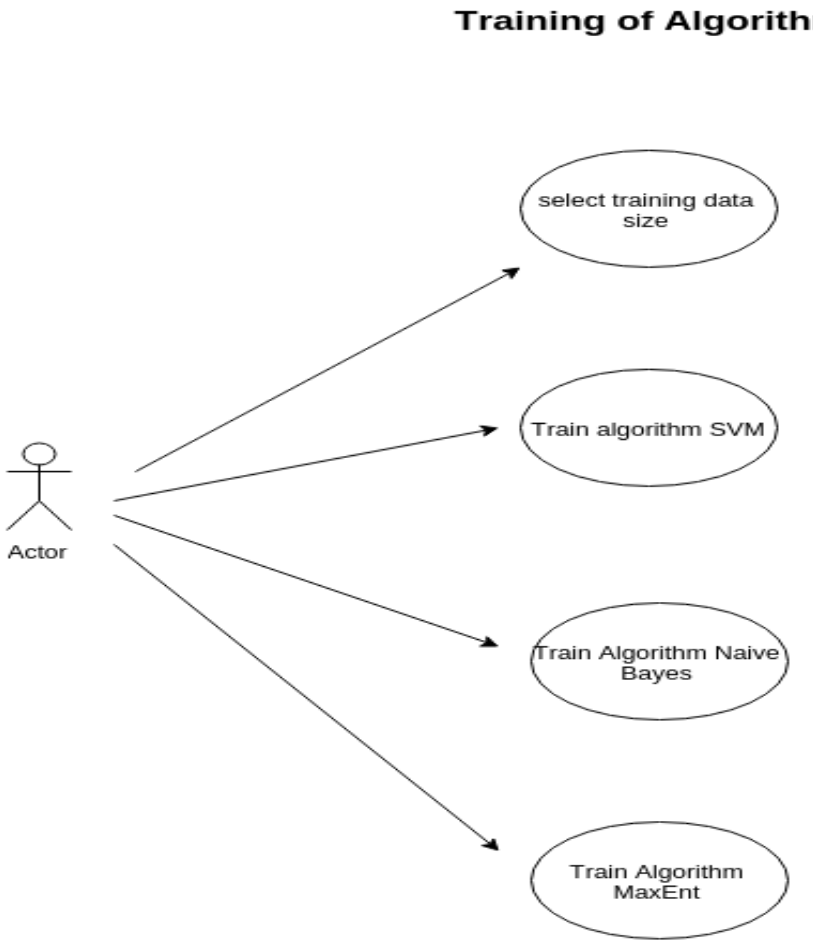


Figure 7.Training of Algorithms

A user can perform a number of processes upon accessing into the system, for example selecting training size and algorithm training. This ensures that the each algorithm achieves independent results after being subjected to same test data.

Database model

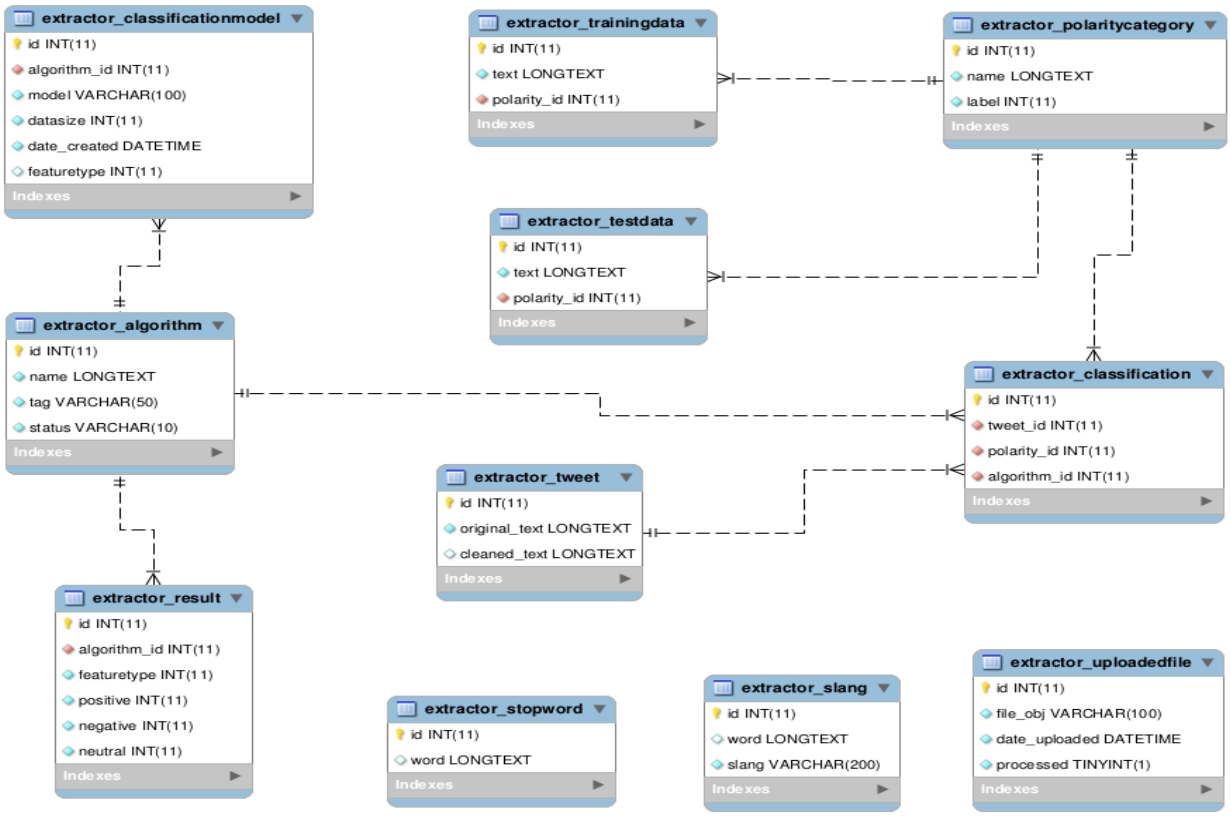


Figure 8. The Database Model

SYSTEM IMPLEMEMNTATION

Programming environment

- Linux operating system-version12.04 and above
- Python interpreter-version2.7.6
- Text editor-sublime. Easy to use, syntax highlighting, lightweight and very customizable

Application Logic

The application logic was implemented using python programming language.

4.90 FRONT END DESIGN

Html-This was used for describing web documents (web pages).

CSS-cascading style sheets were used for describing the presentation of a document written in a markup language.

Java Script-it was used in this project together with Ajax to load data from the database to use interface asynchronously without reloading the page. It was also used to create celery tasks that train algorithms and classify tweets.

High charts- it was used to show classification results and performance comparison from the three algorithms.

4.91 BACKEND LOGIC DESIGN

Python

This is object oriented; high level programming language built in data structures with dynamic semantics thus making it attractive for rapid application development as well as for use as a scripting or glue language to connect existing components together

Django

This is a free and open source web application framework written in python. It was used in this project because python was used as the programming language and the application is web based.

Mysql

It is a free available open source relational database management system that uses structured query language .In this project it was used for storing data, test data, trained algorithm models and tweets to be classified.

Rabbitmq

This was used as message broker software that implements the advanced message queuing protocol (AMQP).The Rabbitmq server is written in Erlang programming language and is built on the open telecom platform framework for clustering and failover. It was chosen because it has robust messaging for applications, easy to use, interoperable, open source and supports a huge number of developer platforms.

Celery

This is an asynchronous task queue based on distributed message passing. It is focused on real time operation but supports scheduling as well. In this project, it was used for executing tasks concurrently on a single or more severs using multiprocessing, this enabled the tasks to execute asynchronously and synchronously. When the train button is clicked, a celery task is created and then pushed to the train queue. The train worker then picks the task from the queue and processes it. In this project background processing has been applied to two major steps.

- **Algorithm training**

Training the algorithms was done on the background so that all the three can be triggered to run in parallel, hence speeding the training process.

- **Data classification**

This was triggered to run in the background hence speeding up the process by running the classification on top of the three algorithms simultaneously.

- **Classifier development**

The three classifiers were implemented using the python programing language and the python natural language tool kit. Support vector machines were implemented using the libsvm library while naïve bayes and maximum entropy was implemented using the natural language tool kit library.

4.92 PRE-PROCESSING

Pre-processing the data is the process of cleaning and preparing the text for classification. Online texts contain usually lots of noise and uninformative parts such as HTML tags, scripts and advertisements. In addition, on words level, many words in the text do not have an impact on the general orientation of it. Keeping those words makes the dimensionality of the problem high and hence the classification more difficult since each word in the text is treated as one dimension.

- **FILTERING**

Use of repeating words like hapyyyy to show their intensity of expression is eliminated because these words are not present in the sentiword .This elimination follows the rule that a letter can't repeat more than three times.

- **QUESTIONS**

The questions like what, which, how etc are not going to contribute to polarity hence in order to reduce the complexity such words are removed.

- **REMOVING SPECIAL CHARACTERS**

Special, characters like [], {}, /' should be removed in order to remove discrepancies using the assignment of polarity. For example "it's good:" if the special characters are not removed sometimes may concatenate with the words and make those words unavailable in the dictionary.

- **REMOVAL OF RETWEETS**

This is the recopying of another user's tweet and posting to another account. This usually happens if a user like s another users tweet.

- **REMOVAL OF URLS**

In general URLS does not contribute to analyze the sentiment in the informal text, for example consider the sentence "I have logged into www.ecstacy .com as I am bored" actually

the above sentence is negative but because of the presence of the word ecstasy it may become neutral and it is a false prediction.

5.0. FEATURE EXTRACTION

We use a variety of features for our classification experiments. For the baseline, we use unigrams and bigrams. We also include features typically used in sentiment analysis, namely features representing information from a sentiment lexicon and POS features. Finally, we include features to capture some of the more domain-specific language of micro blogging.

5.1 N-gram features

To identify a set of useful n-grams, we first remove stop-words. We then performed rudimentary negation detection by attaching the word not to a word that proceeds or follows a negation term. This has proved useful in previous work (Pak and Paroubek 2010). Finally, all unigrams and bigrams were identified in the training data and ranked according to their information gain.

5.2 Lexicon features

Words listed the MPQA subjectivity lexicon (Wilson,Wiebe, and Hoffmann 2009) are tagged with their prior polarity: positive, negative, or neutral. We create three features based on the presence of any words from the lexicon.

5.3 Part-of-speech features

Part-of-Speech Features POS features are common features that have been widely used in the literature for the task of Twitter sentiment analysis. In this project, we build various NB classifiers trained using a combination of word unigrams and POS features and use them as baseline models. We extracted the POS features using the Tweet NLP POS tagger, which is trained specifically from tweets. This differed from the previous work, which relied on POS taggers trained from tree banks in the newswire domain for POS tagging.

5.4 Creating the Training Set

The training dataset was acquired and divided into two sets, 75% used for training and 25%used for testing set.

5.5 Training the Classifiers

The collected data set was used to extract features that we used to train the sentiment classifier .we used the presence of an N-gram as a binary feature, while for general information retrieval purposes, the frequency of a keyword's occurrence is a more suitable feature, since the overall sentiment may not necessarily be indicated through the repeated use of keywords. Pang et.al 2002.we experimented with unigrams, bigrams and trigrams .The process of obtaining n-grams from a tweeter is as follows.

- Removing url links e.g. http://example .com. twitter user names e.g. @alex, special words such as “RT” and emoticons.
- we segment text by splitting it by spaces and punctuation marks to form a bag of words we also ensure that short forms such as “don’t”, ”I’ will” should remain as one word.
- We remove articles (“a”, “an”) from the bag of words.
- Constructing N-grams-a set of N-grams is generated out of consecutive words. A negation (such as “no” and “not” is attached to a word which follows it e.g. “I do not like Safaricom” will form two bigrams “I do not”, “do +not like” and “not +like Safaricom”.This allows to improve accuracy of the classification since the negation plays a special role in an opinion and sentiment expression (Wilson et al' 2005).

Accuracy of support vector machines

We tested our classifier on a set of real twitter posts acquired real-time .The characteristic of the data set are presented in table below

Sentiment	No of samples	Total samples
Positive	1	3
Negative	46	65
Neutral	29	31
Total	76	99

We computed the accuracy of the classifier on the whole evaluation data set i.e.

$$\text{Accuracy} = \frac{\text{N (correct classifications)}}{\text{N (all classifications)}}$$

Accuracy=76/99=76%

Accuracy of naïve bayes

Sentiment	No of samples	manual samples
Positive	0	3
Negative	31	65
Neutral	30	31
Total	61	99

$$\text{Accuracy} = \frac{\text{N (correct classifications)}}{\text{N (all classifications)}}$$

61/99=61%

Accuracy of maximum entropy

Sentiment	No of samples	Manual samples
Positive	1	3
Negative	59	65
Neutral	24	31
Total	74	99

$$\text{Accuracy} = \frac{N \text{ (correct classifications)}}{N \text{ (all classifications)}}$$

74/99=74%

CHAPTER 5.

5.0 FINDINGS AND RESULTS

	Support Vector Machines			**Maximum Entropy**			**Naïve Bayes**		
	Bigram	Unigram	Trigram	Bigram	Unigram	Trigram	Bigram	Unigram	Trigram
5000	76	80	61	74	72	55	61	74	45
4000	73	78	61	72	72	54	58	71	43
3000	69	74	74	71	71	54	56	70	42
2000	67	78	56	70	71	71	55	68	43
1000	66	75	56	69	69	44	54	71	44
Mean %	**70.2**	**77**	**61.6**	**71.2**	**71**	**55.6**	**56.8**	**70.8**	**43.4**

Fig 9: Table of the results

We tested our classifiers against a training set which contains 5112 manually tagged tweets .we provided the test results for unigrams and bigrams both with a test data set of 500 tweets with pos tags .These results are detailed in tabular form. The feature test with the highest accuracy is unigrams with an accuracy of 77%, 70.8% and 71.2%, the best classifier was support vector machines followed by maximum entropy and naïve bayes. The use of bigrams has shown an increase in performance with or without the use of part of speech tags. This also reduces the amount of false positives by the positive classifier; the negative classifier does not seem to be affected much by this. Overall, the use of POS tags has had a negative effect on the accuracy of the occurrence process; this is caused by the ambiguity of the POS tag occurrences across sets. The overall performance of the system is satisfactory; however we would still like to further improve this to ensure that it achieves higher accuracy.

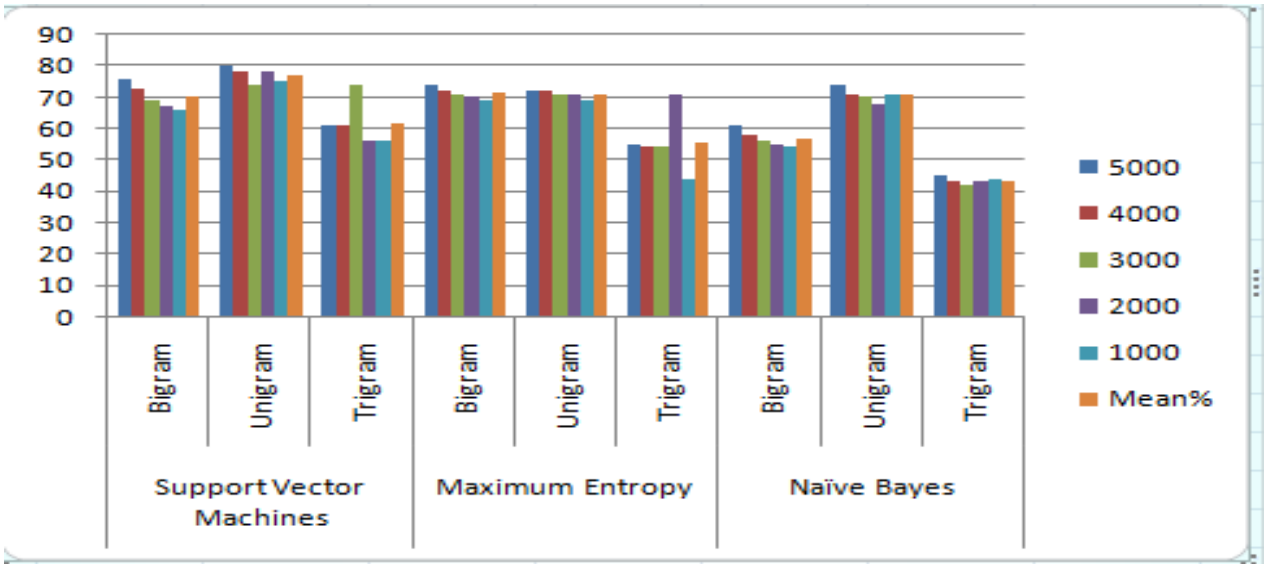
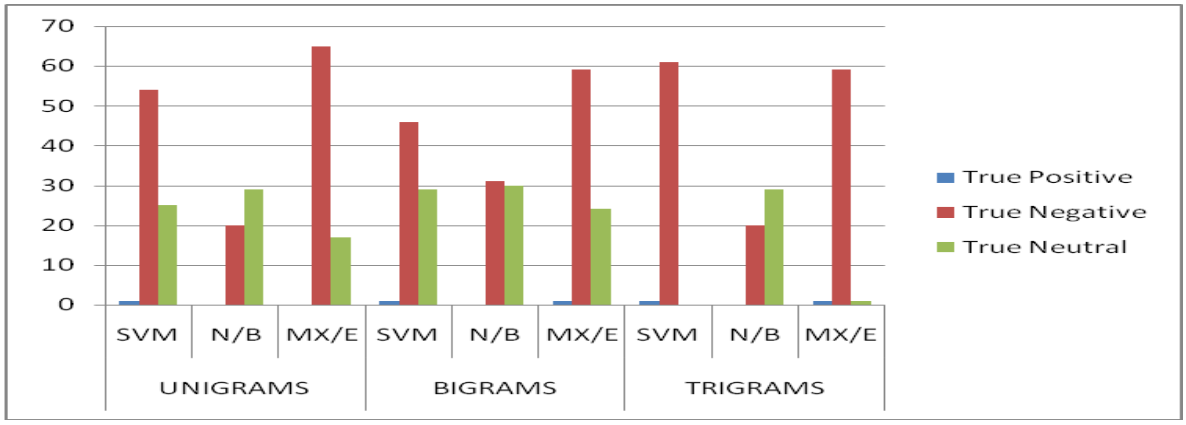


Fig 10: Bar graph of the results

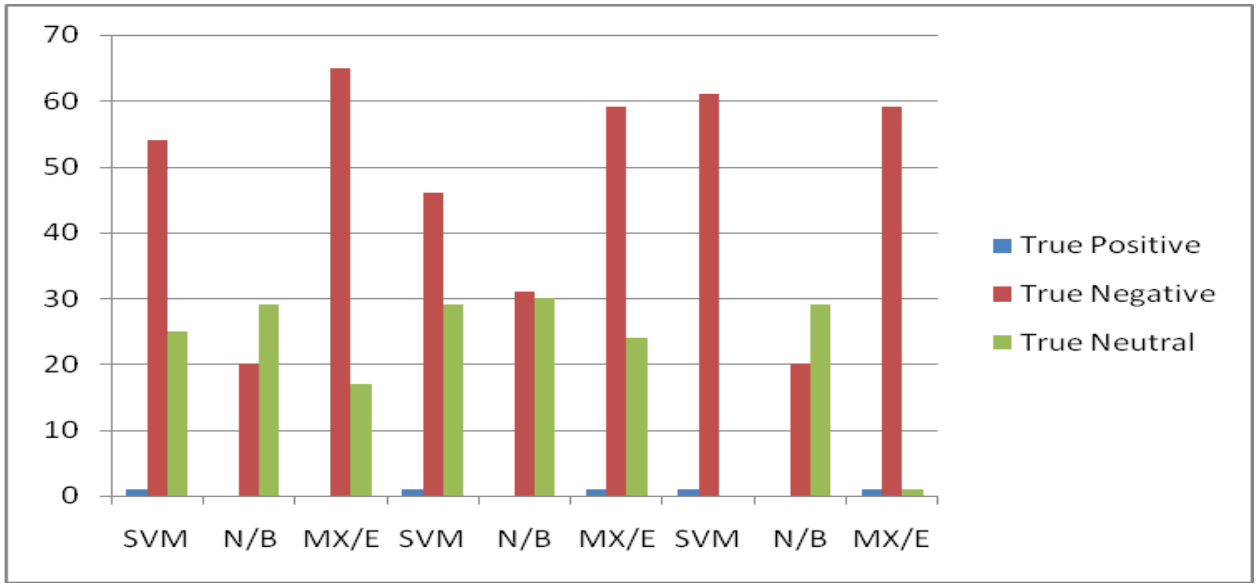
	UNIGRAMS			**BIGRAMS**			**TRIGRAMS**		
	SVM	N/B	MX/E	SVM	N/B	MX/E	SVM	N/B	MX/E
True Positive	1	0	0	1	0	1	1	0	1
True Negative	54	20	65	46	31	59	61	20	59
True Neutral	25	29	17	29	30	24	0	29	1
TOTAL	**80**	**49**	**82**	**76**	**61**	**74**	**64**	**49**	**62**

Fig 11: Table showing true results



	UNIGRAMS			BIGRAMS			TRIGRAMS		
	SVM	N/B	MX/E	SVM	N/B	MX/E	SVM	N/B	MX/E
False Positive	2	3	3	2	3	2	0	3	2
False Negative	4	45	6	19	34	6	4	45	6
False Neutral	31	2	30	2	1	7	31	2	30
TOTAL	**37**	**50**	**39**	**23**	**38**	**15**	**35**	**50**	**38**

Fig 12: Table showing false results



5.1 TESTING FOR ACCURACY

$$\text{M/E. Accuracy} = \frac{\text{N (correct classifications)}}{\text{N (all classifications)}} \qquad \text{Accuracy} = 74/99 = 74\%$$

$$\text{SVM. Accuracy} = \frac{\text{N (correct classifications)}}{\text{N (all classifications)}} \qquad \text{Accuracy} = 76/99 = 76\%$$

$$\text{N/B. Accuracy} = \frac{\text{N (correct classifications)}}{\text{N (all classifications)}} \qquad \text{Accuracy} = 61/99 = 61\%$$

5.2 CLASIFIER PRECISION

This is the exactness of a classifier. A higher precision means less false positives while a lower precision means more false positives this is often at odds with recall as an easy way to improve precision is to decrease recall.

$$\text{PRECISISON} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{SVM (BIGRAMS)} = \frac{1}{1+2} = 33\%$$

$$\text{/E (BIGRAMS)} = \frac{1}{1+3} = 25\%$$

$$\text{N/B (BIGRAMS)} = \frac{0}{0+3} = 0\%$$

5.3 TESTING FOR RECALL

Recall measures the completeness of a classifier. Higher recall means less false negatives while lesser recall means more false negatives .Improving recall often decreased precision because it gets increasingly harder to be precise as the sample space increases.

$$\text{Testing for recall} = \frac{\text{True positives}}{\text{True positives + false negatives}}$$

$$\text{SVM (bigrams)} = \frac{1}{1+4} \quad \text{RECALL} = 20\%$$

$$\text{M/E (BIGRAMS)} = \frac{0}{0+45} \quad \text{RECALL} = 0\%$$

$$\text{NAÏVE BAYES} = \frac{1}{1+6} \quad \text{RECALL} = 14\%$$

Naïve bayes

Naive bayes classifier makes a fast and easy to implement but this adversely affects the quality of the results, if feature words were interrelated. It produced 70.8 % in unigrams, 56.8 % in bigrams and 44 % in trigrams. These low results is attributed to the fact that this classifier is treats each word as independent from each other which is not true and thus does give an accurate result as expected from the other classifiers.

Support vector machines

Support vector machines performed better in all the tests conducted compared to other classification techniques because it is less susceptible to over fitting than other learning methods since the model complexity is independent of the feature space dimension. It can also handle large feature spaces with excellent classification accuracy. It produced the best results both on test and training sets and is robust with respect to the number of features and very fast at training and classification. The greatest challenge with this classification is the complexity in the implementation. In this project the classifier achieved 70.2 % accuracy with bigrams, 77 % in unigrams and 61.6 % in trigrams.

Maximum Entropy

This classifier produced 55.6 % in trigrams,71.2 % in bigrams and 71 % in unigrams .We realize that the classifier has higher accuracy compared to others in unigrams due to the fact that it does the training in iterations which ensures that it perfects the outcome of results. The classifier consumes time in training and learning compared to other classifiers in this project.

5.4 COMPARATIVE ANALYSIS OF THE ALGORITHMS

FEATURE	SUPPORT VECTOR MACHINES	MAXIMUM ENTROPY	NAÏVE BAYES
Accuracy	High	Good	Good
Memory Requirement	High	High	Low
Simplicity	Hard	Hard	Very Simple
Performance	Best	Better	Good
Training Time	High	Moderate	Less
Consistency Of Accuracy	Consistent	consistent	Variable

From our study it was evident that every kind of classification model had its own challenges .Selection of classification models can be decided on the basis of resources, accuracy requirement and training time available. Considering the support vector machines which showed that it was hard to implement, high memory requirements, consistent in data output and consumes more time in training, the classifier was best fit for use in sentiment analysis. However it requires high training time and processing power this hence improved the accuracy of the classifier. If processing power is an issue and memory is an issue then the naïve bayes classifier is selected due to its low processing power and memory consumption less training is required time is required but you have powerful processing system and memory then maximum entropy proves to be a worthy alternative. Support vector machines proved to be average in all aspects and thus proved to be the best choice for sentiment analysis in this project.

CHAPTER 6.
CONCLUSION AND FUTURE WORK

6.0 Introduction

This chapter gives us an overview of the research process, data collection and analysis and finally generation of the findings from the results. The whole research was successful and we compared the three machine learning algorithms based on their performance when subjected to the twitter data for classification.

6.1 Summary

In this research project, we presented a way in which machine learning techniques can be applied to large sets of data to establish their performance in different feature extractions in this case bigrams and unigrams. We looked at common processes in natural language processing that can help us derive meaning or context of a given phrase. We demonstrated how to collect an original twitter posts for sentiment classification and the refinement that is needed with such data. We have applied maximum entropy, naive bayes and support vector machines to this set of data to conduct sentiment analysis and we found the process to be successful.

6.2 Conclusion

On the analysis of the results we have found that bigrams and unigrams offer better performance when conducting the classification process supporting previous results performed by Pak et.al and Turney et.al .we have discovered that collecting data across a short amount of time may be a potential source of error when determining sentiment, this is due to the fact that opinions can shift over time and also the meaning of words. The classification process was successful with accuracy of 77% support vector machine, 71% maximum entropy and 70.8% naive bayes however it is felt that that this could be further improved .This results was best at unigrams followed by bigrams and the worst feature was trigrams when tested with all classifiers under different data sets.

6.3 Recommendations

The results showed a near human accuracy which is 80%. However, in large sets of data it could be impossible to mine the same piece of data in a short time .We have proved that the mechanism can effectively discover market intelligence for supporting decision makers by establishing a monitoring system to track external opinions on different aspects of a business in realtime. A complete accuracy could not be achieved because the classifiers cannot realize texts which are sarcastic training data/feature selection and representation of instances. We also propose on developing an application which carries our textual analysis on video games servers analyzing what a player is expressing and adjusting the game environment accordingly.

6.4 Research contribution

The research process involved devising a way in which we can obtain tweeter data .we classified them manually and developed classifiers which we used to generate a classification model from the tweets. Our major contribution from this of research is that we have exploited three machine learning algorithms to establish their suitability for sentiment classification. We have compared the three based on different aspects of performance including accuracy, precision, recall and hardware/software requirements .The research project based on the results recommended the best algorithm for sentiment analysis, this is a research contribution benefiting business organizations on choosing the best algorithm for social media analysis.

6.5 Code

```
from django.contrib import admin
from extractor.models import *
class TestDataAdmin(admin.ModelAdmin):
   list_display = ('text','polarity')
class TrainingDataAdmin(admin.ModelAdmin):
   list_display = ('text','polarity')
class PolarityCategoryAdmin(admin.ModelAdmin):
   list_display = ('name','label')
class AlgorithmAdmin(admin.ModelAdmin):
   list_display = ('name','tag','status')
class ClassificationModelAdmin(admin.ModelAdmin):
   list_display = ('algorithm','model','datasize','date_created')
```

```
class TweetAdmin(admin.ModelAdmin):
   list_display = ('original_text','cleaned_text')
class ClassificationAdmin(admin.ModelAdmin):
   list_display = ('tweet','polarity','algorithm')
class SlangAdmin(admin.ModelAdmin):
   list_display = ('word','slang')
   search_fields = ['word', 'slang']
class ResultAdmin(admin.ModelAdmin):
   list_display = ('algorithm','featuretype','positive','negative','neutral')
admin.site.register(Result, ResultAdmin)
admin.site.register(Slang, SlangAdmin)
admin.site.register(Classification, ClassificationAdmin)
admin.site.register(Tweet, TweetAdmin)
admin.site.register(ClassificationModel, ClassificationModelAdmin)
admin.site.register(TrainingData, TrainingDataAdmin)
admin.site.register(Algorithm, AlgorithmAdmin)
admin.site.register(PolarityCategory, PolarityCategoryAdmin)
admin.site.register(TestData, TestDataAdmin)
values = ['positive','negative','neutral']
      points = [randint(1,100) for p in range(1,15)]
      print points
      for item in newtweets:
         count+=1
         tweet = item['Tweet']
         bigrams = BaseExtractor.n_grams(tweet, int(featuretype))
         be = BaseExtractor(gram_features)
         polarity = classifier.classify(be.extract_relevant_features(bigrams))
         result = {}
         result["tweet"] = tweet
         result['polarity'] = polarity
         if count in points:
            result['polarity'] = random.choice(values)
            polarity = random.choice(values)
         result['correct'] = item["Correct"]
         if result['correct']==polarity:
            correct+=1
            if polarity=='positive': correct_positive +=1
            if polarity=='negative': correct_negative +=1
            if polarity=='neutral': correct_neutral +=1
         if polarity=='positive': positive_count+=1
         if polarity=='negative': negative_count+=1
         if polarity=='neutral': neutral_count+=1
         tweetl.append(result)
         BaseExtractor.save_classified_tweets(tweet,polarity,"maxent")
      response["tweets"] = tweetl
      response['positives'] = positive_count
```

```
        response['negatives'] = negative_count
        response['neutrals'] = neutral_count

        algorithm = Algorithm.objects.get(tag = "maxent")
        BaseExtractor.save_results(algorithm,
correct_positive,correct_negative,correct_neutral,featuretype)
        try:
            percentage_accuracy = ( int(correct)/float(count) )* 100
        except Exception, e:
            print e
            percentage_accuracy = "NONE"
        response['accuracy'] = str(correct)+"/"+str(count)+"  =  "+ str(percentage_accuracy)+"%"
        print "MaxEnt classification results"
        return response
from extractor.models import *
from extractor.base_extractor import BaseExtractor
import nltk
from django.conf import settings
import pickle
from django.core.files import File
class NaiveBayes(object):
    """docstring for NaiveBayes"""
    def __init__(self):
        super(NaiveBayes, self).__init__()
    @staticmethod
    def train(datasize,featuretype):
        print "inside algo_naive_bayes train..."
        algo = Algorithm.objects.get(tag = 'naive')
        algo.status = 'training'
        algo.save()
        datasize = int(datasize)
        data = TrainingData.objects.all()[:datasize]
        if data:
            gram_features,tweets = BaseExtractor.get_relevant_features(datasize,int(featuretype))
            be = BaseExtractor(gram_features)
            training_set = nltk.classify.apply_features(be.extract_relevant_features,tweets)
            classifier = nltk.NaiveBayesClassifier.train(training_set)
            print "created classifier"
            _file = settings.BASE_DIR + "/data/tempfiles/naive_bayes_classifier.pickle"
            f = open(_file, 'wb')
            pickle.dump(classifier, f)
            f.close()
            print 'saved classifier model in file.'
            with open(_file) as fileobj:
                obj = ClassificationModel.objects.create(
                    model = File(fileobj),
```

```
                algorithm = algo,
                datasize = datasize,
                featuretype = int(featuretype)
                )
            obj.save()
        algo.status = 'trained'
        algo.save()
        return "Finished training."
    else:
        return "No training data was found."
@staticmethod
def classify(tweets,test=False):

    # newtweets = BaseExtractor.clean_data(tweets)
    newtweets = tweets
    response = {}
    tweetl = []
    positive_count=0
    negative_count=0
    neutral_count=0
    correct_positive = 0
    correct_negative = 0
    correct_neutral = 0
    count = 0
    correct = 0
    model = ClassificationModel.objects.filter(algorithm__tag =
'naive').order_by("date_created").last()
    filepath = model.model.path
    datasize = model.datasize
    featuretype = model.featuretype
    f = open(filepath)
    classifier = pickle.load(f)
    f.close()
    gram_features,tweets = BaseExtractor.get_relevant_features(datasize,int(featuretype))
    for item in newtweets:
        tweet = item['Tweet']
        bigrams = BaseExtractor.n_grams(tweet, int(featuretype))
        be = BaseExtractor(gram_features)
        polarity = classifier.classify(be.extract_relevant_features(bigrams))
        result = {}
        result['tweet'] = tweet
        result['polarity'] = polarity
        result['correct'] = item["Correct"]
        if result['correct']==polarity:
            correct+=1
            if polarity=='positive': correct_positive +=1
```

```
                if polarity=='negative': correct_negative +=1
                if polarity=='neutral': correct_neutral +=1
            if polarity=='positive': positive_count+=1
            if polarity=='negative': negative_count+=1
            if polarity=='neutral': neutral_count+=1
            count+=1
            tweetl.append(result)
            BaseExtractor.save_classified_tweets(tweet,polarity,"naive")
        response['tweets'] = tweetl
        response['positives'] = positive_count
        response['negatives'] = negative_count
        response['neutrals'] = neutral_count
        if test:
            algorithm = Algorithm.objects.get(tag = "naive")
            BaseExtractor.save_results(algorithm,
correct_positive,correct_negative,correct_neutral,featuretype)
            try:
                percentage_accuracy = ( int(correct)/float(count) )* 100
            except Exception, e:
                print e
                percentage_accuracy = "NONE"
            response['accuracy'] = str(correct)+"/"+str(count)+"  =  "+
str(percentage_accuracy)+"%"
        print "naive_bayes classification results"
        print response
        return response
from libsvm.python.svmutil import *
from extractor.models import *
from libsvm.python import svm
import nltk
import csv, pickle
from django.core.files import File
from extractor.base_extractor import BaseExtractor
from django.conf import settings
class SupportVectorMachine(object):
    """docstring for SupportVectorMachine"""
    def __init__(self):
        super(SupportVectorMachine, self).__init__()
    @staticmethod
    def train(datasize,featuretype):
        trainingData = TrainingData.objects.values('text','polarity__name').distinct()[:datasize]
        data = []
        labels = []
        feature_vectors = []
        words = []
        gram_features,tweets = BaseExtractor.get_relevant_features(datasize,int(featuretype))
```

```
    be = BaseExtractor(gram_features)
    if trainingData:
        for item in trainingData:
            text = item['text']
            polarity = item['polarity__name']
            words = [w for w in text.split()]
            bigrams = BaseExtractor.n_grams(text, int(featuretype))
            polarity = polarity.lower().strip()
            label = PolarityCategory.objects.get(name = polarity).label
            fv = be.extract_relevant_features(bigrams)
            feature_vectors.append(fv.values())
            labels.append(label)
        try:
            problem = svm_problem(labels, feature_vectors)
        except Exception, e:
            raise e
        param = svm_parameter('-q')
        param.kernel_type = LINEAR
        classifier = svm_train(problem, param)
        _file = settings.BASE_DIR + "/data/tempfiles/svm_classifier.pickle"
        svm_save_model(_file, classifier)
        algo = Algorithm.objects.get(tag = 'svm')
        with open(_file) as fileobj:
            obj = ClassificationModel.objects.create(
                model = File(fileobj),
                algorithm = algo,
                datasize = datasize,
                featuretype = featuretype
                )
            obj.save()
        algo.status = 'trained'
        algo.save()
        return "Finished training."
    else:
        return "No training data found"
@staticmethod
def classify(newtweets,test = False):
    if newtweets:
        text_list = []
        feature_vectors = []
        response = {}
        tweetsl = []
        positive_count = 0
        negative_count = 0
        neutral_count = 0
        correct_positive = 0
```

```
        correct_negative = 0
        correct_neutral = 0
        tweets_list = []
        for item in newtweets:
          tweets_list.append(item['Tweet'])
        processedData = tweets_list
        if not processedData:
            print "No data to classify..."
            return
        classifier = ClassificationModel.objects.filter(algorithm__tag =
'svm').order_by("date_created").last()
        if not classifier:
            print "Trained model not found..."
            return
        featuretype = classifier.featuretype
        filepath = classifier.model.path
        datasize = classifier.datasize
        gram_features,tweets = BaseExtractor.get_relevant_features(datasize,int(featuretype))
        be = BaseExtractor(gram_features)
        for text in processedData:
            bigrams = BaseExtractor.n_grams(text, int(featuretype))
            fv = be.extract_relevant_features(bigrams)
            feature_vectors.append(fv.values())
        classifier = svm_load_model(filepath)
        p_labels, p_accs, p_vals = svm_predict([0] * len(feature_vectors), feature_vectors,
classifier)
        count = 0
        correct = 0
        for text in newtweets:
            found = False
            polarity = p_labels[count]
            if(polarity == 1):
                polarity = 'positive'
                found = True
                positive_count +=1
            elif(polarity == -1):
                polarity = 'negative'
                found = True
                negative_count+=1
            elif(polarity == 0):
                polarity = 'neutral'
                found = True
                neutral_count+=1
            count += 1
            res = {}
            if found:
```

```
                res["tweet"] = text['Tweet']
                res['polarity'] = polarity
                res['correct'] = text['Correct']
                if test:
                    if res['correct'] == res['polarity']:
                        correct +=1
                        if polarity=='positive': correct_positive +=1
                        if polarity=='negative': correct_negative +=1
                        if polarity=='neutral': correct_neutral +=1
                BaseExtractor.save_classified_tweets(text['Tweet'],polarity,"svm")
            else:
                res["tweet"] = text['Tweet']
                res['polarity'] = "unclassified"
                # if test: res['correct'] = text['Correct']
                res['correct'] = text['Correct']
            tweetsl.append(res)
        response['tweets'] = tweetsl
        response['positives'] = positive_count
        response['negatives'] = negative_count
        response['neutrals'] = neutral_count
        if test:
            algorithm = Algorithm.objects.get(tag = "svm")
            BaseExtractor.save_results(algorithm,
correct_positive,correct_negative,correct_neutral,featuretype)
            try:
                percentage_accuracy = ( int(correct)/float(count) )* 100
            except Exception, e:
                print e
                percentage_accuracy = "NONE"
            response['accuracy'] = str(correct)+"/"+str(count)+"  =  "+
str(percentage_accuracy)+"%"
        return response
    from django.db import models
from django.utils import timezone
class UploadedFile(models.Model):
    file_obj = models.FileField(upload_to='uploaded_files/%Y/%m/%d')
    date_uploaded = models.DateTimeField(default = timezone.now)
    processed = models.BooleanField(default = False)
class PolarityCategory(models.Model):
    name = models.TextField()
    label = models.IntegerField()
    def __str__(self):
        return self.name
class TrainingData(models.Model):
    text = models.TextField()
    polarity = models.ForeignKey(PolarityCategory)
```

```
    def __str__(self):
    return self.text
class TestData(models.Model):
    text = models.TextField()
    polarity = models.ForeignKey(PolarityCategory)
    def __str__(self):
        return self.text
class Algorithm(models.Model):
    name = models.TextField()
    tag = models.CharField(max_length = 50)
    STATUSES = (
        ('trained', 'Trained'),
        ('training', 'Training'),
        ('untrained', 'Untrained')
    )
    status = models.CharField(max_length = 10,choices = STATUSES,default = 'untrained')
    def __str__(self):
        return self.name
class Result(models.Model):
    algorithm = models.ForeignKey(Algorithm)
    featuretype =  models.IntegerField(default = 2)
    positive = models.IntegerField(default = 0)
    negative = models.IntegerField(default = 0)
    neutral = models.IntegerField(default = 0)
class Tweet(models.Model):
    original_text = models.TextField()
    cleaned_text = models.TextField(null=True,blank=True)
    def __str__(self):
        return self.cleaned_text
class Classification(models.Model):
    tweet = models.ForeignKey(Tweet,  related_name='classification_tweet')
    polarity = models.ForeignKey(PolarityCategory, related_name='classification_polarity')
    algorithm = models.ForeignKey(Algorithm, related_name='classification_algorithm')
    class Meta:
        unique_together = ('tweet', 'polarity','algorithm',)
class ClassificationModel(models.Model):
    algorithm = models.ForeignKey(Algorithm)
    model = models.FileField(upload_to='classifier_models/%Y/%m/%d')
    datasize = models.IntegerField(default = 0)
    featuretype = models.IntegerField(default = 2)
    date_created = models.DateTimeField(default = timezone.now)
class Slang(models.Model):
    word = models.TextField(null=True,blank=True)
    slang = models.CharField(max_length = 200, unique=True)
    def __str__(self):
        return self.slang
```

```
class Stopword(models.Model):
    word = models.TextField(null=True,blank=True)
    def __str__(self):
        return self.word
from __future__ import absolute_import
from celery import shared_task
from extractor.algo_naive_bayes import NaiveBayes
from extractor.algo_svm import SupportVectorMachine
from extractor.algo_maxent import MaxEnt
from extractor.models import *
MAX_RETRIES = 1
COUNTDOWN = 20
@shared_task(bind=True,name="extractor.tasks.train",max_retries=MAX_RETRIES,ignore_
result=True)
def train(self,algorithm, datasize,featuretype):
    try:
        print "inside train..."
        results = []
        if algorithm == 'svm':
            response = SupportVectorMachine.train(datasize,featuretype)
        elif algorithm == 'naive':
            response = NaiveBayes.train(datasize,featuretype)
            print response
        elif algorithm == 'max':
            response = MaxEnt.train(datasize,featuretype)
            print response
    except Exception as e:
      raise self.retry(exc=e, countdown=COUNTDOWN)
from django.shortcuts import render
from django.views.generic import TemplateView
from django.views.generic import DetailView
import json
from django.shortcuts import *
from extractor.models import *
from django.core import serializers
from extractor import tasks
from extractor.text_parser import Parser
import requests
from extractor.algo_naive_bayes import NaiveBayes
from extractor.algo_svm import SupportVectorMachine
from extractor.algo_maxent import MaxEnt
TOKEN = False
TEST = True
class HomeView(TemplateView):
    context_object_name = "home"
    template_name = "home.html"
```

```
def dashboard(request):
    args = {}
    try:
        args['svm'] = Algorithm.objects.get(tag = 'svm').status
        args['naive'] = Algorithm.objects.get(tag = 'naive').status
        args['max_ent'] = Algorithm.objects.get(tag = 'maxent').status
        args['positives'] = 0
        args['negatives'] = 0
        args['neutrals'] = 0
        args['correct'] = 0
        args['total'] = 0
    except Exception, e:
        pass
    args['results'] = Classification.objects.all()
    return render(request, "dashboard.html",args)
def get_training_data():
    data = []
    try:
       objectQuerySet = TestData.objects.all()
        for tweet in objectQuerySet:
            cleaned_text = Parser.process(tweet.text)
            obj,created = Tweet.objects.get_or_create(
                        original_text = tweet.text,
                        cleaned_text = cleaned_text
                    )
            res = {}
            res['cleaned_text'] = cleaned_text
            res['polarity'] = tweet.polarity.name
            data.append(res)
    except Exception, e:
        print e
    return data
def load_test_data(request):
    global TEST
    TEST = True
    response = {}
    try:
        response["tweets"] = get_training_data()
    except Exception, e:
        print e
    return HttpResponse(json.dumps(response), content_type = "application/json")
def getToken():
     appID = "920297654658810"
     appSECRET = "d67902fbb470c3efbd0d12b4b1c689ab"
     url = "
     response = requests.get(url)
```

```
    token = response.content.split("=")[1]
    return token
def load_live_comments(request):
    try:
        global TOKEN
        global TEST
        TEST = False
        if not TOKEN:
            TOKEN = getToken()
        token = TOKEN
        page = "SafaricomLtd"
        limit = 250
        data = []
        state = False
        filteredData = []
        response2 = {}
        url =         response = requests.get(url)
        if response.status_code==400:
                    response = requests.get(url)
            data = response.json()
        elif response.status_code==200:
            data = response.json()
        if data:
            state = True
            temp = []
            for d in data['data']:
                try:
                    text = d['message']
                    cleaned_text = Parser.process(text)
                    obj,created = Tweet.objects.get_or_create(
                                original_text = text,
                                cleaned_text = cleaned_text
                            )
                    res = {}
                    if cleaned_text not in temp:
                        temp.append(cleaned_text)
                        res['cleaned_text'] = cleaned_text
                        res['polarity'] = ""
                        filteredData.append(res)
                except Exception, e:
                    print str(e)
        response2['tweets'] = filteredData
    except Exception, e:
        print e
        response2['tweets'] = filteredData
    return HttpResponse(json.dumps(response2), content_type = "application/json")
```

```
def toggle_train_status(request):
    algos = Algorithm.objects.all()
    data = {}
    for algo in algos:
        if algo.trained:
            algo.trained = False
            data[algo.tag] = False
        else:
            algo.trained = True
            data[algo.tag] = True
    return HttpResponse(json.dumps(data), content_type = "application/json")
def train(request,algorithm,datasize,featuretype):
    try:
        tasks.train.apply_async(
            args=[algorithm,datasize,featuretype],
            queue = 'train',
            )
    except Exception, e:
        print e
    return HttpResponse(status=200)
def classify(request):
    response = {}
    global TEST
    try
      data = json.loads(request.POST['tweets'])
        algorithm = request.POST['algorithm']
        algorithm = algorithm.strip().lower()

        response = run_classifier(data, algorithm, TEST)
    except Exception, e:
        print e
    return HttpResponse(json.dumps(response), content_type = "application/json")
def compare(request,featuretype):
    response = {}
    try:
        svm = Result.objects.get(algorithm__tag = 'svm',featuretype = featuretype)
        response['svm'] = [svm.positive,svm.neutral,svm.negative]
    except Exception, e:
        print e
    try:
        naive = Result.objects.get(algorithm__tag = 'naive',featuretype = featuretype)
        response['naive'] = [naive.positive,naive.neutral,naive.negative]
    except Exception, e:
        print e
    try:
        maxent = Result.objects.get(algorithm__tag = 'maxent',featuretype = featuretype)
```

```
        response['maxent'] = [maxent.positive,maxent.neutral,maxent.negative]
    except Exception, e:
        print e
    p = TestData.objects.filter(polarity__name = "positive").count()
    n = TestData.objects.filter(polarity__name = "negative").count()
    neu = TestData.objects.filter(polarity__name = "neutral").count()
    response['manual'] = [p,neu,n]
    print response['manual']
    return HttpResponse(json.dumps(response), content_type = "application/json")
def run_classifier(data, algorithm,test):
    try:
        if algorithm == 'svm':
          response = SupportVectorMachine.classify(data,test = test)
        elif algorithm == 'naive':
          response = NaiveBayes.classify(data,test = test)
        elif algorithm == 'maxent':
          response = MaxEnt.classify(data,test = test)
        else:
            response = "Algorithm not found.."
    except Exception as e:
      print e
      response = e
    return response
def clear_classification(requests):
    Classification.objects.filter().delete()
    response= "All classification results have been cleared from the database."
    return HttpResponse(response, content_type = "application/json")
# def getToken():
#     appID = "920297654658810"
#     appSECRET = "d67902fbb470c3efbd0d12b4b1c689ab"
#     url = "
#     response = requests.get(url)
#     token = response.content.split("=")[1]
#     return token
# import requests
# token = getToken()
# page = "SafaricomLtd"
# limit = 250
# data = []
# state = False
# filteredData = []
# r = SupportVectorMachine.classify(t)
# from extractor.models import *
# trainingData = TrainingData.objects.values('text','polarity__name').distinct()[:datasize]
from libsvm.python.svmutil import *
from extractor.models import *
```

```
from libsvm.python import svm
import nltk
import csv, pickle
from django.core.files import File
from extractor.base_extractor import BaseExtractor
from django.conf import settings
class SupportVectorMachine(object):
    """docstring for SupportVectorMachine"""
    def __init__(self):
        super(SupportVectorMachine, self).__init__()
    @staticmethod
    def train(datasize,featuretype):
        trainingData = TrainingData.objects.values('text','polarity__name').distinct()[:datasize]
        data = []
        labels = []
        feature_vectors = []
        words = []
        gram_features,tweets = BaseExtractor.get_relevant_features(datasize,int(featuretype))
        be = BaseExtractor(gram_features)
        if trainingData:
            for item in trainingData:
                text = item['text']
                polarity = item['polarity__name']
                words = [w for w in text.split()]
                bigrams = BaseExtractor.n_grams(text, int(featuretype))
                polarity = polarity.lower().strip()
                label = PolarityCategory.objects.get(name = polarity).label
                fv = be.extract_relevant_features(bigrams)
                feature_vectors.append(fv.values())
                labels.append(label)
            try:
                problem = svm_problem(labels, feature_vectors)
            except Exception, e:
                raise e
            param = svm_parameter('-q')
            param.kernel_type = LINEAR

            classifier = svm_train(problem, param)
            _file = settings.BASE_DIR + "/data/tempfiles/svm_classifier.pickle"
            svm_save_model(_file, classifier)
            algo = Algorithm.objects.get(tag = 'svm')
            with open(_file) as fileobj:
                obj = ClassificationModel.objects.create(
                    model = File(fileobj),
                    algorithm = algo,
                    datasize = datasize,
```

```
                featuretype = featuretype
                )
            obj.save()
        algo.status = 'trained'
        algo.save()
        return "Finished training."
    else:
        return "No training data found"
@staticmethod
def classify(newtweets,test = False):
    if newtweets:
        text_list = []
        feature_vectors = []
        response = {}
        tweetsl = []
        positive_count = 0
        negative_count = 0
        neutral_count = 0
        correct_positive = 0
        correct_negative = 0
        correct_neutral = 0
        tweets_list = []
        for item in newtweets:
          tweets_list.append(item['Tweet'])
        processedData = tweets_list
        if not processedData:
            print "No data to classify..."
            return
        classifier = ClassificationModel.objects.filter(algorithm__tag =
'svm').order_by("date_created").last()
        if not classifier:
            print "Trained model not found..."
            return
        featuretype = classifier.featuretype
        filepath = classifier.model.path
        datasize = classifier.datasize
        gram_features,tweets = BaseExtractor.get_relevant_features(datasize,int(featuretype))
        be = BaseExtractor(gram_features)
        for text in processedData:
            bigrams = BaseExtractor.n_grams(text, int(featuretype))
            fv = be.extract_relevant_features(bigrams)
            feature_vectors.append(fv.values())
        classifier = svm_load_model(filepath)
        p_labels, p_accs, p_vals = svm_predict([0] * len(feature_vectors), feature_vectors,
classifier)
        count = 0
```

```
        correct = 0
        for text in newtweets:
            found = False
            polarity = p_labels[count]
            if(polarity == 1):
                polarity = 'positive'
                found = True
                positive_count +=1
            elif(polarity == -1):
                polarity = 'negative'
                found = True
                negative_count+=1
            elif(polarity == 0):
                polarity = 'neutral'
                found = True
                neutral_count+=1
            count += 1
            res = {}
            if found:
                res["tweet"] = text['Tweet']
                res['polarity'] = polarity

                res['correct'] = text['Correct']
                if test:
                    if res['correct'] == res['polarity']:
                        correct +=1
                        if polarity=='positive': correct_positive +=1
                        if polarity=='negative': correct_negative +=1
                        if polarity=='neutral': correct_neutral +=1
                BaseExtractor.save_classified_tweets(text['Tweet'],polarity,"svm")
            else:
                res["tweet"] = text['Tweet']
                res['polarity'] = "unclassified"
                # if test: res['correct'] = text['Correct']
                res['correct'] = text['Correct']
            tweetsl.append(res)
        response['tweets'] = tweetsl
        response['positives'] = positive_count
        response['negatives'] = negative_count
        response['neutrals'] = neutral_count
        if test:
            algorithm = Algorithm.objects.get(tag = "svm")
            BaseExtractor.save_results(algorithm,
correct_positive,correct_negative,correct_neutral,featuretype)
            try:
                percentage_accuracy = ( int(correct)/float(count) )* 100
```

```
            except Exception, e:
                print e
                percentage_accuracy = "NONE"
            response['accuracy'] = str(correct)+"/"+str(count)+"  =  "+
str(percentage_accuracy)+"%"
        return response
    from libsvm.python.svmutil import *
from extractor.models import *
from libsvm.python import svm
import nltk
import csv, pickle
from django.core.files import File
from extractor.base_extractor import BaseExtractor
from django.conf import settings
class SupportVectorMachine(object):
    """docstring for SupportVectorMachine"""
    def __init__(self):
        super(SupportVectorMachine, self).__init__()
    @staticmethod
    def train(datasize,featuretype):
        trainingData = TrainingData.objects.values('text','polarity__name').distinct()[:datasize]
        data = []
        labels = []
        feature_vectors = []
        words = []
        gram_features,tweets = BaseExtractor.get_relevant_features(datasize,int(featuretype))
        be = BaseExtractor(gram_features)
        if trainingData:
            for item in trainingData:
                text = item['text']
                polarity = item['polarity__name']
                words = [w for w in text.split()]
                bigrams = BaseExtractor.n_grams(text, int(featuretype))
                polarity = polarity.lower().strip()
                label = PolarityCategory.objects.get(name = polarity).label
                fv = be.extract_relevant_features(bigrams)
                feature_vectors.append(fv.values())
                labels.append(label)
            try:
                problem = svm_problem(labels, feature_vectors)
            except Exception, e:
                raise e
            param = svm_parameter('-q')
            param.kernel_type = LINEAR
            classifier = svm_train(problem, param)
            _file = settings.BASE_DIR + "/data/tempfiles/svm_classifier.pickle"
```

```
            svm_save_model(_file, classifier)
            algo = Algorithm.objects.get(tag = 'svm')
            with open(_file) as fileobj:
               obj = ClassificationModel.objects.create(
                  model = File(fileobj),
                  algorithm = algo,
                  datasize = datasize,
                  featuretype = featuretype
                  )
               obj.save()
            algo.status = 'trained'
            algo.save()
            return "Finished training."
         else:
            return "No training data found"
      @staticmethod
      def classify(newtweets,test = False):
         if newtweets:
            text_list = []
            feature_vectors = []
            response = {}
            tweetsl = []
            positive_count = 0
            negative_count = 0
            neutral_count = 0
            correct_positive = 0
            correct_negative = 0
            correct_neutral = 0
            tweets_list = []
            for item in newtweets:
              tweets_list.append(item['Tweet'])
            processedData = tweets_list
            if not processedData:
               print "No data to classify..."
               return
            classifier = ClassificationModel.objects.filter(algorithm__tag =
'svm').order_by("date_created").last()
            if not classifier:
               print "Trained model not found..."
               return
            featuretype = classifier.featuretype
            filepath = classifier.model.path
            datasize = classifier.datasize
            gram_features,tweets = BaseExtractor.get_relevant_features(datasize,int(featuretype))
            be = BaseExtractor(gram_features)
            for text in processedData:
```

```
            bigrams = BaseExtractor.n_grams(text, int(featuretype))
            fv = be.extract_relevant_features(bigrams)
            feature_vectors.append(fv.values())
        classifier = svm_load_model(filepath)
        p_labels, p_accs, p_vals = svm_predict([0] * len(feature_vectors), feature_vectors,
classifier)
        count = 0
        correct = 0
        for text in newtweets:
            found = False
            polarity = p_labels[count]
            if(polarity == 1):
                polarity = 'positive'
                found = True
                positive_count +=1
            elif(polarity == -1):
                polarity = 'negative'
                found = True
                negative_count+=1
            elif(polarity == 0):
                polarity = 'neutral'
                found = True
                neutral_count+=1
            count += 1
            res = {}
            if found:
                res["tweet"] = text['Tweet']
                res['polarity'] = polarity
                res['correct'] = text['Correct']
                if test:
                    if res['correct'] == res['polarity']:
                        correct +=1
                        if polarity=='positive': correct_positive +=1
                        if polarity=='negative': correct_negative +=1
                        if polarity=='neutral': correct_neutral +=1
                BaseExtractor.save_classified_tweets(text['Tweet'],polarity,"svm")
            else:
                res["tweet"] = text['Tweet']
                res['polarity'] = "unclassified"
                # if test: res['correct'] = text['Correct']
                res['correct'] = text['Correct']
            tweetsl.append(res)
        response['tweets'] = tweetsl
        response['positives'] = positive_count
        response['negatives'] = negative_count
        response['neutrals'] = neutral_count
```

```
            print str(e)
from nltk.util import ngrams
import nltk
from extractor.models import *
from extractor.text_parser import Parser
class BaseExtractor(object):
    """docstring for BaseExtractor"""
    def __init__(self, relevant_features):
        super(BaseExtractor, self).__init__()
        self.relevant_features = relevant_features
    @staticmethod
    def n_grams(tweet_text, n):
        try:
            if n==1:
                stopwords = Stopword.objects.values_list("word",flat=True)
                splitted = [e.lower() for e in tweet_text.split() if len(e) >= 3 and e not in stopwords]
            else:
                splitted = [e.lower() for e in tweet_text.split() if len(e) >= 3]

            NGRAMS = ngrams(splitted, n)
        except Exception, e:
            print e
        gramlist = nltk.FreqDist(NGRAMS)
        return gramlist.keys()
    def extract_relevant_features(self,gram_features):
        featurelist = self.relevant_features
        features = {}
        for feature in featurelist:
            features[feature] = (feature in gram_features)
        return features
    @staticmethod
    def get_relevant_features(datasize,featuresize):
        data = TrainingData.objects.all()[:datasize]
        if data:
            relevant_features = []
            tweets = []
            for entry in data:
                bigrams = BaseExtractor.n_grams(entry.text, int(featuresize))
                relevant_features.extend(bigrams)
                e = (bigrams, entry.polarity.name)
                tweets.append(e)
            return relevant_features,tweets
    @classmethod
    def save_classified_tweets(cls,tweet,polarity,algo_tag):

        polarity = PolarityCategory.objects.get(name = polarity)
        algorithm = Algorithm.objects.get(tag = algo_tag)
```

```
        tweet = Tweet.objects.filter(cleaned_text = tweet)[0]
        obj,created = Classification.objects.get_or_create(
                tweet = tweet,
                polarity = polarity,
                algorithm = algorithm
            )
    @classmethod
    def clean_data(cls,datalist):
        newlist = []
        for text in datalist:
            cleaned_text = Parser.process(text)
            obj,created = Tweet.objects.get_or_create(original_text = text)
            obj.cleaned_text = cleaned_text
            obj.save()
            newlist.append(cleaned_text)
        return newlist
    @classmethod
    def save_results(cls,algorithm, positive,negative,neutral,featuretype):
        obj, created = Result.objects.get_or_create(
                algorithm = algorithm,
                featuretype = featuretype
            )
        obj.negative = negative
        obj.positive = positive
        obj.neutral = neutral
        obj.save()
from django.db import models
from django.utils import timezone
class UploadedFile(models.Model):
    file_obj = models.FileField(upload_to='uploaded_files/%Y/%m/%d')
    date_uploaded = models.DateTimeField(default = timezone.now)
    processed = models.BooleanField(default = False)
class PolarityCategory(models.Model):
    name = models.TextField()
    label = models.IntegerField()
    def __str__(self):
        return self.name
class TrainingData(models.Model):
    text = models.TextField()
    polarity = models.ForeignKey(PolarityCategory)
    def __str__(self):
        return self.text
class TestData(models.Model):
    text = models.TextField()
    polarity = models.ForeignKey(PolarityCategory)
    def __str__(self):
```

```
        return self.text
class Algorithm(models.Model):
    name = models.TextField()
    tag = models.CharField(max_length = 50)
    STATUSES = (
        ('trained', 'Trained'),
        ('training', 'Training'),
        ('untrained', 'Untrained')
    )
    status = models.CharField(max_length = 10,choices = STATUSES,default = 'untrained')
    def __str__(self):
        return self.name
class Result(models.Model):
    algorithm = models.ForeignKey(Algorithm)
    featuretype =  models.IntegerField(default = 2)
    positive = models.IntegerField(default = 0)
    negative = models.IntegerField(default = 0)
    neutral = models.IntegerField(default = 0)
class Tweet(models.Model):
    original_text = models.TextField()
    cleaned_text = models.TextField(null=True,blank=True)
    def __str__(self):
        return self.slang
class Stopword(models.Model):
    word = models.TextField(null=True,blank=True)
    def __str__(self):
        return self.word
```

CHAPTER 7.

7.0 REFERENCES

Luciano Barbosa and Junlan Feng. 2010. Robust sentiment detection on twitter from biased and noisy data. Proceedings of the 23rd International Conference on Computational Linguistics: Posters, pages 36–44

Michael Gamon. 2004. Sentiment classification on customer feedback data: noisy data, large feature vectors, and the role of linguistic analysis. Proceedings of the 20th international conference on Computational Linguistics

Alec Go, Richa Bhayani, and Lei Huang. 2009. Twitter sentiment classification using distant supervision.Technical report, Stanford.

Alexander Pak and Patrick Paroubek. 2010. Twitter as a corpus for sentiment analysis and opinion mining.Proceedings of LREC.

Language processing Techniques - IBM Tokyo Research Lab, 1623-14, Shimotsuruma Yamatoshi, Kanagawa-ken 242-8502, Japan nasukawa@ip.ibm.com

Bouma, G. 2009. Normalized (Pointwise) Mutual Information in Collocation Extraction. In Proceedings of theBiennial GSCL Conference 2009, pp. 31–40, Tübingen,Gunter Narr Verlag.

Gamon, M., Aue, A., Corston-Oliver, S., and Ringger, E.2005. Pulse: Mining Customer Opinions from Free Text. In Advanses of Intelligent Data Analysis IV, Lecture Notes inComputer Sciences. Berlin: Springer-Verlag, 2005.

Pak, A. and Paroubek, P. 2010. Twitter as a corpus for sentiment analysis and opinion mining. In Proceedings of LREC 2010 (pp. 1320-1326). Paris:European Language Resource Association.

Witten, I. H. and Frank, E. 2005. Data Mining: Practical Machine Learning tools and Techniques. San Francisco:Morgan Kaufmann.

N. Malandrakis, A. Potamianos, K. J. Hsu, K. N.Babeva, M. C. Feng, G. C. Davison, and S. Narayanan. 2014. Affective language model adaptation via corpus selection. In proc. ICASSP pages 4871–4874.

Andrew-retrevo blog, "is social media a new addiction?" 2010, available at:http://tinyurl.com/ydvkm4g(www.retrevo.com/content/blog/2010/03/social-media-new-addiction%3f.

Clinton watts, "foreign fighters: how are they being recruited? Two imperfect recruitmentmodels,"2008,availableat:http://www.homelandsecurity.org/hsireports/internet_radicalization.pdf.

Cambria, e., schuller, b., xia, y., havasi, c.: new avenues in opinion mining and sentiment analysis. Ieee intelligent systems 28(2), 15–21 (2013).

David carr, "how obama tapped into social networks' power," new york times-online,9november2008,availableat:http://www.nytimes.com/2008/11/10/business/media/10carr.html.

Eklund, p., hoang, a. (2002), a performance survey of public domain machine learning algorithms technical report, school of information technology, griffith university.

Facebook.com,"facebookads,"n.d.,availableat:http://www.youtube.com/t/press_statistic.

Facebook.com,"statistics,"2011,availableat:https://www.facebook.com/press/info.php?statistics.

Homeland security institute, "the internet as a terrorist tool for recruitment and radicalizationofyouth,"2009,availableat:http://www.homelandsecurity.org/hsireports/internet_radicalization.pdf.

John mathiason, "patterns of powerlessness among urban poor: toward the use of mass communications for rapid social change," comparative international development (1972): 64–84.

Jia, c. Yu, andw. Meng , “the effect of negation on sentiment analysis and retrieval effectiveness”, in proceedings of cikm,2009.

Liu, h. And h. Motoda (2001),instance selection and constructive data mining, kluwer, boston.

U.s.congress,"h.r. 1955 [110th congress 2007–2008] violent radicalization and homegrownterrorismpreventionactof2007,"2007,availableat:http://www.govtrack.us/congress/bill.xpd?bill=h110-1955.

Weich selbraun, a., gindl, s., scharl, a.: extracting and grounding context-aware sentiment lexicons. Ieee intelligent systems 28(2), 39–46 (2013.

Xia, r., zong, c., hu, x., cambria, e.: feature ensemble plus sample selection: a comprehensive approach to domain adaptation for sentiment classification. IEEE intelligent systems 28(3), 10–18 (2013).

Youtube.com, "statistics," n.d., available at: http://www.youtube.com/t/press_statistics