CS541 – Project Report

SENTIMENT ANALYSIS OF PRESIDENTIAL CANDIDATES USING TWITTER STREAM DATA

- Shreepad Patil & Vignesh Gowri Manoharan

Overview :

Twitter is a main medium of conversing many topics which are of public interest. US Presidential election is one such event, for which people, news media and the political parties use twitter platform to share opinions, converse, canvass and much more.

We want to capture and analyze the sentiment for each candidate from the data that is generated on Twitter in real time. Use streaming engines to compute the sentiment in real time .

And also compare the sentiments with prediction poll numbers for each candidate.

Technologies used and Architecture:

Back-end :

Kafka Producer - To fetch tweets from Twitter streaming API and converts tweets to Avro format, which is a data serialization system that produces a compact,binary and fast data format.

Kafka Consumer – Receives the tweets from Producer and invokes spark streaming engine.

Spark Streaming Engine - Forms Resilient Distributed Datasets(chards) of tweets , to perform quick in-memory computation on the chards created.

Stanford CoreNLP Library – Is invoked for each of the RDD of tweets created and each tweet is processed for sentiment analysis and categorized one out of 5 possible category – Very Positive, Positive, Neutral, Negative, Very Negative.

MongoDB - Aggregate count in each category for each RDD object is calculated and stored in MongoDB in JSON format along with time stamp.

User Interface:

A client-server model has been used to implement this project. Any javaScript based browser can be used by the user to access the server. UI is built in HTML5 and javaScript. Ajax calls are made to get the data and displayed in the form of graphs. The data is fetched from two types of sources, i.e. DB server and Huffington Post website(scraped for election polls).

Resources used:

OpenUI5: This is an Open Source JavaScript UI library, maintained by SAP and available under the Apache 2.0 license. OpenUI5 lets you build enterprise-ready web applications, responsive to all devices, running on almost any browser of your choice. It's based on JavaScript, using JQuery as its foundation and follows web standards. It eases your development with a client-side HTML5 rendering library including a rich set of controls and supports data binding to different models (JSON, XML and Odata).

plotly.js: Built on top of d3.js and stack.gl, plotly.js is a high-level, declarative charting library. plotly.js ships with 20 chart types, including 3D charts, statistical graphs, and SVG maps.

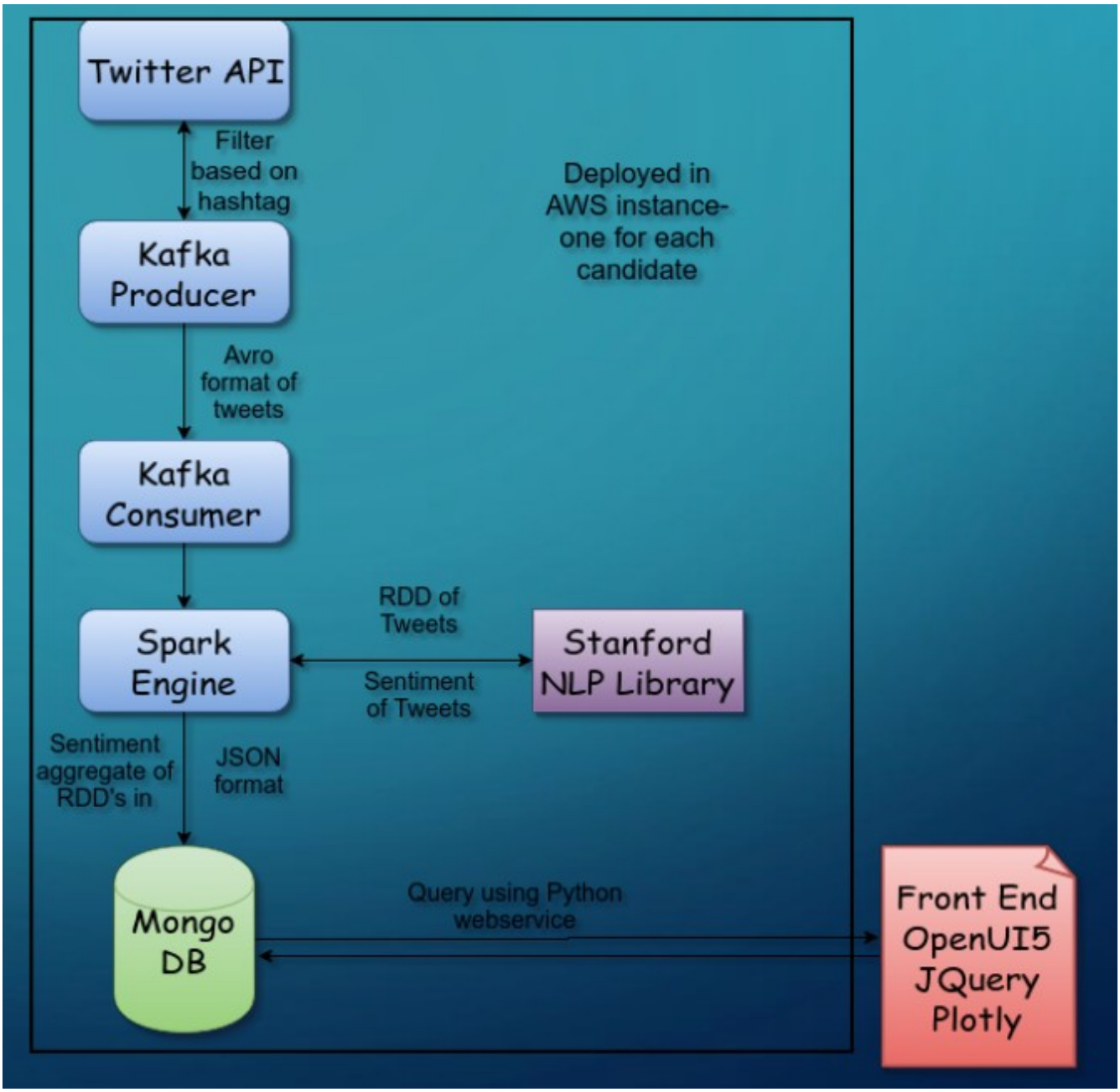
Integration: The UI queries the database using REST API calls with a GET request and receives the data as an HTTP response.

python-eve: Eve is an open source Python REST API framework. It allows to effortlessly build and deploy highly customizable, fully featured RESTful Web Services.

Eve is powered by Flask, Redis, Cerberus, Events and offers support for both MongoDB and SQL backends [*].

In the UI, we have given four categories for the user to select.

1. **Live Data –** This view shows live stream of data for the users to see the current sentiments for each candidate. Here we have shown a comparative graphs for the candidates from a same party. This view has two subviews, one for each party.
2. **Aggregate Data** - This view shows the aggregate sentiment data for each candidate. We have also shown a comparison of the aggregate sentiments for the candidate with his/her competitor in the party. There are four sub-views here, one for each candidate.
3. **Time Period –** In this view, we have given the user an option to view the data for a particular time period – To date and From date – this could be used to get the data for a particular time-period(like Super Tuesday). An option to select a particular candidate has also been given to the user.
4. **Poll Data –** In this last view, we have shown the comparison of sentiment vs the poll prediction from Huffington Post. Four sub-views are provided, one for each candidate, to the user.


Twitter API
Filter based on hashtag
Deployed in AWS instance- one for each candidate
Kafka Producer
Avro format of tweets
Kafka Consumer
RDD of Tweets
Spark Engine
Stanford NLP Library
Sentiment of Tweets
Sentiment aggregate of RDD's in
JSON format
Query using Python webservice
Mongo DB
Front End OpenUI5 JQuery Plotly

Closer look at Technologies used for real-time Sentiment analysis

Kafka

- Its a high throughput messaging system.
- Fast – It can handle hundreds of MB of read/write per second
- Durable and distributed by design.
- Scalable horizontally and can be designed as clusters

Why not Flume

- We initially used flume to stream data from Twitter , but it needs a data sink. Spark streaming engine should get data out of the sink to perform real-time computation.
- Kafka pulls data , so each consumer has and manages its own read pointer. So it allows large number of consumers of each kafka queue, that pull data at their own pace.
- Few things we missed out by not using flume, is the support for content based event routing and pre-built collectors.
- Kafka just provides the messaging, which is simple and enough for our requirements.

Spark(and why not Storm)

- In-memory distributed data analysis platform, targeted at speeding up batch analysis jobs
- Use of RDD's – Resilient Distributed Datasets, which are great for pipelining parallel operators for computation and are immutable.
- Spark streaming is designed to do micro-batching, so we get near-real-time computation.
- It performs data-parallel computation as opposed to task-parallel computation used in Storm systems, that gives real-time computation.
- There are few downsides for spark streaming too. Data is not backed up in persistent storage like HDFS, so it might just miss some data if it doesn't have enough memory to invoke the real-time computation methods for the RDD's
- It's not good for huge volume of data and complex multi-stage algorithms that has to be performed in real-time , since the time it takes to write data between stages is large
- Spark writes shuffle output to an OS buffer cache. If the task is large, it spends a lot of time writing output to an OS buffer cache.
- Given its downsides, for our project with just a single library for sentiment analysis and not so heavy stream data volume, spark was a good choice, because of its relatively simple setup and rich external libraries support when compared with Storm.

Challenges

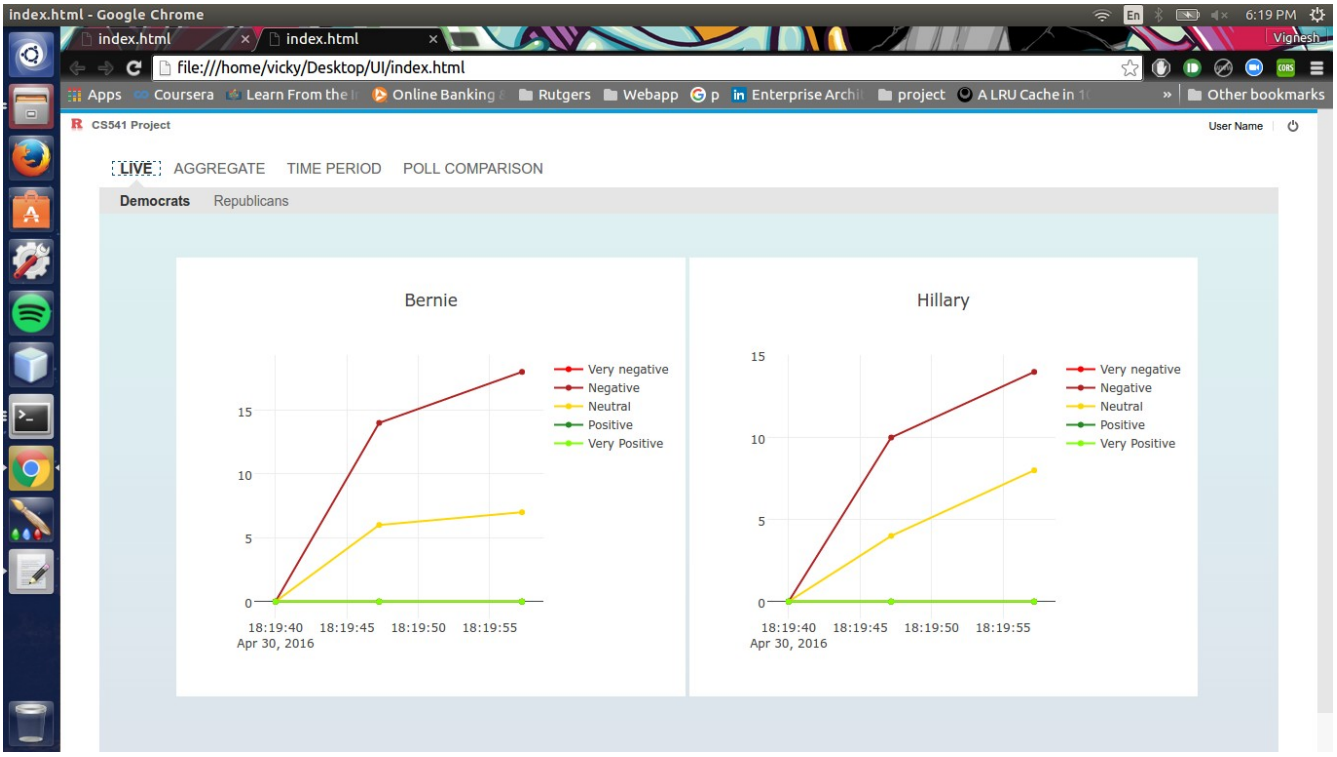
- Integration of Kafka and Spark. Initially we used just spark streaming engine to test and get used to coding with the libraries and used bag-of-words model for sentiment analysis
- Then we moved on to Flume along with Spark using a HDFS sink, where coding got a little complex and it was hard moving in and out of HDFS sink.
- We decided to use Kafka(suggested by a friend working as Data Engineer), and there was no more complication of using HDFS.
- Still writing code for RDD needed some hands-on and transforming Dstream into Data Frame was tricky and had to take care to maintain schema properly.
- Usage of Scala for Kafka and use Java libraries and methods inside,were tricky and feels awkward.
- We couldn't pull of an ensemble model for sentiment analysis even though we put in significant amount of effort on it.
- Deploying all the entire setup in AWS was pretty challenging and educational ,like pushing a system for production.
- Taking raw data from DB and processing it so that it could be plugged into plotly was a mojor challenge in UI side

Things we could have done Differently in Retrospective

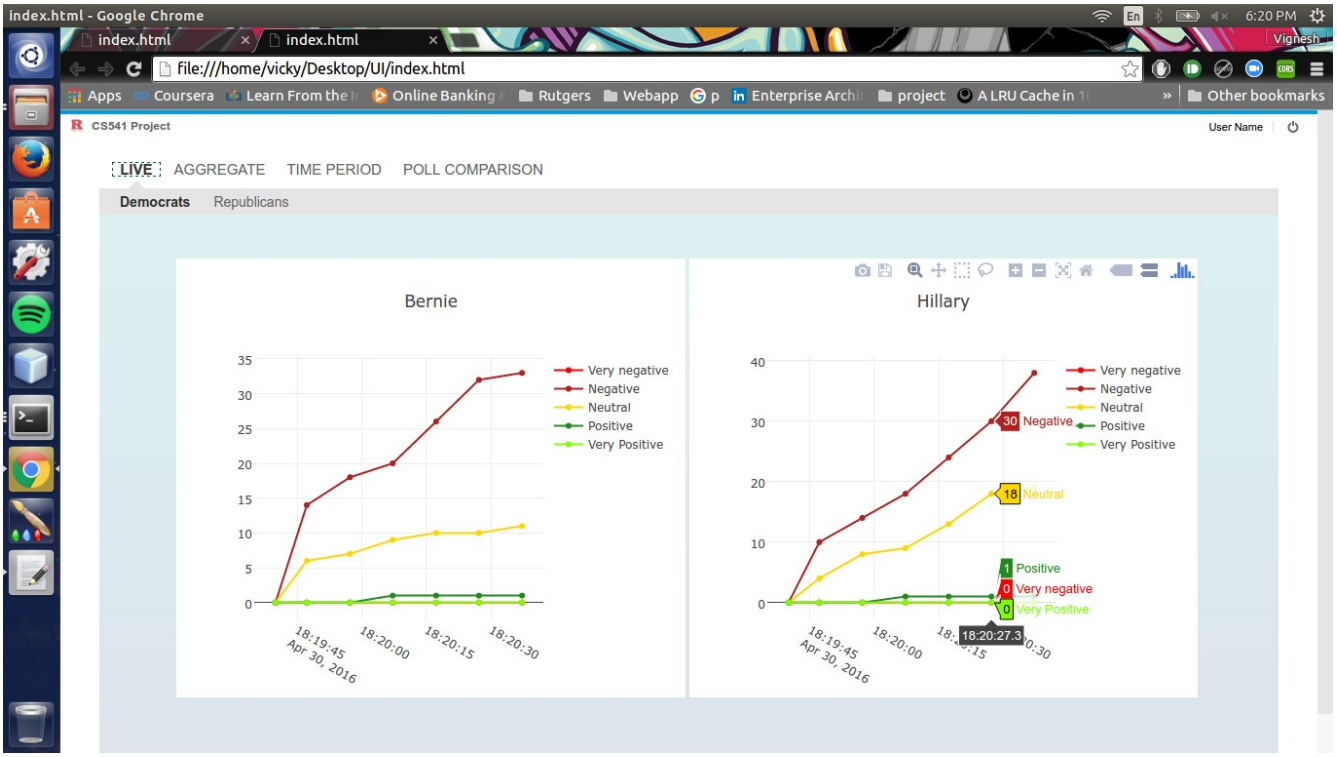
- Use of other NLP libraries for sentiment analysis. CoreNLP was choosen because we had prior experience using it, but it was not so good for computing sentiments for Tweets.
- Use a traditional DB instead of MongoDB, since we felt we could have used some SQL operations like Joins,aggregates to better projection in UI.
- Create artificial stream of twitter data, we collected across two months, to get results for a longer time-period to be shown in demo.
- Deploy a cluster setup in Kafka using zookeeper, to show fault tolerance and scalability, as it would have been an additional feature and wouldn't have needed much effort since we had most of the setup.

Screen shots of Demo :

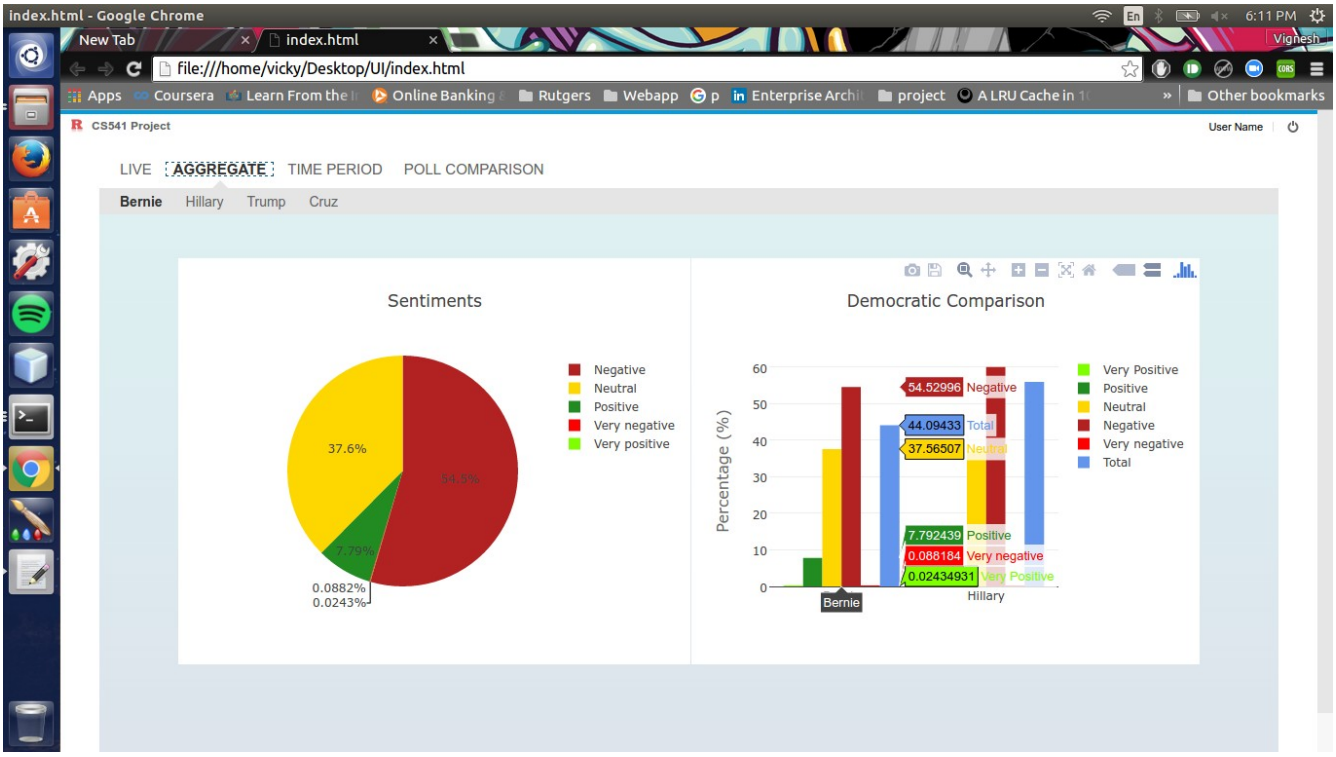
Live Data :



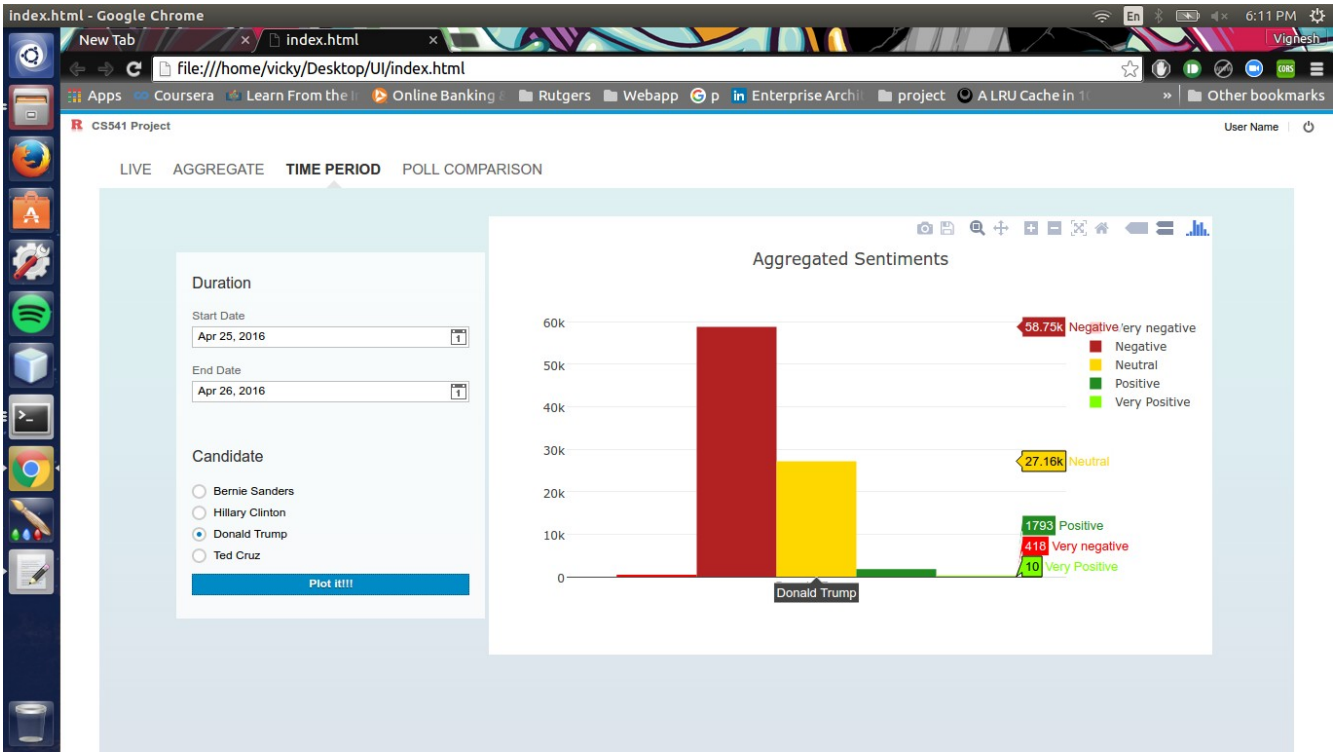
After few seconds...



Aggregate Data :



Time Period:



Poll Data Comparison: