Sentiment analysis with YouTube

Instructor: Dr. Shonda L. Bernadin

Course: Natural Language Processing (EEL 5930 – 04)

Name : [Amol More] [aam15k]

[Sourindu Chatterjee] [sc15s]

Date : December 08, 2016.

Objectives :

The main objective of this project is to study how to apply NLP techniques to process real time data. Sentiment Analysis is recent trend in the field of human computer interaction and artificial intelligence and there are many tools available to achieve this functionality. Another aim for choosing this project is to implement custom designed sentiment analysis tool and compare it's performance with some of the available free tools. YouTube Data API is used to fetch the comments in real time and these comments are used for sentiment analysis.

Introduction :

The sentiment analysis is a powerful tool in the modern era solely dependent on information technology which lets us analyze data in a very intimate and detailed way. SA paves for one to understand the hidden connections between the action of a user and the information presented by him to the internet. This technology is also being used to predict user action based on the information given out by the user in the form of text. Thus the comments on various social platforms like Facebook, twitter, and on other online shopping websites like amazon reveal a lot of information on the user preference and understating of a topic or product. The implementation is done using python and the nltk(Natural Language Toolkit) library and the modules used in this project.

Background :

The course of Natural Language Processing with python introduced to the concept of semantics. Analyzing linguistic information to extract necessary information has always been a tedious task when tried to accomplish manually. It increase time as well as reduces efficiency due to dependency on resources. This course helped to understand how to automate processing of raw data in textual format and use it for application.

Some concepts from NLP which are used in this project are as follows :

Word_tokenization : This is a method which helps to convert sentence formatted data into a list of words based on spacing. This makes easier to generate an indexed formatted dataset for complex processes.

Tagging : This concept helps to identify parts of speech related to a word in a text. Tagging is based on different tagsets. Usually tagset named universal is used for consistent analysis. There is a type of tagging called NGram tagging which a contextual tagging method which helps to find out the exact meaning of the text. This type tagging is dependent on tags of previous words in a sentence. As the value of N increases for a NGram tagger, the accuracy of the result set increases but it increase time of execution.

Classifier : It is a tool trained with labelled input dataset to identify patterns in a text and classify it into a label used during training. It is a supervised type of classification.

Program Description :

NLTK :

NLTK (Natural Language Tool Kit) is an open source platform to analyze and study concepts of Natural Language Processing with the help of python programming language. NLTK offers many corpora along with inbuilt libraries to perform tasks such as word tokenization, part of speech tagging etc. NLTK also supports graphical representation and categorization of linguistic data. It also supports research fields related to Human Computer Interaction and Artificial Intelligence.

YouTube DATA API Setup :

Nowadays many applications or websites use videos for conveying a better information. All these platforms cannot download and show video each time of execution. To avoid interaction between different advanced platforms and YouTube information, YouTube developer's team has created multiple APIs which are compatible with most of the platforms for exchanging information. YouTube Data API is one of those development platforms which has the ability to transfer process control to and fro in different platforms.

YouTube Developer's help document contains all information for different processes such as getting video ids, channels, playlists, number of subscriptions, reading comments, posting comments, deleting threads along with tags required for them. This code is freely available to use after completion of necessary mentioned in help document.

For getting access to YouTube API which is used here, first project/app needs to be registered on GDC (Google Developer's Console) using google account credentials. After registration, process of authorization needs to be completed which generated a client id and a access key. This generated key is termed as developer's key and is essential part of access authorization process for all the requests flowing through designed modules to YouTube servers.

The already existing techniques used for sentiment analysis in this project are described below.

Vader :

Vader is a simple and effective tool for sentiment analysis for social media contexts. It is open sourced under the [MIT License]. Vader stands for Valence Aware Dictionary and sEntiment Reasoner. It is a lexicon and rule based sentiment analysis tool. , Vader is useful to identify the polarity and intensity of sentiments in social media text. While using any linguistic data analytical tool, there is requirement of lexicon feature set. But manually creating such a huge amount of dataset is time consuming. This pushes for using preexisting word banks. This polarity analysis tool uses already developed word banks for Linguistic Inquiry and Word Count(LIWC), ANEW and General Inquirer(GI). Vader also analyses emoticons, sentiment related acronyms

and common slang expressions. Using all these word banks, around 9000 lexical feature words were used to create a feature set for Vader. This feature set contains a sentiment intensity score for each word calculated using Wisdom of Crowd approach. This score is used to calculate the sentence score in terms of Compound, Positive, Negative and Neutral. Compound represents value of sentiment intensity and other values state connotation. This is one of the best tools to use for sentiment analysis which requires no data for training. Also when performance of some other analysis techniques including vader was compared with manually sentiment analyzed datasets involving humans, vader has found out to be the most effective tool in terms of closeness to the accuracy.

This tool can be installed in system using following command :

$ pip install vaderSentiment

Textblob Polarity :

Similar to Vader, this is another open source library which is available for identifying sentence polarity. Textblob has many features such as tokenization, Noun Phrase chunkers and language converter. The sentiment Analyzer in textblob library has two different techniques to get sentence polarity. First technique is dependent on PatternAnalyzer library which is another open source library which detects contextual patterns in the input text and it is also used for data mining ,machine learning and network analysis. Based on sentiments, it returns value of sentence polarity. If the sentence polarity is greater than zero then the sentiment is positive otherwise for polarity less than zero, it is negative. Range for sentence polarity value is [-1.0 , 1.0]. Another technique is NaiveBayes Analyzer which uses a already trained naivebayes classifier. This classifier is trained with movie_review corpus from nltk. This technique returns sentiment value in the format of classification, p_pos and p_neg scores where classification gives value as pos or neg and p_pos and p_neg gives score.

This tool can be installed in system using following command :

$ pip install textblob

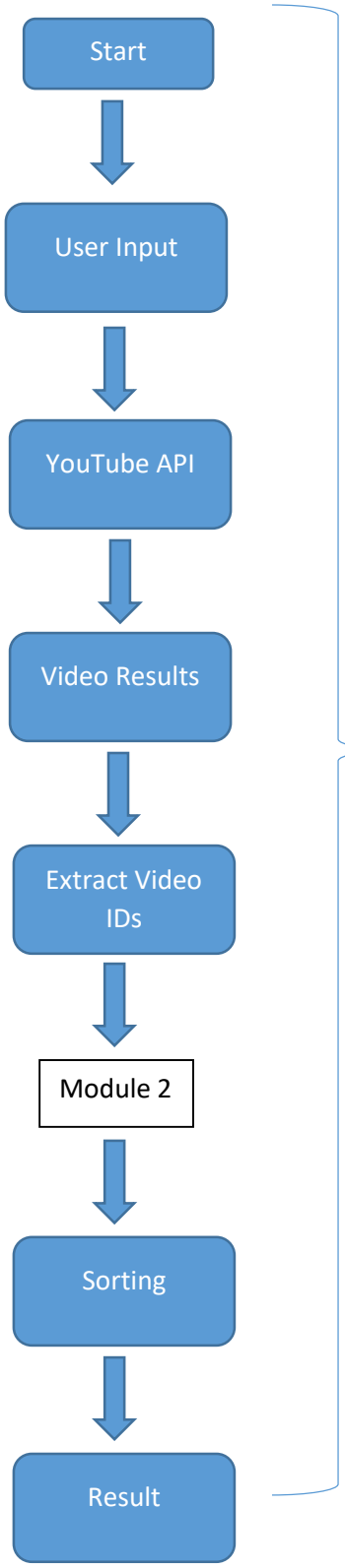
Lexicon Analysis :

This is another simple tool which is available with installation of NLTK itself. This tool uses a list of positive and negative words in the form of two different files. While measuring sentiment for a given text, this tool counts the number of positive and negative words. If any of the word in given text does not appear in the stored list of words then that word is considered as neutral. Depending count of positive and negative words, sentence sentiment value is decided.

This functionality is already available in NLTK installation package under the class nltk.sentiment.util.

Flowchart :

Following diagram displays flowchart for designed modules.

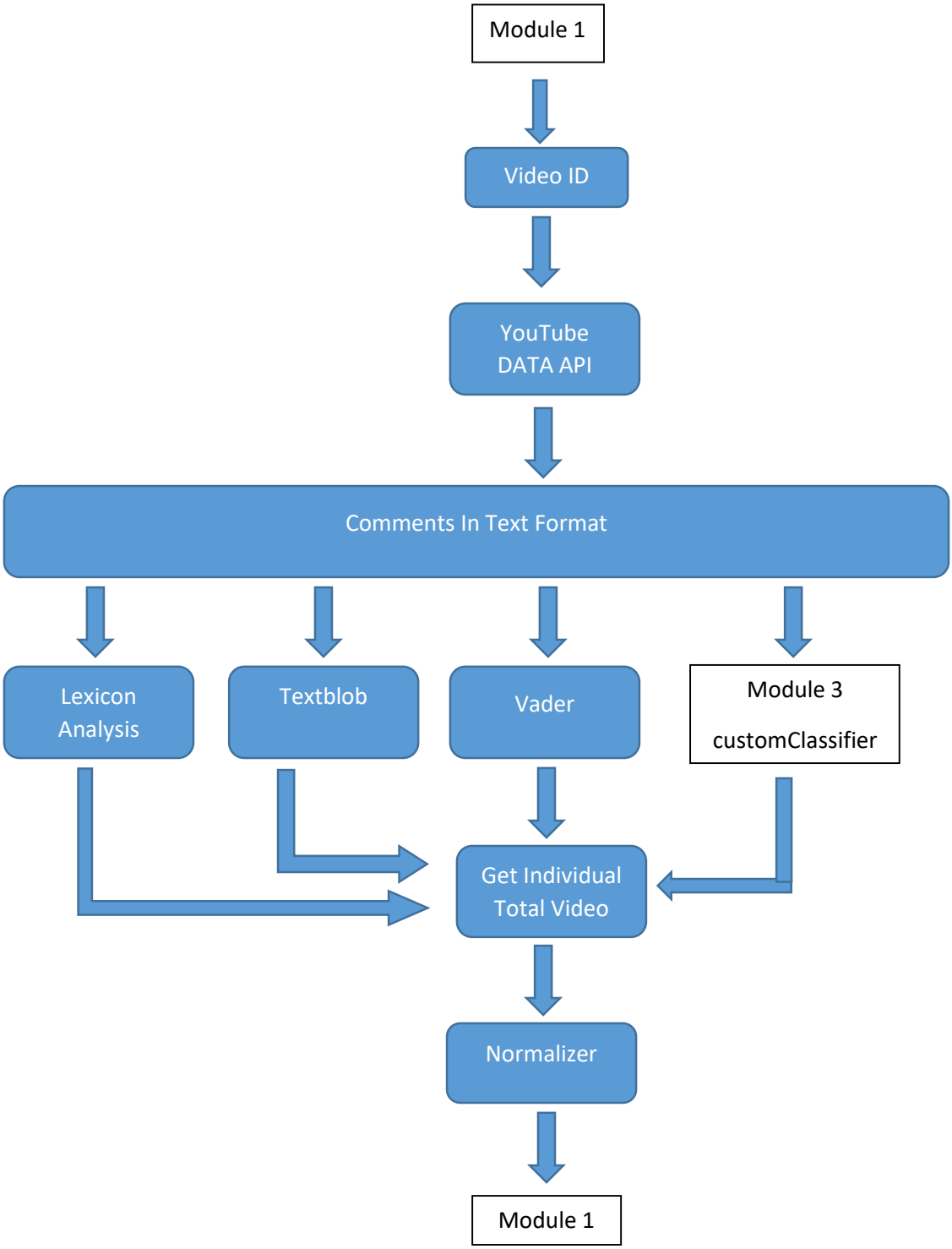


Module 1

This module is designed to get input keyword from user. User can enter any search keyword related to topic of interest. This search keyword is then passed to YouTube Data API. This API retrieves a video list related to entered keyword. From the returned result set, using specific tags, video id is retrieved. Video ID acts like an identity for a particular video using which we can extract information related to a particular id.

These video ids are then passed to module 2 for further processing.

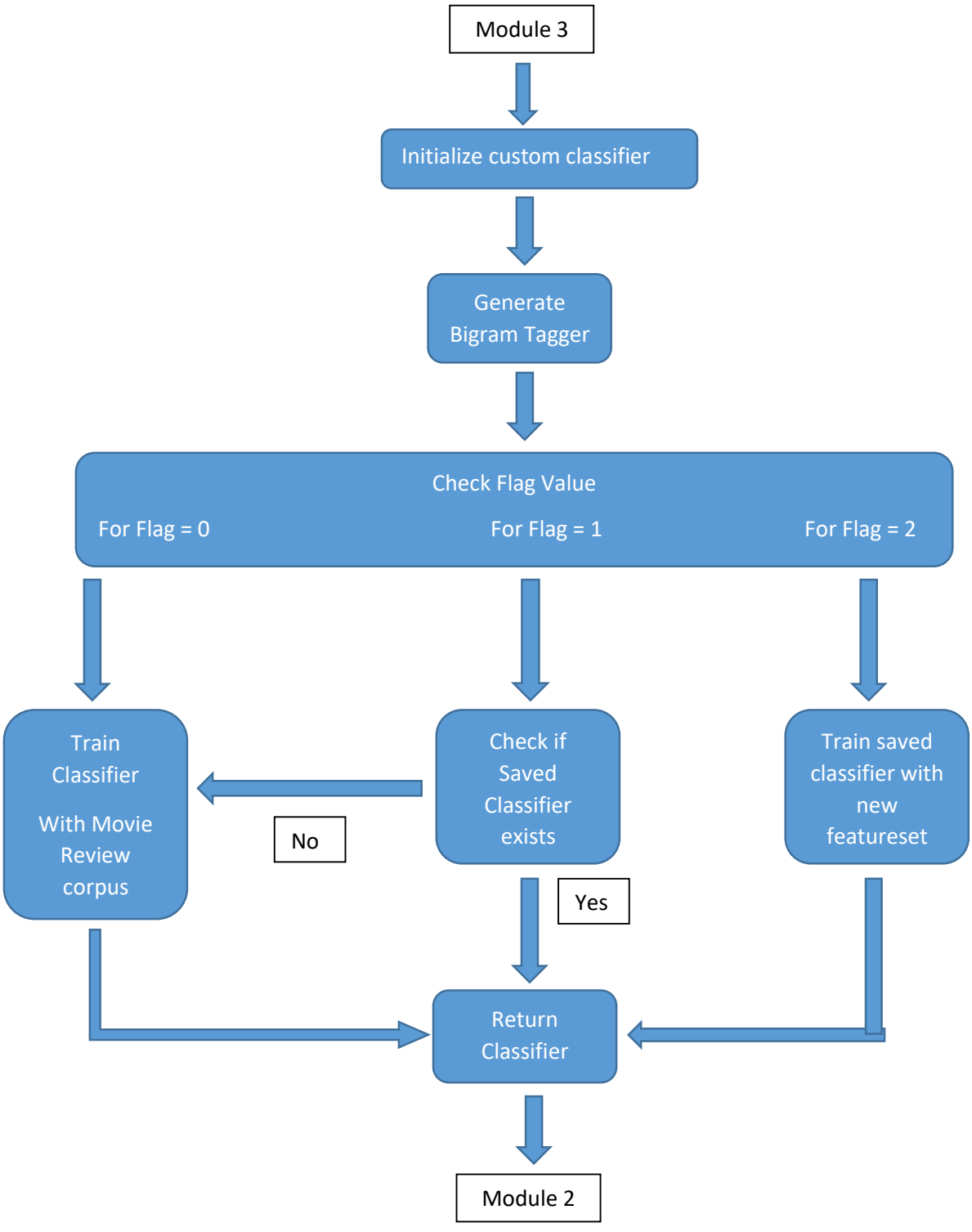
Following flow chart displays Module 2 functionality.



Module fetches comment for each video id and then applies different sentiment analysis techniques and calculates total video score for each technique. Further this score needs to be normalized since number of comments for each video may not be consistent. All calculated scores are then passed to module 1 again for sorting and displaying result.

In module 2, one of the analysis technique is using the custom designed classifier. This classifier is implemented in module 3.

Module 3 block diagram is shown below.



Module 3 creates a bigram Tagger. This tagger is further used to tag sentences based on contextual meaning. The tagged sentences are later used to train classifier. Once the classifier is created and trained then it is returned to module 2 for sentiment analysis of comments.

The detailed functionality of each module is explained below in bottom up approach.

Module 3 : custom_Classifier :

This module is used to create a naivebayes classifier to identify sentiment of a comment. First in this module a bigram tagger is created. Bigram Tagger helps for contextual tagging of words in tokenized comment text with the help of tagging of words before current word. To train Bigram Tagger, nps_chat corpus is used. All tagged words from nps_chat corpus are extracted and passed to the bigram tagger. Instead of bigram tagger, we could have used NGram tagger. But using a NGram tagger increases the execution time required by module resulting in poor performance. Also, it is rare to find such a long comment in which we can use tagging of previous words for current word tagging. Use of bigram tagger helps in creating a feature set which will be further used to create training dataset for the classifier.

Once the Bigram tagger is created, a flag value is checked. Generation of classifier depends on this flag.

For Flag = 0:

When the flag value is 0 then it means code execution is happening for the first time. Here for creating featureset, movie_review corpus is used because it has categorized documents. The documents are divided into 2 categories as positive and negative. This helps us for getting label name for feature set. Each sentence in each movie review document is first tagger using bigram tagger and then it is stored in featueset dictionary in the form of concatenated overlapped words as key and concatenated overlapped tags for these words is value for this key.

E.g. Suppose a sentence is given as “This is NLTK”.

Using Bigram Tagger , tagged output = [(“This”,”Tag1”)(“is”,”Tag2”)(“NLTK”,”Tag3”)]

Using feature extraction technique,

Feature set dictionary : featureset[“This is”] = “Tag1 Tag2”

featureset[“is NLTK”] = “Tag2 Tag3”

All movie review corpus sentences are converted into above format of dictionary with key value pair. This dictionary along with document category is then used to train NaiveBayes classifier. Once the classifier is trained and ready to use, then this classifier is saved in a file in current directory so that each time there is no need to retrain classifier with same data.

Pickle library form NLTK is used to save classifier in current directory.

For Flag = 1 :

When the flag value reads as 1 then code checks if there exists a previously trained and saved classifier. If it exists then that classifier is imported using pickle library and it will be returnd to module 2 for text classification.

If there is no previously trained and saved classifier then all the steps for flag = 0 are repeated and a new classifier is created and saved.

Pickle library saves classifier in .pickle format with all trained data and label.

For Flag = 2 :

If the flag is set to 2 while calling sentiment initializer function, then it means new dataset is available to train the previously saved classifier. Sentiment analysis using classifier is domain specific. It means data used for classifier training in one domain may not be helpful in achieving efficient result. Whenever classifier training is required by a new dataset, data should be in format of a list of tuple having first element as category of the sentence and second element as sentence. Then this sentence is again converted required overlapped words key value pair along with category.

For training of classifier with new dataset, previously trained and saved classifier is first retrieved. Then same classifier is trained with newly created dataset and again the classifier is saved in current directory to use later.

Whenever a new classifier is created or already saved classifier is being used, it again needs to be saved. Whether saving current classifier is needed or not, is again decided by a variable Is_Classifier_TrainingRequired in the module code. Whenever this variable is set to “Y” or “YY”, classifier will be saved in current directory otherwise saved classifier will be returned.

Module 2 : ProcessVideoID :

This module deals with retrieving comments related to a video id and then further processing those comments for calculating video score using 4 techniques : Vader, textblob, Lexicon Analysis and custom classifier.

YouTube Data API is the interface used for connection this module through internet to actual YouTube severs for fetching comments related to a video id. A developer key provided by Google Developer’s console is use to get authorized access to public information related to any video id. This API returns all public information available for a video id. From that result set, textual formatted comments are extracted using specific tags such as item["snippet"]["topLevelComment"] where snippet and topLevelComment are tags provided by youtube. This result set contains comment threads and all the comments under all threads. For analyzing just important information, only original comment thread is used for sentiment analysis excluding all replies lying under that thread. topLevelComment this tag returns the first

comment in any thread. Considering the time factor for execution of all functionalities within less time, maximum 1000 comments are used for all video ids.

Once the necessary comments are ready to be analyzed, each comment is passed to all 4 tools one by one and score for each tool is stored in a dictionary using name of tool as key. For the next comment again, sentiment score related to each tool is updated in the same dictionary after performing some mathematical computations to normalize the value.

Vader :

For using vader analysis tool , an object is created to call analyzer function.

For getting vader analysis score, simple text of comment is passed to internal function SentimentIntensityAnalyzer of vader libraries using the class object created. This function returns sentiment intensity scores in the format of 4 values named as compound, pos, neg and neu. Pos, neg and neu gives value for whether the input text is positive or negative or neutral and compound stands for intensity of sentiments in the input text.

textblob :

The function TextBlob of this library takes input as a simple text and returns a single value named as sentiment polarity. If this polarity value is greater than 0 then input is positive else it's a negative sentiment. There is no need of creating any class for this library.

Lexicon Analysis :

This method uses a function named as check_lexicon which is defined in the same module. Comment text here in this function is first tokenized and then each word in tokenized text is compared against two opinion word banks containing positive and negative words. All the positive and negative words appearing in input text are calculated and words not appearing in these two word banks are considered as neutral and neglected. In the end, if number of positive words is greater than negative words then result is positive sentiment else a negative.

Custom classifier :

In this part, first an initialization function from module 3 is called to import a trained classifier with the help of flag settings explained in module 3. Comment text is first tokenized and later tagged using Bigram tagger from module 3. Again this tagged text is converted into a dictionary with overlapping words as key and tags as value. This dictionary is then evaluated for sentiment with the help of imported classifier. Since the classifier is trained default with only positive and negative labels, it computes only these two sentiments for any given input.

Above process of analysis is repeated for each comment of a video id and once all the scores for all the comments are fetched, they are normalized so as to use for sorting.

Module 1 : Get Video IDs and Sorting :

This module acts as the main module for all the functionalities described. The process of execution starts here. User is prompted in real time to enter a search keyword through python shell. This keyword acts as input and gets list of videos based on parameter settings in YouTube API. Number of videos to be fetched for any keyword is controlled by a parameter named Default. Whatever integer value is stored in this variable, represents number of videos to be used for further analysis.

The result which is returned by API here contains public information related to all videos. Out of all this information, only video ids are required for further execution. Predefined tags such as ["snippet"]["title"] and ["id"]["videoId"] gives necessary information to start the further process.

Each extracted video id is then passed to module 2 to calculate a video score related to it.

Once all the video scores are returned from module 2, then they are sorted here based on sorting parameters to find out which video would be better according to each technique.

Later on in the end, best possible video id is opened in a google chrome browser of user as a part of user empathy.

Normalization and Sorting :

Normalizing a set of values is very important for comparative study and inference. In this particular project the normalized value of the sentiment score collected by using the tools defined in NLTK like VADER, Text Blob, Lexicon Polarity, Custom Classifier, is computed on the basis of the number of comment thread in each video. Thus a normalized of the sentiment score that takes care of the length of comment thread and also the length of the comment text in some context is of the utmost importance. This more so because if a video has more number of positive comment as individual sentences as compared to the number of comments being considered its value would otherwise be less if any other video which has equal or more number of positive comment text but has the number of comments double than the other video. In order to deal with scenario, we would have pave a way that generates normalized sentiment scores. The techniques used for normalizing various functions vary a little but the overall idea is the same.

Vader:

In Vader the normalization is done taking into consideration the length of the comment thread, that is the number of comments a video has or the maximum number that is being considered in this project. The equation used is given as

$$individual_{comment_{score}} = \frac{\left(individual_{comment_{score}}\right)^2}{total_{num_{comments}}}$$

$$final_{video_{score}} = final_{video_{score}} + \left(individual_{comment_{score}}\right)^{\frac{1}{2}}$$

Thus the final output is between -1 to +1 also that negative sign of the values are taken care of and the normalized value can also be negative if the non-normalized value is negative.

Text Blog:

Similarly, like the Vader normalization technique we employ the squaring and square rooting on the non-normalized and the normalized values respectively. This technique though has a catch, unlike the VADER the sentiment score of individual comments is not between -1 and +1 here few of the results can be actually +1 and -1 which are not very promising values if the final value has to kept between these two outer limits. Thus for a negative sentiment score the equation can be,

$$individual_{comment_{score}} = -1 * \frac{mod\left(individual_{comment_{score}} + 0.1\right)^2}{total_{num_{comments}}}$$

$$final_{video_{score}} = -1 * \left(mod\left(individual_{comment_{score}} + 0.1\right)^{\frac{1}{2}}\right)$$

Custom Classifier

The Normalization used for the custom designed sentiment classifier is very unique as this only returns a integer value of 1 and -1 or ‘pos’ and ‘neg’ as its output for the sentiment score of the individual comments. Thus the value should be normalized in-order for the program to generate a better response. The factors considered in this scenario is the length of each comment string and off course the total number of comments for each video. The equation for a ‘neg’ result translated to =1 is,

$$final_{video_{score}} = final_{video_{score}} - \left(\frac{1}{length(comment_{Text})}\right)$$

$$final_{video_{score}} = -1 * \frac{mod\left(final_{video_{score}}\right)^2}{total_{num_{comments}}}$$

NLTK Lexicon Score

The sentiment score generated by this similar to Custom Classifer but the underlying functioning varies a lot as compared to the later. Due to this fact the normalization technique used in this has a similar mathematical structure. The equation for a ‘pos’ or +1 would have to be,

$$final_{video_{score}} = final_{video_{score}} + \left(\frac{1}{length(comment_{Text})}\right)$$

$$final_{video_{score}} = \frac{\left(final_{video_{score}}\right)^2}{total_{num_{comments}}}$$

Results :

The results generated by the program after using four different tools like VADER, TEXT BLOB, NLTK LEXICON ANALYZER, CUSTOM CLASSIFIER are quite impressive. The program as explained earlier perfoms sentiment analysis of the youtube comments set by a search keyword initially. In this example the search keyword used is ‘lamborgini huracan’ the favourite sports car. The table of the generated score data is as follows :

Video ID	Compound	Positive	Negative	Neutral	textblob_polarity	lexicon_score	classifier_score
eJnWPhSQjPs	0.2086335299833977	0.17691071194249375	0.441933762005122	0.2443240505285279	0.23449563187944256	0.004109336726070479	0.07705540912904814
fYpjmc2MyII	0.3798415726764691	0.11285222638477276	0.5703683020645519	0.2454928546142773	0.2941404047538776	0.0058895730295214395	0.02269121165052246
LN7autjEVZo	0.42524653756599173	0.17299811402910473	0.5476216605018902	0.3385520138792485	0.3546331901904446	0.014505633319792129	0.02719225302081193
pa--G4fiZ8M	0.22300362924280448	0.15632913458254585	0.45631650890858744	0.24674344516337698	0.20163749598402503	0.0037930793881991044	0.035810241450374015
PDHQNYARUAI	0.2495189063285278	0.11337457771385467	0.3867422118951311	0.28469410034145554	0.2689308922103651	0.008392789290641686	0.0758042718465564
FL6pJEwjGWw	0.3484055555909904	0.10532109633560284	0.5644439003715662	0.22229115666920563	0.36092260908699	0.0030850454404979296	0.011856220009721906
hIFOvykQxVE	0.0	0.0	0.0	0.0	0.08512565307587486	0.0	0.15701743491229814
b3S2MfIzCI0	0.1769184866447446	0.1768446823251032	0.578000987600079	0.1523756323476078	-0.16089853828134837	-0.00036313060175789484	0.006170339490418812
eAVTI9NEBjQ	0.28714155553779863	0.0876189667442691	0.36734267607598586	0.21805519179021324	0.19515211881971775	0.00040655861158399515	0.10636140822592473
22UTS-GdLdY	0.3713913897987764	0.08440591408979192	0.4296019766403936	0.2794360093712572	0.34245848940005796	0.011247127039802894	0.020255072661955167

Video ID	classifier_score	Compound	lexicon_score	Positive	Negative	Neutral	textblob_polarity
LN7autjEVZo	0.02719225302081193	0.42524653756599173	0.014505633319792129	0.17299811402910473	0.5476216605018902	0.3385520138792485	0.3546331901904446
fYpjmc2MyII	0.02269121165052246	0.3798415726764691	0.0058895730295214395	0.11285222638477276	0.5703683020645519	0.2454928546142773	0.2941404047538776
22UTS-GdLdY	0.020255072661955167	0.3713913897987764	0.011247127039802894	0.08440591408979192	0.4296019766403936	0.2794360093712572	0.34245848940005796
FL6pJEwjGWw	0.011856220009721906	0.3484055555909904	0.0030850454404979296	0.10532109633560284	0.5644439003715662	0.22229115666920563	0.36092260908699
eAVTI9NEBjQ	0.10636140822592473	0.28714155553779863	0.00040655861158399515	0.0876189667442691	0.36734267607598586	0.21805519179021324	0.19515211881971775
PDHQNYARUAI	0.0758042718465564	0.2495189063285278	0.008392789290641686	0.11337457771385467	0.3867422118951311	0.28469410034145554	0.2689308922103651
pa--G4fiZ8M	0.035810241450374015	0.22300362924280448	0.0037930793881991044	0.15632913458254585	0.45631650890858744	0.24674344516337698	0.20163749598402503
eJnWPhSQjPs	0.07705540912904814	0.2086335299833977	0.004109336726070479	0.17691071194249375	0.441933762005122	0.2443240505285279	0.23449563187944256
b3S2MfIzCI0	0.006170339490418812	0.1769184866447446	-0.00036313060175789484	0.1768446823251032	0.578000987600079	0.1523756323476078	-0.16089853828134837
hIFOvykQxVE	0.15701743491229814	0.0	0.0	0.0	0.0	0.0	0.08512565307587486

Video ID	classifier_score	Compound	lexicon_score	Positive	Negative	Neutral	textblob_polarity
eJnWPhSQjPs	0.07705540912904814	0.2086335299833977	0.004109336726070479	0.17691071194249375	0.441933762005122	0.2443240505285279	0.23449563187944256
b3S2MfIzCI0	0.006170339490418812	0.1769184866447446	-0.00036313060175789484	0.1768446823251032	0.578000987600079	0.1523756323476078	-0.16089853828134837
LN7autjEVZo	0.02719225302081193	0.42524653756599173	0.014505633319792129	0.17299811402910473	0.5476216605018902	0.3385520138792485	0.3546331901904446
pa--G4fiZ8M	0.035810241450374015	0.22300362924280448	0.0037930793881991044	0.15632913458254585	0.45631650890858744	0.24674344516337698	0.20163749598402503
PDHQNYARUAI	0.0758042718465564	0.2495189063285278	0.008392789290641686	0.11337457771385467	0.3867422118951311	0.28469410034145554	0.2689308922103651
fYpjmc2MyII	0.02269121165052246	0.3798415726764691	0.0058895730295214395	0.11285222638477276	0.5703683020645519	0.2454928546142773	0.2941404047538776
FL6pJEwjGWw	0.011856220009721906	0.3484055555909904	0.0030850454404979296	0.10532109633560284	0.5644439003715662	0.22229115666920563	0.36092260908699
eAVTI9NEBjQ	0.10636140822592473	0.28714155553779863	0.00040655861158399515	0.0876189667442691	0.36734267607598586	0.21805519179021324	0.19515211881971775
22UTS-GdLdY	0.020255072661955167	0.3713913897987764	0.011247127039802894	0.08440591408979192	0.4296019766403936	0.2794360093712572	0.34245848940005796
hIFOvykQxVE	0.15701743491229814	0.0	0.0	0.0	0.0	0.0	0.08512565307587486

The first table is a representation of the unarranged data as it is generated by the program. The second table are sorted on the basis of ‘compound’ which is an abbreviation of VADER compound score. This score reflects the polarity of the sentiment with a value for intensity. The third table is sorted on the basis of ‘lexicon_score’ which an abbreviation of the NLTK lexicon analyzer score which in short gives the result on the basis of negative and positive words in the individual comments. The third table is sorted on the basis of the VADER`s negative intensity score abbreviated as ‘neg’.

Video ID	classifier_score	Compound	lexicon_score	Positive	Negative	Neutral	textblob_polarity
hIFOvykQxVE	0.15701743491229814	0.0	0.0	0.0	0.0	0.0	0.08512565307587486
eAVTI9NEBjQ	0.10702036045765904	0.28714155553779863	0.00040655861158399515	0.0876189667442691	0.36734267607598586	0.21805519179021324	0.19515211881971775
eJnWPhSQjPs	0.0793663792293826	0.2086335299833977	0.004109336726070479	0.17691071194249375	0.441933762005122	0.2443240505285279	0.23449563187944256
PDHQNYARUAI	0.07540384381421736	0.2495189063285278	0.008392789290641686	0.11337457771385467	0.3867422118951311	0.28469410034145554	0.2689308922103651
pa--G4fiZ8M	0.035810241450374015	0.22300362924280448	0.0037930793881991044	0.15632913458254585	0.45631650890858744	0.24674344516337698	0.20163749598402503
LN7autjEVZo	0.029825307134128388	0.42524653756599173	0.014505633319792129	0.17299811402910473	0.5476216605018902	0.3385520138792485	0.3546331901904446
22UTS-GdLdY	0.022478277700986357	0.3713913897987764	0.011247127039802894	0.08440591408979192	0.4296019766403936	0.2794360093712572	0.34245848940005796
fYpjmc2MyII	0.021345365253838653	0.3798415726764691	0.0058895730295214395	0.11285222638477276	0.5703683020645519	0.2454928546142773	0.2941404047538776
FL6pJEwjGWw	0.011844075108110206	0.3484055555909904	0.0030850454404979296	0.10532109633560284	0.5644439003715662	0.22229115666920563	0.360922609086699
b3S2MfIzCI0	0.0062087452640917475	0.1769184866447446	-0.00036313060175789484	0.1768446823251032	0.578000987600079	0.1523756323476078	-0.1608985382813483

The above table represents sentiment analysis score for individual score generated by the Custom Sentiment Classifier.

Conclusion

The concept of sentiment analysis on the YouTube comments to generate a more intuitive video recommender system is disruptive idea. The project performs as promised and gives out relatively god results and can also be used in practical applications.

Future Work

The design of the Custom Classifier is very intuitive and can be a little weak in terms of execution time. Thus better implementation techniques can be employed in order to improve the performance of the Custom classifier and in turn the whole project.

Currently feature extraction is based on a single type. In near future, more number of features can be extracted and used for classifier training. Also current classifier was trained with only a single corpora. There is a function included in current implementation which trains the previously saved classifier adding more datasets to it. In near future, this new dataset training can be automated based on user input. Sorting of video scores can be made dynamic and user will be given a choice to sort result set based on his own choice and analysis technique.

Reference

1. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit, Steven Bird, Ewan Klein, Edward Loper
2. http://comp.social.gatech.edu/papers/icwsm14.vader.hutto.pdf
3. http://blog.algorithmia.com/benchmarking-sentiment-analysis-algorithms/

Appendix Code:

KeywordSearchResults_Main:

```
###!/usr/bin/python3

import httplib2;
import os;
import sys;
from CommentsTest_v24 import ProcessVideoID;
from apiclient.discovery import build;
from apiclient.errors import HttpError;
from oauth2client.tools import argparser;
from openUrl import openURL as web

##DEVELOPER_KEY = 'AIzaSyDHHWxvvvt6eojYOW0Y9O8h6JF7ScXRGi0';
DEVELOPER_KEY = 'AIzaSyDz8_XTYvc-7e_VwkJEZqfHbQQEv5WYCOU'
YOUTUBE_API_SERVICE_NAME = "youtube";
YOUTUBE_API_VERSION = "v3";

def youtube_search(options):
  youtube = build(YOUTUBE_API_SERVICE_NAME, YOUTUBE_API_VERSION,
developerKey=DEVELOPER_KEY)
  search_response = youtube.search().list(
    q=options.q,
    part="id,snippet",
    maxResults=options.max_results
  ).execute();

  videos = [];
  video_ids = [];

  for search_result in search_response.get("items", []):
    if search_result["id"]["kind"] == "youtube#video":
      videos.append("%s (%s)" % (search_result["snippet"]["title"],
search_result["id"]["videoId"]));
      video_ids.append(search_result["id"]["videoId"]);

  return video_ids;

def bubble(bad_list,keyin):
  bad_list = sorted(bad_list,key=lambda x:x[keyin],reverse=True)
  return bad_list
```

```
#########################################################################
########################################

if __name__ == "__main__":
  input_keyword = input('Enter a keyword to seearch videos : ');
  ar = input_keyword;
  #argparser.add_argument("--q", help="Search term", default="Google")
  argparser.add_argument("--q", help="Search term", default=ar)
  argparser.add_argument("--max-results", help="Max results", default=10)
  args = argparser.parse_args()

  trial_if_error = 0

  video_Table = {'Video ID' : [],'classifier_score': [],'compound' : [],'neg' : [],'neu' : [],'pos'
: [],'textblob_polarity': [],'lexicon_score': []}
  video_Table2 = []
  while(trial_if_error < 3):
    try:
      video_ids = youtube_search(args);
      for x in video_ids:
          print()
          print('The video is :',x)
          video_Score = ProcessVideoID(x)

          video_Table['Video ID'].append(x)

          temp = [x]

          for y in sorted(video_Score.keys()):
            video_Table[y].append(video_Score[y])
            temp.append(video_Score[y])

          video_Table2.append(temp)

      break

    except HttpError as e:
      print ("An HTTP error %d occurred:\n%s" % (e.resp.status, e.content))
      trial_if_error = trial_if_error +1

##  print(video_Table)
  print()
```

```
  print()
  heading_print = "{:<10} {:<20} {:<22} {:<15} {:<26} {:<20} {:<20} {:<20}";

  value_print = "{:<10} {:<20} {:<22} {:<20} {:<24} {:<20} {:<20} {:<20}";
  #value_print = '{:20.5f} {:20.5f} {:20.5f} {:20.5f} {:20.5f} {:20.5f} {:20.5f} {:20.5f}'

  print (heading_print.format('Video
ID','Compound','Positive','Negative','Neutral','textblob_polarity','lexicon_score','classifier
_score'));

  for x in zip(video_Table['Video
ID'],video_Table['compound'],video_Table['neg'],video_Table['neu'],video_Table['pos']

,video_Table['textblob_polarity'],video_Table['lexicon_score'],video_Table['classifier_sc
ore']):
    print (value_print.format(x[0],x[1],x[2],x[3],x[4],x[5],x[6],x[7]))

  #print(video_Table2)
  print()
  print()
## sorting the table on the basis of Compound
  video_Table2 = bubble(video_Table2,2)
  print('Sorted Tableon the basis of VADER "compound"..')
  print (heading_print.format('Video
ID','classifier_score','Compound','lexicon_score','Positive','Negative','Neutral','textblob_p
olarity'));
  for x in video_Table2:
    print (value_print.format(x[0],x[1],x[2],x[3],x[4],x[5],x[6],x[7]))

  web(video_Table2[0][0])

  #print(video_Table2)
  print()
  print()
## sorting the table on the basis of lexicon_score
  print('Sorted Tableon the basis of "lexicon score"..')
  video_Table2 = bubble(video_Table2,4)

  print (heading_print.format('Video
ID','classifier_score','Compound','lexicon_score','Positive','Negative','Neutral','textblob_p
olarity'));
  for x in video_Table2:
    print (value_print.format(x[0],x[1],x[2],x[3],x[4],x[5],x[6],x[7]))
```

```
  print()
  print()
## sorting the table on the basis of negativity
  print('Sorted Tableon the basis of VADER "negative"..')
  video_Table2 = bubble(video_Table2,6)

  print (heading_print.format('Video
ID','classifier_score','Compound','lexicon_score','Positive','Negative','Neutral','textblob_p
olarity'));
  for x in video_Table2:
    print (value_print.format(x[0],x[1],x[2],x[3],x[4],x[5],x[6],x[7]))


## sorting the table on the basis of Custom Classifier
  print('Sorted Tableon the basis of VADER "Custom Classifier"..')
  video_Table2 = bubble(video_Table2,1)

  print (heading_print.format('Video
ID','classifier_score','Compound','lexicon_score','Positive','Negative','Neutral','textblob_p
olarity'));
  for x in video_Table2:
    print (value_print.format(x[0],x[1],x[2],x[3],x[4],x[5],x[6],x[7]))

  web(video_Table2[0][0])
```

CommentsTest_v24:

```
from apiclient.errors import HttpError;
from apiclient.discovery import build;

import nltk;

from textblob import TextBlob;
from nltk.corpus import opinion_lexicon;
from nltk.tokenize import treebank;

from nltk.sentiment.vader import SentimentIntensityAnalyzer;
sid = SentimentIntensityAnalyzer();

#from Sentiment_Analyzer_Design_v9 import Get_BigramTagging,
Initialize_SentimentAnalyzer;
import Sentiment_Analyzer_Design_v9 as custom
from nltk import word_tokenize;
classifier = custom.Initialize_SentimentAnalyzer(1);

import pickle;

YOUTUBE_API_SERVICE_NAME = "youtube";
YOUTUBE_API_VERSION = "v3";
##DEVELOPER_KEY = 'AIzaSyBS5zcC0yuhCfVP5mihP-Io5PfGOgNExo4';
DEVELOPER_KEY = 'AIzaSyDz8_XTYvc-7e_VwkJEZqfHbQQEv5WYCOU';

def get_comment_threads(youtube, video_id, comments):
   threads = [];
   results = youtube.commentThreads().list(
   part="snippet",
   videoId=video_id,
   textFormat="plainText"
   ).execute();
   #Get the first set of comments
   for item in results["items"]:
      threads.append(item)
      comment = item["snippet"]["topLevelComment"]
      text = comment["snippet"]["textDisplay"]
      comments.append(text)

   #Keep getting comments from the following pages
   while ("nextPageToken" in results):
```

```
        results = youtube.commentThreads().list(
        part="snippet",
        videoId=video_id,
        pageToken=results["nextPageToken"],
        textFormat="plainText"
        ).execute()
        if len(threads) > 100:
            break;
        for item in results["items"]:
            threads.append(item)
            comment = item["snippet"]["topLevelComment"]
            text = comment["snippet"]["textDisplay"]
            comments.append(text)

    return threads;

##########################################################################
###########

def check_lexicon(sentence):
    """
    Basic example of sentiment classification using Liu and Hu opinion lexicon.
    This function simply counts the number of positive, negative and neutral words
    in the sentence and classifies it depending on which polarity is more represented.
    Words that do not appear in the lexicon are considered as neutral.

    :param sentence: a sentence whose polarity has to be classified.
    :param plot: if True, plot a visual representation of the sentence polarity.
    """
    tokenizer = treebank.TreebankWordTokenizer()
    pos_words = 0
    neg_words = 0
    tokenized_sent = [word.lower() for word in tokenizer.tokenize(sentence)]

    x = list(range(len(tokenized_sent))) # x axis for the plot
    y = []

    for word in tokenized_sent:
        if word in opinion_lexicon.positive():
            pos_words += 1
            y.append(1) # positive
        elif word in opinion_lexicon.negative():
            neg_words += 1
```

```
            y.append(-1) # negative
        else:
            y.append(0) # neutral

    if pos_words > neg_words:
        return 'Positive'
    elif pos_words < neg_words:
        return 'Negative'
    elif pos_words == neg_words:
        return 'Neutral'

    if plot == True:
        _show_plot(x, y, x_labels=tokenized_sent, y_labels=['Negative', 'Neutral',
'Positive'])

##############################################################################
###########
##if __name__ == "__main__":
def ProcessVideoID(video_id):
  youtube = build(YOUTUBE_API_SERVICE_NAME, YOUTUBE_API_VERSION,
developerKey=DEVELOPER_KEY)

  try:
    comments = [];
    Total_textblob_polarity = 0;
    classifier_score = 0;
    com_Lex_Score = 0
    video_comment_threads = get_comment_threads(youtube, video_id, comments);
    comlen = len(comments);

    print("Total Threads: %d" % len(video_comment_threads));
##  video_comment_threads = get_comment_threads(youtube, '-8N9UR6OTCs',
comments);

    total_video_score =
{'compound':0.0,'neg':0.0,'neu':0.0,'pos':0.0,'textblob_polarity':0.0,'lexicon_score':0.0,'clas
sifier_score':0.0};



    for thread in video_comment_threads:
        try:
            comment = thread["snippet"]["topLevelComment"];
```

```
        text = comment["snippet"]["textDisplay"];
##        print(text);
########################################## textblob
#########################################################

        textblob_polarity = ((TextBlob(text)).sentiment).polarity;
        #print(textblob_polarity);
        if(textblob_polarity < 0):
           if(textblob_polarity == -1):
              textblob_polarity = -1*(abs(textblob_polarity+0.1)**2)/comlen
           else:
              textblob_polarity = -1*(abs(textblob_polarity)**2)/comlen
        else:
           if(textblob_polarity == 1):
              textblob_polarity = ((textblob_polarity-0.1)**2)/comlen
           else:
              textblob_polarity = (textblob_polarity**2)/comlen

        Total_textblob_polarity = Total_textblob_polarity + textblob_polarity;


########################################## nltk lexicon
#########################################################
        lexicon_result = check_lexicon(text)
        if lexicon_result == 'Positive':
           com_Lex_Score = com_Lex_Score+(1/len(text))
        elif lexicon_result == 'Negative':
           com_Lex_Score = com_Lex_Score-(1/len(text))
        else:
           None
        #print(com_Lex_Score);


########################################## vader
#########################################################

        ss = sid.polarity_scores(str(text));
##        print(ss);
        for k in sorted(ss):

           if(ss[k] == 1):
              ss[k] = 0;
```

```
            if(ss[k] < 0):
               total_video_score[k] = total_video_score[k]-(ss[k]**2)/comlen
            else:
               total_video_score[k] = total_video_score[k]+(ss[k]**2)/comlen

########################################### custom classifer
#############################################################

         tokenized_comment = word_tokenize(text);
         tgd = custom.Get_BigramTagging(tokenized_comment);
         t=[(w,k) for w,k in tgd];

         f={};
         for i in range(len(t)-1):
            f[t[i][0]+ ' ' + t[i+1][0]] = t[i][1]+ ' ' + t[i+1][1];
         classifier_result = classifier.classify(f);
         if classifier_result =='pos':
            classifier_score = classifier_score + (1/len(text));
         else:
            classifier_score = classifier_score - (1/len(text));


      except UnicodeEncodeError as e2:
         None;
      except ZeroDivisionError as e3:
         None;

   total_video_score['textblob_polarity']=Total_textblob_polarity;

   total_video_score['lexicon_score']= com_Lex_Score;

   total_video_score['classifier_score'] = classifier_score

   for k in ['compound','neg','neu','pos','textblob_polarity']:
      if(total_video_score[k] < 0):
         total_video_score[k] = -1*(abs(total_video_score[k])**(1/2));
      else:
         total_video_score[k] = (abs(total_video_score[k])**(1/2));
      print('{0}: {1}, '.format(k, total_video_score[k]), end='')

   for k in ['lexicon_score','classifier_score']:

      if(total_video_score[k] < 0):
```

```
            total_video_score[k] = -1*(abs(total_video_score[k])**2)/comlen
        else:
            total_video_score[k] = (total_video_score[k])**2/comlen
        print('{0}: {1}, '.format(k, total_video_score[k]), end='')

    print()
    return total_video_score

##  print(total_video_score);

  except HttpError as e:
    print ("An HTTP error %d occurred:\n%s" % (e.resp.status, e.content));

##ProcessVideoID('cYVL3LkPBXA');
```

Sentiment_Analyzer_Design_v10:

```
import nltk, random, getpass, os, pickle;
from nltk.corpus import nps_chat;
from nltk.corpus import brown;
from nltk import word_tokenize;
from nltk.corpus import movie_reviews as movies

default_sentence_set = [];
new_sentence_set = [];
Is_Classifier_TrainingRequired = "N";
classifier_training=[];

posts = nltk.corpus.nps_chat.xml_posts();
featuresets = [nltk.pos_tag(word_tokenize(post.text)) for post in posts];
t0= nltk.DefaultTagger('NN');
t1= nltk.UnigramTagger(featuresets, backoff=t0);
t2= nltk.BigramTagger(featuresets, backoff= t1);

print('global execution')

##text = word_tokenize("I am good");
##print(t2.tag(text));
##print(text);

def Get_BigramTagging(text):
   tagged_text = t2.tag(text);
   return tagged_text;

def Initialize_SentimentAnalyzer(flag=0):
   global Is_Classifier_TrainingRequired;
   if flag == 0:
      Default_Dataset();
      default_dataset = default_sentence_set;
      for i in range(len(default_dataset)):
         label = default_dataset[i][0];
         sent = default_dataset[i][1];
         tagged = t2.tag(sent);
         pairs = [(w,k) for w,k in tagged];
         feature={};
         for i in range(len(pairs)-1):
            feature[pairs[i][0]+ ' ' + pairs[i+1][0]] = pairs[i][1]+ ' ' + pairs[i+1][1];
```

```
        temp = (feature, label);
        classifier_training.append(temp);

    Is_Classifier_TrainingRequired = "Y";

elif flag == 1:
    usrname = getpass.getuser();
    fileDir = os.path.dirname(os.path.realpath('__file__'))
    filepath = os.path.join(fileDir, '../naivebayes.pickle')
    #filepath = "C:/Users/"+usrname+"/OneDrive/Project/naivebayes.pickle";
    if os.path.exists(filepath):
        classifier_f = open("naivebayes.pickle", "rb");
        classifier = pickle.load(classifier_f);
        classifier_f.close();
        print('Classifier Loading')
        Is_Classifier_TrainingRequired = "N";
    else:
        Initialize_SentimentAnalyzer(0);

elif flag == 2:
    New_Dataset();
    new_dataset = new_sentence_set;
    for i in range(len(new_dataset)):
        label = new_dataset[i][0];
        sent = new_dataset[i][1];
        tagged = t2.tag(sent);
        pairs = [(w,k) for w,k in tagged];
        feature={};
        for i in range(len(pairs)-1):
            feature[pairs[i][0]+ ' ' + pairs[i+1][0]] = pairs[i][1]+ ' ' + pairs[i+1][1];

        temp = (feature, label);
        classifier_training.append(temp);

    Is_Classifier_TrainingRequired = "YY";

print('local execution')

if Is_Classifier_TrainingRequired == "Y":
    random.shuffle(classifier_training);
    classifier = nltk.NaiveBayesClassifier.train(classifier_training);
    save_classifier = open("naivebayes.pickle","wb");
    pickle.dump(classifier, save_classifier);
```

```
        save_classifier.close();
    elif Is_Classifier_TrainingRequired == "YY":
        random.shuffle(classifier_training);
        usrname = getpass.getuser();
        fileDir = os.path.dirname(os.path.realpath('__file__'));
        filepath = os.path.join(fileDir, '../naivebayes.pickle');
        if os.path.exists(filepath):
            classifier_f = open("naivebayes.pickle", "rb");
            classifier = pickle.load(classifier_f);
            classifier = classifier.train(classifier_training);
            classifier_f.close();
        else:
            classifier = nltk.NaiveBayesClassifier.train(classifier_training);

        save_classifier = open("naivebayes.pickle","wb");
        pickle.dump(classifier, save_classifier);
        save_classifier.close();

    else:
        None;
    print('after class execution')
    return classifier;

def Default_Dataset():
    documents = movies.fileids();
    for doc in documents:
        sents = movies.sents(doc);
        doc_label = doc[:3];
        for sent in sents:
            default_sentence_set.append((doc_label,sent));


def New_Dataset():
    documents = movies.fileids();
    for doc in documents:
        sents = movies.sents(doc);
        doc_label = doc[:3];
        for sent in sents:
            new_sentence_set.append((doc_label,sent));


####testing simple text
##a = Initialize_SentimentAnalyzer(0);
```

```
##text = word_tokenize("I am good");
##tgd = Get_BigramTagging(text);
##t=[(w,k) for w,k in tgd];
##f={};
##for i in range(len(t)-1):
##    f[t[i][0]+ ' ' + t[i+1][0]] = t[i][1]+ ' ' + t[i+1][1];
##
##print(f);
##print('input : ',text,'classification : ',a.classify(f))
##
##b = Initialize_SentimentAnalyzer(1);
##text = word_tokenize("I am not good");
##tgd = Get_BigramTagging(text);
##t=[(w,k) for w,k in tgd];
##f={};
##for i in range(len(t)-1):
##    f[t[i][0]+ ' ' + t[i+1][0]] = t[i][1]+ ' ' + t[i+1][1];
##
##print(f);
##print('input : ',text,'classification : ',b.classify(f))
```

openUrl:

```
import webbrowser

def openURL(videoID):
    try:
        url = 'https://www.youtube.com/watch?v='+videoID

        chrome_path = 'C:/Program Files (x86)/Google/Chrome/Application/chrome.exe
%s'

        webbrowser.get(chrome_path).open(url)

        return True

    except SomeError as e:
        return False


openURL('9IezBgi7S0I')
```