**Abstract**

Prediction is an abstract thought but usually under typical circumstances entities tend to follow patterns in the past. ’PreTrender’ is an application that tries to predict future of trending by analyzing data of the past. Since tweets are a direct representation of the trend at a particular point of time, PreTrender analyzes tweets over a past time period to generate a trending pattern of the unit. This trending pattern is then extrapolated to user’s expected date in the future to obtain a sentiment score. Simply put, PreTrender attempts to foretell what the trend of an entity be based on its trend behavior in the past.

1. **Introduction**

Prediction is a tentative and abstract concept that is inconceivable due to the infinite anomalies that can alter it, but ‘history tends to repeat itself’. This observation is the basis of our application ‘PreTrender’, acronym for Prediction of Trend Patterns. True to its name, Pretrender can give an inept prediction of trend tendencies in the near future of any entity with a web presence.

The application basically works on people’s perception of the entity expressed in the form of tweets over a fixed range of time in the past.

Each tweet can be associated with a sentiment score indicating the opinion of the ‘tweple’. A large set of such scores generates a trend pattern. Disregarding all anomalous scenarios, the future trend pattern is expressed an extension of the past pattern. Thus, prediction of sentiment on user entered date in the future is computed by analysis and enumeration of their opinion in the past.

The scope of this project lies in the fact that computation is closest we can get to any form of prediction. Although not 100% accurate,’ Pretrender’ would work great for entities with a regular and expressible trend pattern.

It would enhance customer experience by being one step ahead in product evaluation.

The concept in large can be used in stock prediction and solving complex computational problems.

1. **Problem Definition**

To evaluate trend patterns of specified entity via fetching real time tweets about it and assigning a sentiment score to them, over a fixed period of time, to obtain a trend graph and derive entity’s near future tendencies on a specific date using curve extrapolation of the obtained trend pattern in the past.

The client application is simple with all complex computation occurring at the server end.

1. **Literature Survey**

Survey involved a thorough analysis of twitter-4j library and its tweet handling methods from twitter4j official documentation.

Stanford-nlp website and github project (open source) gave an insight in NLP libraries and how different modules annotate, tokenize, parse and associate each sentence with a sentiment.

Research involved studying Lagrange’s interpolation theorem that is used to implement graphical polynomial extrapolation.

Android development was matriculated from developers. android.

Paper by - G.Vinodhini (Assistant Professor, Department of Computer Science and Engineering, Annamalai University) & RM.Chandrasekaran (Professor, Department of Computer Science and Engineering, Annamalai University) on the topic ‘Sentiment Analysis and Opinion Mining’ was one of our sources.

1. **Project Requirement Definition**

Project requires the user to be on the same network as the server.

The application takes name of the entity for which prediction is to be made along with the date of prediction. Data is transferred over an active internet connection to the server that in turn needs to have access to twitter database. After computation, results are sent and displayed on the client application.

1. **System Requirements Definition**

Some of the system requirements for the projects were:

* A twitter handle with unique set of developer keys for real time tweet queries.
* Twitter-4j library to support various tweet handling methods.
* Ejml - 0.23 library to support parsing and tokenizing of data obtained.
* Stanford-corenlp-models library to support, filter and analyze tweets in different languages.
* Stanford-corenlp library to obtain sentiment score for sentences.
* Reg-Ex package for filtering of tweets.
* Lagrange’s interpolation algorithm for sentiment prediction via polynomial extrapolation of trend graph.
* Java.net package for socket programming and interconnection between client and server.
* Android Studio for development of client side application.
* NetBeans (Any IDE) for server side development.

Multi-threading to speed up the process of data retrieval and score computation.

1. **System Design**

* The scope of the project begins with retrieval of real time tweets of a given topic from the twitter server.
* This data is then mapped to a sentiment analyzer which computes a sentiment score for every tweet that has been fetched.
* A prediction analyzing algorithm then estimates a future sentiment score of the topic based on a curve of the sentiment scores against an attribute such as the dates of the tweets when they first appeared.
* The interface used is an Android application which takes as input a topic and a date and gives as output the sentiment score (degree of goodness/badness) of that topic in the future.
* The importance of the results lies in an increased awareness among users about a particular brand based on the sentiment score accumulated for the brand over a period of time by hundreds of users. This directly results in a greater concern among brands about customer satisfaction as tweets about it would depend on how satisfied the customers are with it.

**7. Pseudo code**

The project is divided into five parts, real time tweet fetching, sentiment core attachment, sentiment constriction, predictive extrapolation and finally the android application. Each part is implemented by a core piece of code.

-*Starting from the beginning the real implementation of real time tweet fetching happens here*;

public class tweetquery

{LinkedHashSet<String> comm;

public void setup(int numberOfTweets, String topic)

{comm = new LinkedHashSet<String>();

Twitter twitter = new TwitterFactory().getInstance();

Query query = new Query(topic);

//Adding Date

query.setSince("2015-09-29");

query.setUntil("2015-11-18");

long lastID = Long.MAX\_VALUE;

ArrayList<Status> tweets = new ArrayList<Status>();

while (tweets.size() < numberOfTweets)

{if (numberOfTweets - tweets.size() > 100) {

query.setCount(100);

} else {query.setCount(numberOfTweets - tweets.size());

}

try { QueryResult result = twitter.search(query);

tweets.addAll(result.getTweets());

System.out.println("Gathered " + tweets.size()+"0" + " tweets");

for (Status t : tweets)

{if (t.getId() < lastID) {

lastID = t.getId(); } }

if (result.nextQuery() == null) {

break; }

} catch (TwitterException te) {

System.out.println("Couldn't connect: " + te); }

query.setMaxId(lastID - 1);

}

for (int i = 0; i < tweets.size(); i++) {

Status t = (Status) tweets.get(i);

if (t.getLang().equals("en")) {

comm.add(t.getText());}

-*Each tweet is attached with a sentiment score using Stanford core nlp libarary*;

public static void init()

{

Properties props = new Properties();

props.setProperty("annotators", "tokenize, ssplit, parse, sentiment");

pipeline = new StanfordCoreNLP(props);

}

public static float findSentiment(String tweet) {

float mainSentiment = 0;

int count = 0, length = 0;

if (tweet != null && tweet.length() > 0) {

int longest = 0;

Annotation annotation = pipeline.process(tweet);

for (CoreMap sentence : annotation.get(CoreAnnotations.SentencesAnnotation.class)) {

Tree tree = sentence.get(SentimentAnnotatedTree.class);

int sentiment = RNNCoreAnnotations.getPredictedClass(tree);

String partText = sentence.toString();

mainSentiment += sentiment \* partText.length();

length += partText.length();

++count;

}

}

return mainSentiment / length;

}

}

-*Thousands of tweet scores are constricted to get an average pattern;*

public float[] compute(float[]arr) throws IOException {

float[] lol = new float[30];

float acc = 0,count = 0;

int temp = 0;

int len = arr.length/30;

for(int i = 0 ; i < arr.length ;++i)

{ acc += arr[i];

++count;

if(count == len)

{ lol[temp] = acc/len;

acc = 0;

count = 0;

++temp; }

if(temp == 29)

break; }

count = 0;

acc = 0;

int dam = len\*30;

while(dam<arr.length)

{ acc += arr[dam];

++dam;

++count; }

lol[29] = acc/count;

//System.out.println("LAST VALUE + "+lol[29]);

return lol;

}

-*Predictive algorithm works Lagrange’s interpolation;*

while(loop\_var<get)

{

f11=p1.get\_out(day,value,loop\_var);

f12=p1.get\_out(day,value,loop\_var+0.01);

f13=(f12-f11)/0.01;

for(int iz=0;iz<count-1;iz++)

{ ijk=store1[iz]-f13;

store2[iz]=100\*ijk;

}

min=(int)store2[0];

minpos=0;

int getx1=0;

int getx2=1;

for(int iz=0;iz<count-1;iz++)

{

if(store2[iz]<min && iz >100 && iz<1800 && iz!=prevx)

{

//System.out.println("previous : "+prevx+" current : "+iz);

min=(int)store2[iz];

minpos=iz;

prevx=iz;

getx1=iz;}

getx2=getx1;

}

System.out.println("Loop variable : "+loop\_var+"Matching with pos : "+minpos+" and value : "+store1[minpos]);

getfinal1=store1[minpos];

getfinal2=store1[minpos+1];

finalx=finalx+(getfinal2-getfinal1);

loop\_var=loop\_var+0.01;

f114=p1.get\_out(day,value,28);

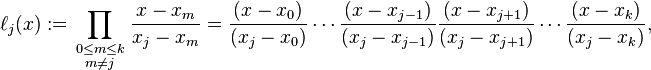
System.out.println("\n\nThe final result is : "+finalx);

}

Given a set of *k* + 1 data points, where no two  are the same, the **interpolation polynomial in the Lagrange form** is a linear combination

L(x) := \sum_{j=0}^{k} y_j \ell_j(x)

of Lagrange basis polynomials



where 0\le j\le k. Note how, given the initial assumption that no two x_i are the same, x_j - x_m \neq 0, so this expression is always well-defined. The reason pairs x_i = x_jwith y_i\neq y_j are not allowed is that no interpolation function L such that y_i = L(x_i) would exist; a function can only get one value for each argument x_i. On the other hand, if also y_i = y_j, then those two points would actually be one single point.

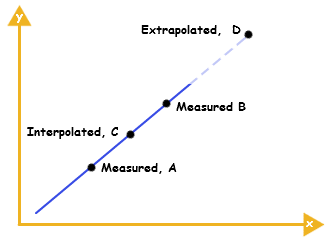
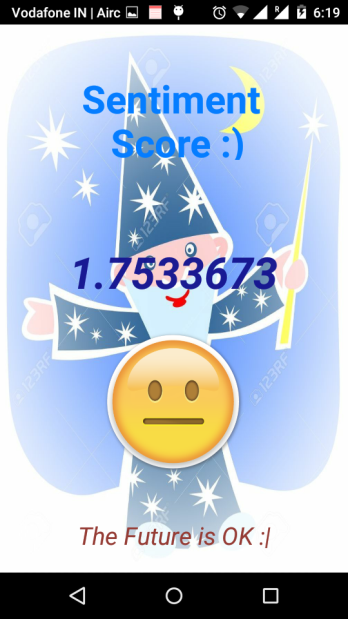


Fig. Example of Graph Extrapolation using Lagrange’s Interpolaion.

*-Application has a simple interface and basically uses async task to run background network connection;*

protected String doInBackground(String... uri)

{

try{

DatagramSocket clientSocket = new DatagramSocket();

InetAddress IPAddress = InetAddress.getByName("192.168.43.22");

byte[] sendData = new byte[1024];

byte[] receiveData = new byte[1024];

sendData = (topic+"~"+dat).getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9891);

clientSocket.send(sendPacket);

DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

clientSocket.receive(receivePacket);

score = new String(receivePacket.getData());

}

catch(Exception e)

{

Toast.makeText(getApplicationContext(),"Not Able To Send", Toast.LENGTH\_LONG).show();

android.util.Log.e("err","",e);

}

return "Not Worked";

}

**8. Results Discussion**

We tested the correctness of the tweets fetched from the twitter web server after several executions of the code on different input. The input was varied between trending topics on twitter, about which a lot of data was available in the servers and topics which were not very popular among twiterratti. The tweets received in the output belonged to the topic specified in the input and the number of tweets fetched were in accordance with the number asked for. Thus the software gave very satisfying results in this regard with an accuracy of about 95%.

In the testing for the calculation of sentiment of the tweets, we manually checked a large number of tweets for their sentiment (tone/semantic meaning) from a human perspective against the sentiment as calculated by the software. Apart from a few cases of highly ingrained sarcasm etc in some tweets, the sentiment as calculated adhered to what was intended in the tweet by its author.

For the prediction of the sentiment score Lagrange’s Interpolation algorithm gave an accuracy of up to --%. When tested on a large range of numbers as the input, the algorithm gave good results for inputs close to the realistic points in the curve (for whom the sentiment values were known). Thus the accuracy of the prediction values is higher in the more recent time intervals rather than for points of time much later (farther away than the realistic points of the graph).

**9. Conclusion**

After a considerable amount of learning and research about the project we were finally able to develop the software. We developed code to fetch real time data from Twitter about the user mentioned entity, filter out the unwanted elements of the tweets and determine correctly the sentiment associated with each tweet. Then we successfully constrict ed thousands of sentiment scores to obtain a trending pattern using which we were successfully able to extrapolate and predict sentiment on user entered date. We also developed an Android application that implements the interface of the software.

**10. Further Enhancements**

Pretrender is an application that makes predictions disregarding anomalies, thus it has a large scope of improvement in terms of efficiency:-

• Tweets obtained are filled with gibberish text thus project requires a superior tweet filtering mechanism.

• Elevating efficiency of tweet analysis by taking into account sarcasm, metaphors and slang.

• Heavy computation on server side can be made faster by using multi-processing and executing them on different cores.

• Enhancing competence polynomial extrapolation by considering more points in obtained trend pattern for accurate graph tracing.

• Adding more features in android application such as cache, history and exception handling.

**11. Bibliography**

We made references to the following topics for the completion of the project –

* Paper by - G.Vinodhini (Assistant Professor, Department of Computer Science and Engineering, Annamalai University) & RM.Chandrasekaran (Professor, Department of Computer Science and Engineering, Annamalai University) on the topic ‘Sentiment Analysis and Opinion Mining’ was one of our sources.
* Twitter4j library documentation – http://twitter4j.org/javadoc/
* Stanford-core nlp library- **http://nlp.stanford.edu/software/corenlp.shtml**
* Lagrange’s interpolation algorithm – https://en.wikipedia.org/wiki/Lagrange\_polynomial
* Android developers documentation - http://developer.android.com/index.html