

Project Report

1. Project description

(1) Area: social media analytics

(2) Topic: Twitter sentiment analysis

Our web app is designed for searching tweets and delivering sentiment analysis using key words as input.

(3) Business case:

Business popularity analysis; market prediction; targeted promotion...

2. Dataset

(1) For training experiment in Azure ML Studio, we used a labeled dataset Sentiment140 which comprises approximately 1,600,000 automatically annotated tweets.

(a) Dataset Description

Each instance in the data set has 6 fields:

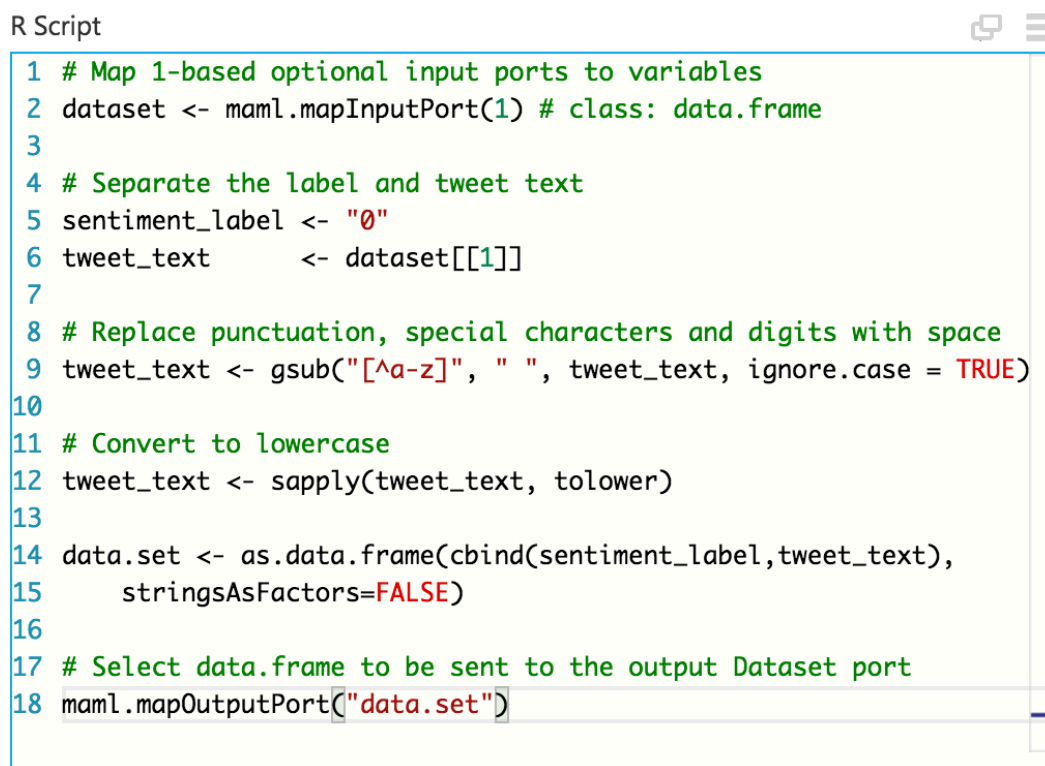
- sentiment_label - the polarity of the tweet (0 = negative, 2 = neutral, 4 = positive)
- tweet_text - the text of the tweet
- tweet_id - the id of the tweet
- time_stamp - the date of the tweet (UTC)

- target - the query (lyx). If there is no query, then this value is NO_QUERY.
- user_id - the user who posted the tweet

We used the first two fields – tweet_text, sentiment_label – for training models in Azure Studio.

(b) Data Preprocessing and Cleaning

- We used the R code in **Execute R Script** module to remove punctuation marks, special characters and digits, and then performed case normalization.



```

R Script
1 # Map 1-based optional input ports to variables
2 dataset <- maml.mapInputPort(1) # class: data.frame
3
4 # Separate the label and tweet text
5 sentiment_label <- "0"
6 tweet_text      <- dataset[[1]]
7
8 # Replace punctuation, special characters and digits with space
9 tweet_text <- gsub("[^a-z]", " ", tweet_text, ignore.case = TRUE)
10
11 # Convert to lowercase
12 tweet_text <- sapply(tweet_text, tolower)
13
14 data.set <- as.data.frame(cbind(sentiment_label, tweet_text),
15                           stringsAsFactors=FALSE)
16
17 # Select data.frame to be sent to the output Dataset port
18 maml.mapOutputPort("data.set")
  
```

- We used the **Metadata Editor** module to change the: we marked the text column as non-categorical column and marked the text column as a non-feature. Because we want the learner to ignore the source text and not use it as a feature when training

the model, but rather to use the extracted features that we build in the next step.

Column

Selected columns:
Column names: tweet_text

Launch column selector

Data type
String

Categorical
Make non-categorical

Fields
Clear feature

(2) For predictive experiment, we used the real-time twitter data source as the web service input.

We used twitter4j.jar to get real-time tweets and old tweets within 7 days by keyword. Also, we can get a several days' tweets by setting since date and until date.

Firstly, we got authentication by inputting OAuth access code and Token code

```
ConfigurationBuilder cb = new ConfigurationBuilder();
ModelAndView mv = new ModelAndView();
System.out.println("321");
cb.setDebugEnabled(true)
    .setOAuthConsumerKey("eUL89IkTvO54eMTeNmUzXr2EK")
    .setOAuthConsumerSecret("7tSH8eSvufxjwgtgy9RLRgClDIHOx0Bpot0fh46goXe5zIdtdc3")
    .setOAuthAccessToken("2692099357-tSArR3QAD6ML1Io27URWLFpQ23NAtkkcdus8q64")
    .setOAuthAccessTokenSecret("psoNY7VqcKVom6hiidmKdJWNnQGAXnLau40Zd1ZTo2sii");
TwitterFactory tf = new TwitterFactory(cb.build());
Twitter twitter = tf.getInstance();
```

After authentication, we can use getTweets() method to get today's tweets, and store them as web service input format.

```
String res = "{\"Inputs\": {\"input1\": {\"ColumnNames\": [\"tweet_text\"], \"Values\": [";  
for (int i = 0; i < list.size(); i++) {  
    String str = list.get(i);  
    str = str.replaceAll("[^\\w\\s]+", " ");  
  
    System.out.println(str);  
    res += "\"" + str + "\", ";  
}  
res = res.substring(0, res.length() - 1);  
res += "]]}, \"GlobalParameters\": {}}";
```

The web service API used like following

```
JSONArray predictOutput = null;  
String Twitters;  
String finalAnswer = "not Valid";  
String restUrl = "https://ussouthcentral.services.azureml.net/workspaces/94a7bddd924402681345";  
JSONObject user = new JSONObject();  
JSONObject input = new JSONObject();  
Twitters = res;  
  
String userInput = Twitters;  
System.out.println("*****Connect to AZURE Start*****");  
HttpPostReq httpPostReq = new HttpPostReq();  
HttpPost httpPost = httpPostReq.createConnectivity(restUrl);  
predictOutput = httpPostReq.executeReq(userInput, httpPost);
```

3. Azure Studio – Model & Web service

(1) Compare Different Models.

(a) Two-class Support Vector Machines (SVMs)

SVMs are supervised learning models that analyze data and recognize patterns. They are good for large feature sets.

We used the **Feature Hashing** module to represent variable-length tweet_text as equal-length numeric feature vectors.

Feature Hashing

Target column(s)

Selected columns:

Column names: tweet_text

Launch column selector

Hashing bitsize

17

N-grams

2

Then we used the **Filter Based Feature Selection** module to select a compact feature subset from the exhaustive list of extracted hashing features. The aim is to reduce the computational complexity without affecting classification accuracy.

Filter Based Feature Selection

Feature scoring method

Chi Squared



Operate on feature co...

Target column

Selected columns:

Column names:


sentiment_label

Launch column selector

Number of desired features

20000


Below is the performance metrics with threshold set as 0.5.

True Positive	False Negative	Accuracy	Precision	Threshold		AUC
51339	12661	0.776	0.763	0.5		0.855
False Positive	True Negative	Recall	F1 Score			
15980	48020	0.802	0.782			
Positive Label	Negative Label					
4	0					

(b) Two-Class Logistic Regression

We also tried this module for training model. It predicts the probability of occurrence of an event by fitting data to a logistic function.

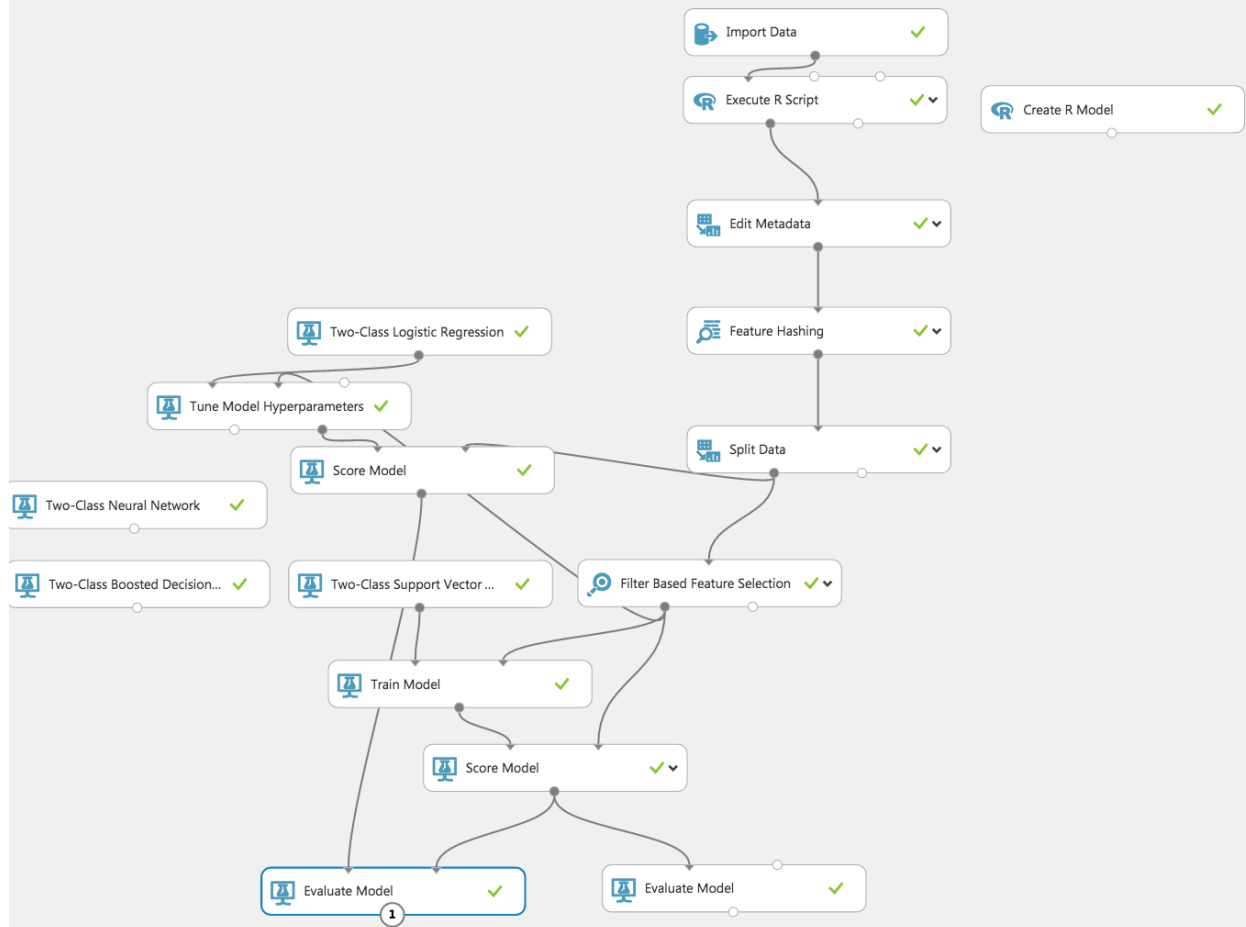
Below is the performance metrics with threshold set as 0.5.

True Positive	False Negative	Accuracy	Precision	Threshold		AUC
51339	12661	0.776	0.763	0.5		0.855
False Positive	True Negative	Recall	F1 Score			
15980	48020	0.802	0.782			
Positive Label	Negative Label					
4	0					

It worked not as good as the SVMs.

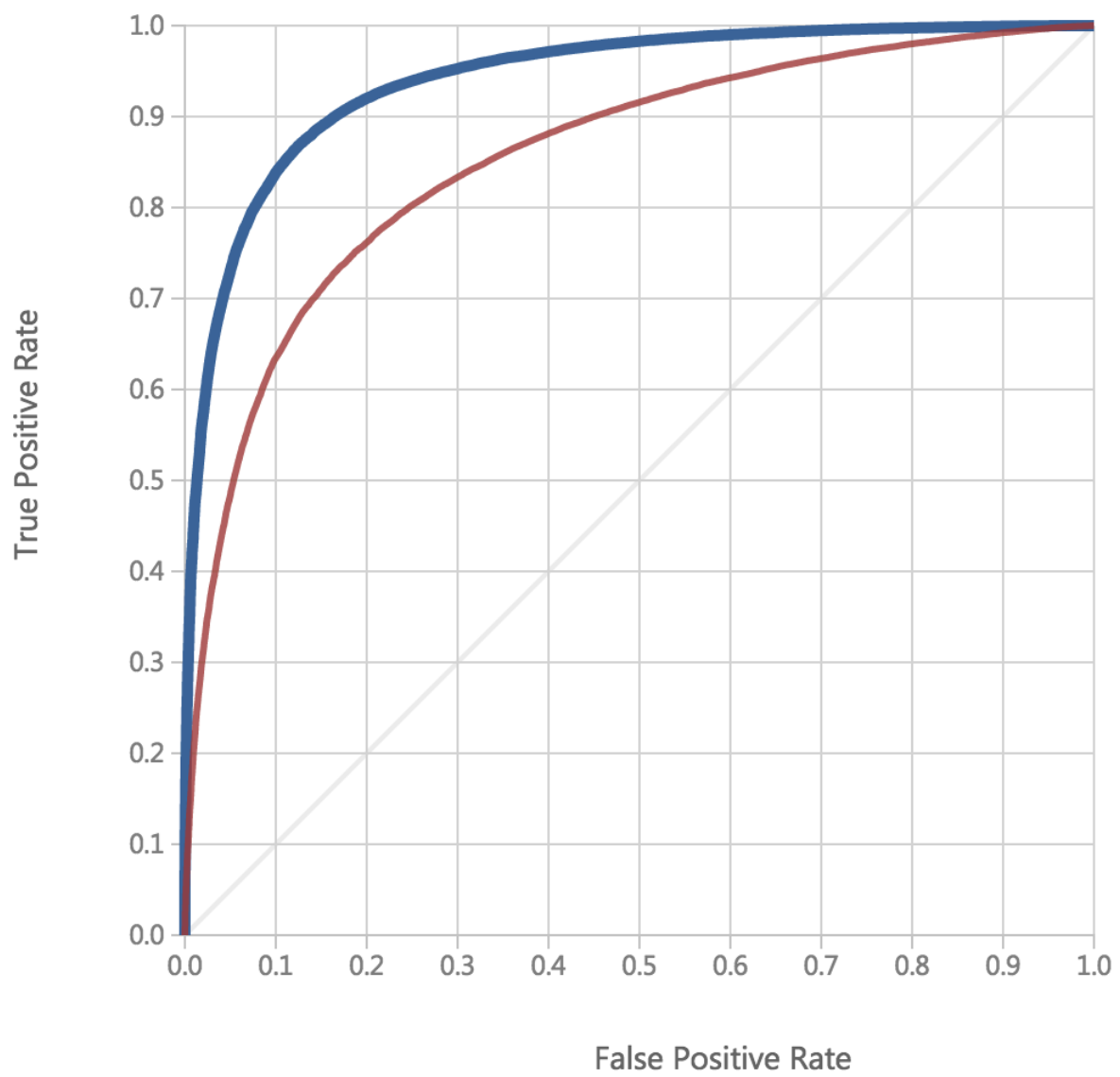
Sentiment Analysis - Copy2

Finished running ✓

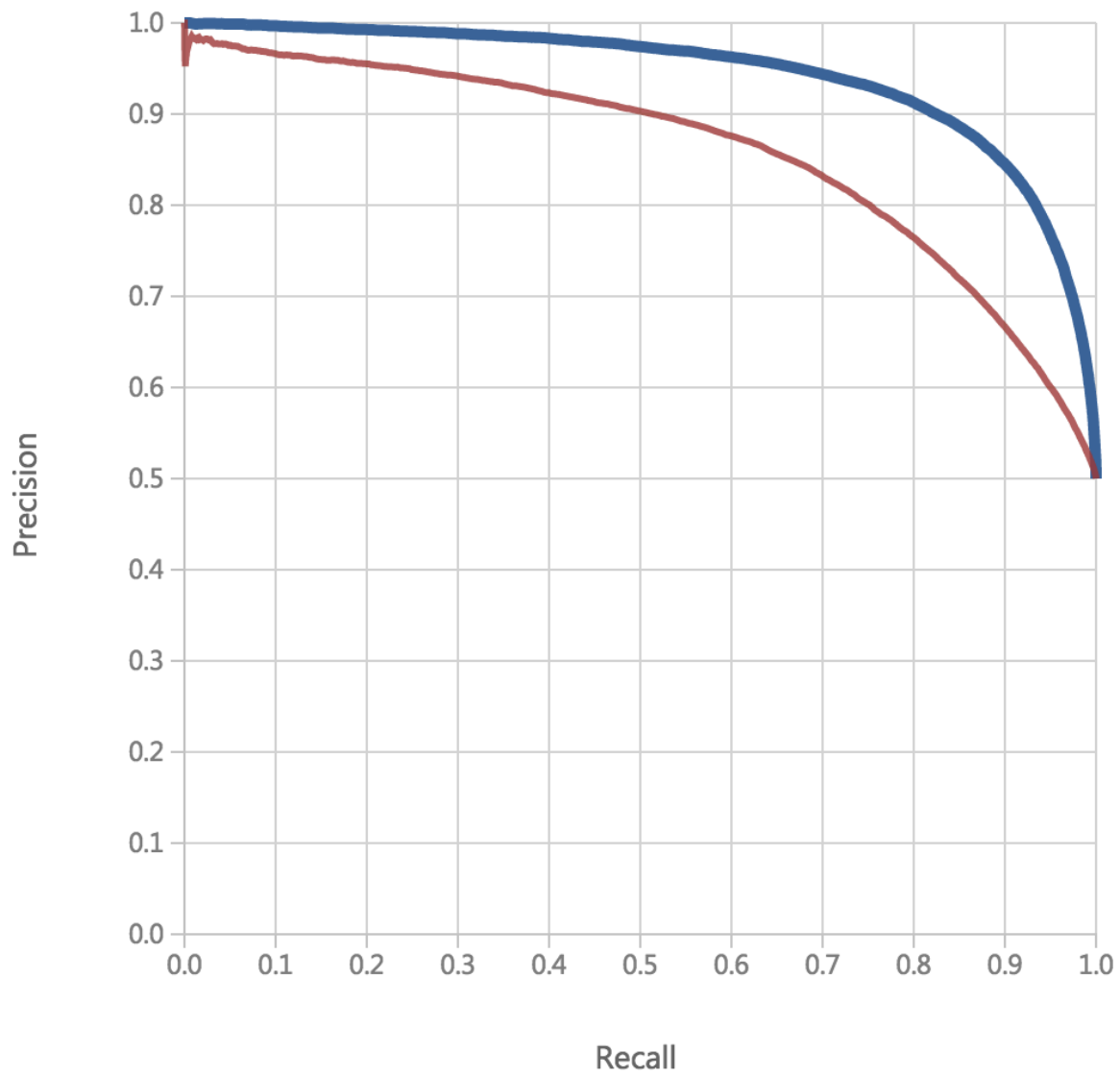


Below is the visualized evaluation when we compared these two models.

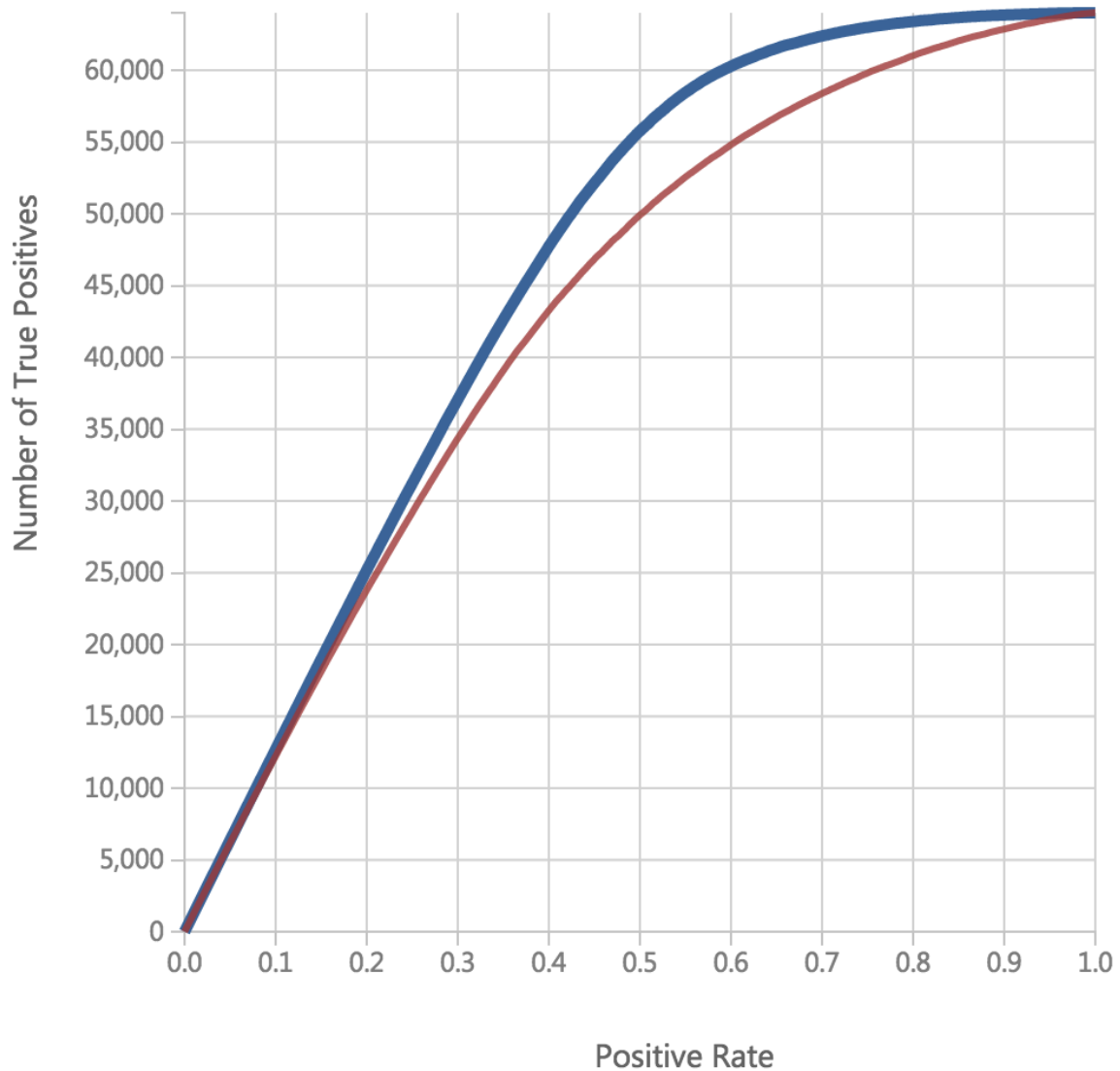
ROC:



Precision-recall:



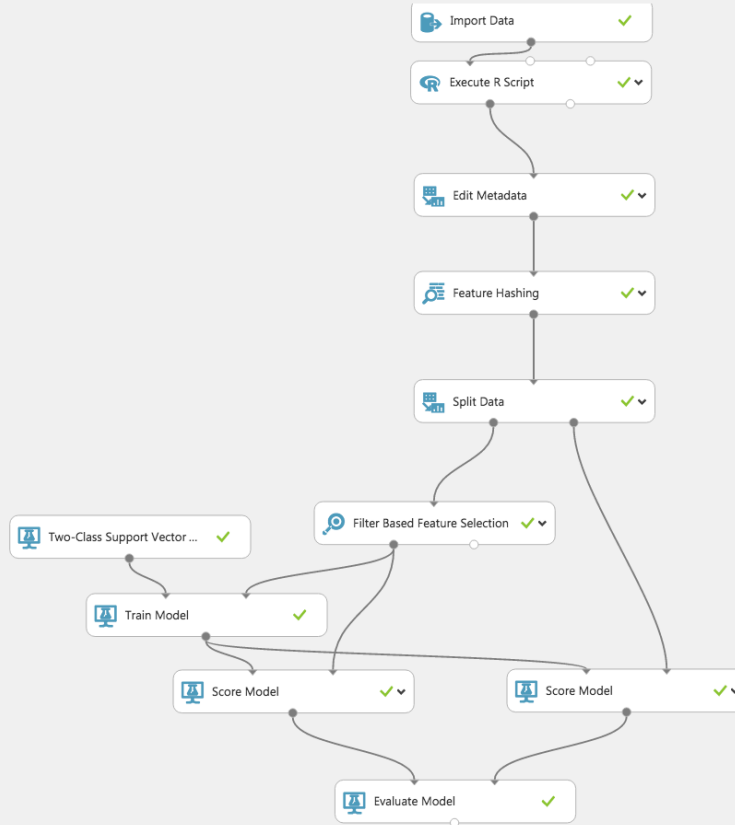
Lift:



(2) In the experiment for deploying web service, we used the two-class vector machine model.

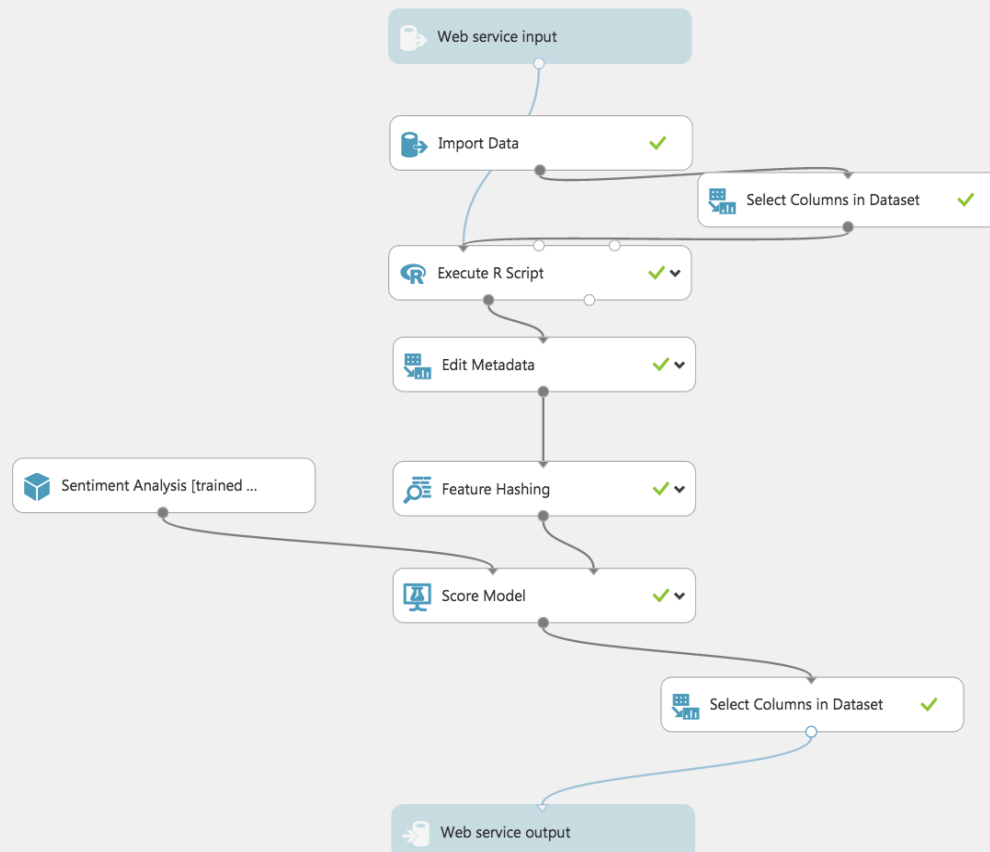
Sentiment Analysis

Finished running ✓



Sentiment Analysis [Predictive Exp.]

Finished running ✓



(3) Web Service/API Testing

(a) Use web service in Azure Studio

×

Test Sentiment Analysis [Predictive Exp.] Service

Enter data to predict

TWEET_TEXT

Assignment 3 is very difficult.

✓

← 'Sentiment Analysis [Predictive Exp.]' test returned [{"4","0.928538978099823"}]...

✓ Result: {"Results":{"output1":{"type":"table","value":{"ColumnNames":["Scored Labels","Scored Probabilities"],"ColumnTypes":["String","Double"],"Values":{"4","0.928538978099823"}}}}}

(b) Test API in Azure Portal

DELETE BES Delete

POST BES Start

GET BES Status

POST BES Submit

POST RRS Execute

Sentiment Analysis API 1

BES Delete

Try it

Request URL

<https://sentimentanalysis.azure-api.net/sentiment-analysis1/workspaces/{workspace}/services/{service}/jobs/{jobid}?api-version:>

4. Azure portal - Stream Analytics

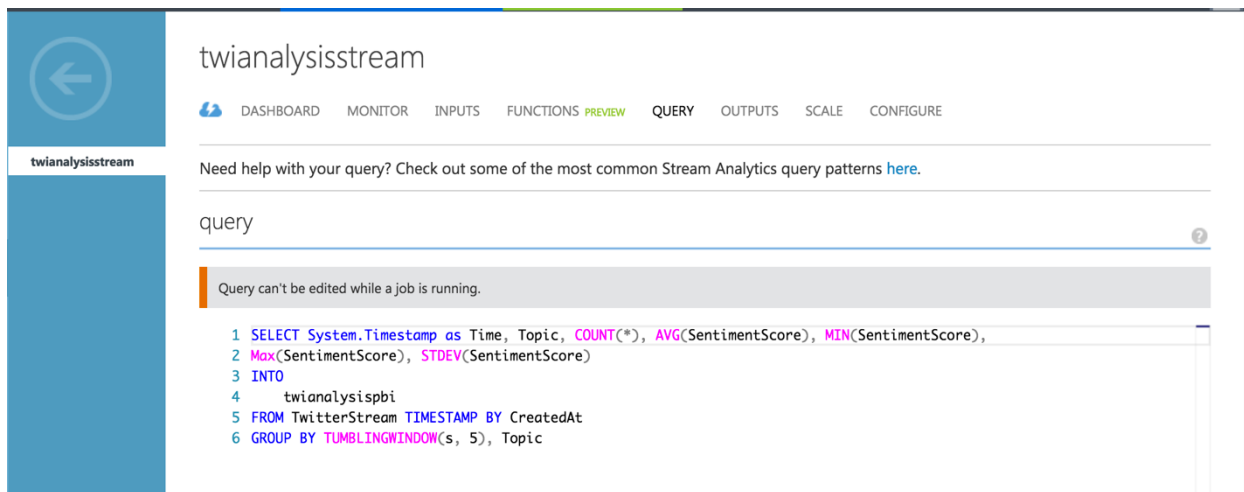
We also built a sentiment analysis solution for social media analytics by bringing real-time Twitter events into Event Hubs. We used Stream Analytics query to analyze

the tweets data and then used a dashboard to provide insights in real time using Power BI.

First, we created an Event Hub input and a Consumer Group in the Azure Portal.

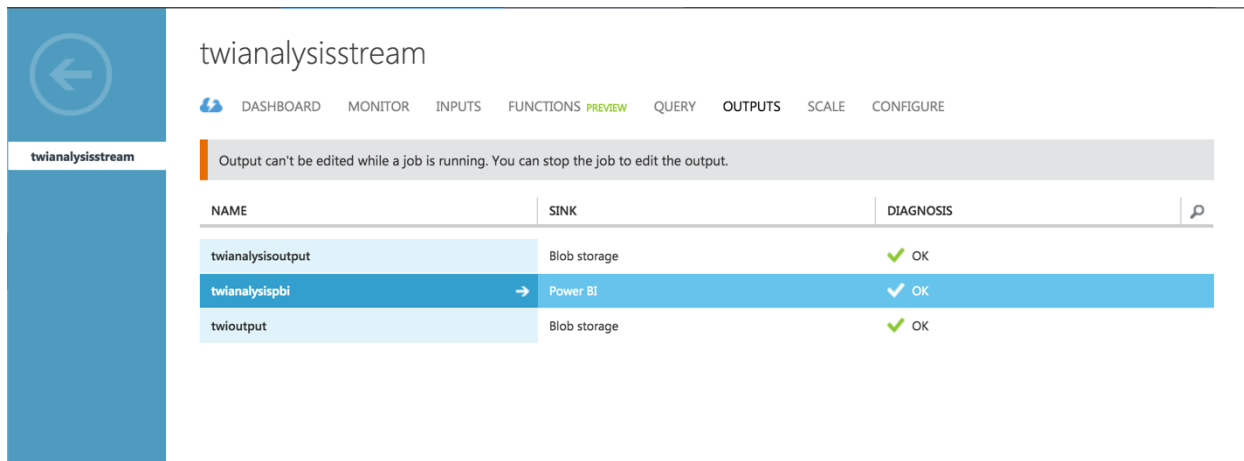
Then, we run a client application that taps into Twitter data via Twitter's Streaming APIs to collect Tweet events about a parameterized set of topics. The 3rd party open source tool Sentiment140 is used to assign a sentiment value to each tweet (0: negative, 2: neutral, 4: positive) and then Tweet events are pushed to Event Hub. To use this client, we generated our Twitter account and OAuth access token.

At last, we set up a Stream Analytics job to analyze these events in real time.

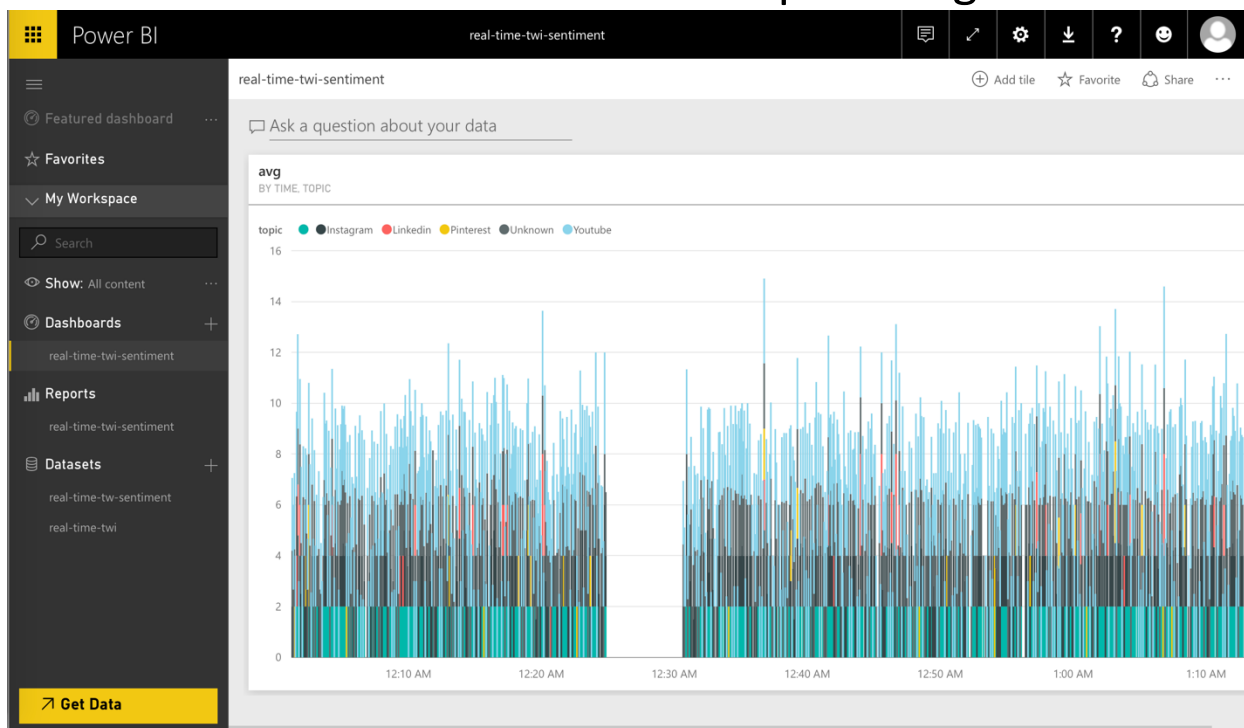


The screenshot shows the Azure Stream Analytics portal interface for a job named 'twianalysisstream'. The left sidebar contains a navigation menu with a back arrow icon and the job name. The main area has a top navigation bar with tabs: DASHBOARD, MONITOR, INPUTS, FUNCTIONS (highlighted in green), QUERY, OUTPUTS, SCALE, and CONFIGURE. Below the tabs, there is a link to 'Need help with your query? Check out some of the most common Stream Analytics query patterns [here](#).' The 'query' tab is selected, showing a text area with a SQL query. A message states 'Query can't be edited while a job is running.' The query is as follows:

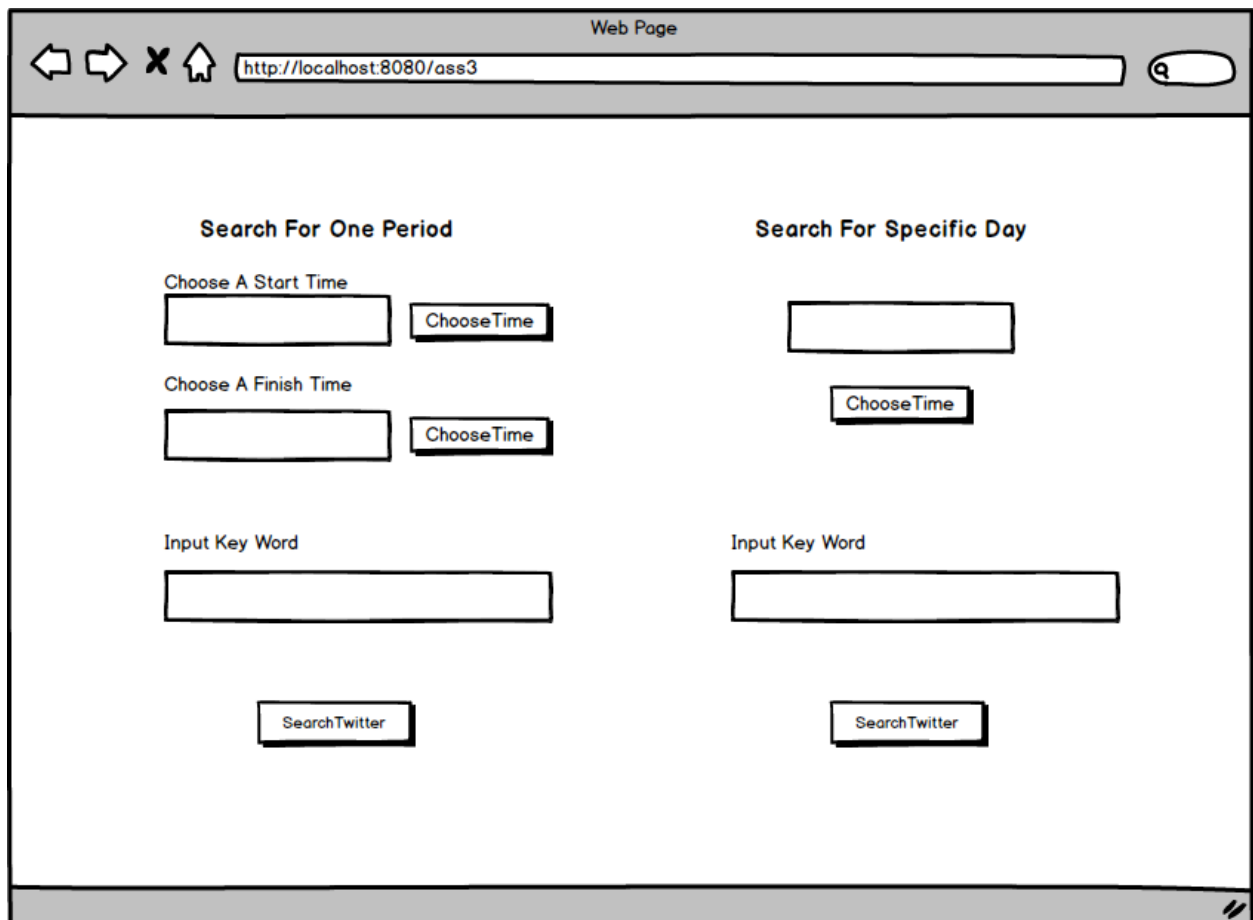
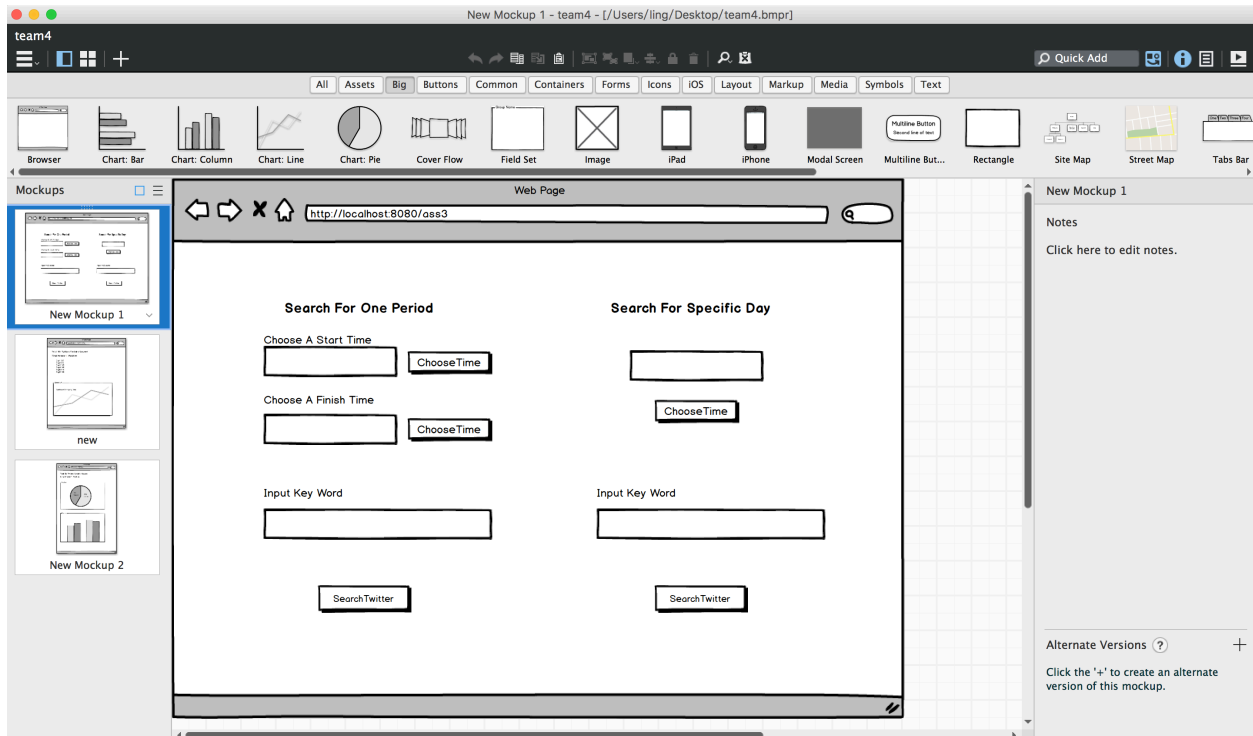
```
1 SELECT System.Timestamp as Time, Topic, COUNT(*), AVG(SentimentScore), MIN(SentimentScore),
2 Max(SentimentScore), STDEV(SentimentScore)
3 INTO
4     twianalysisbpi
5 FROM TwitterStream TIMESTAMP BY CreatedAt
6 GROUP BY TUMBLINGWINDOW(s, 5), Topic
```

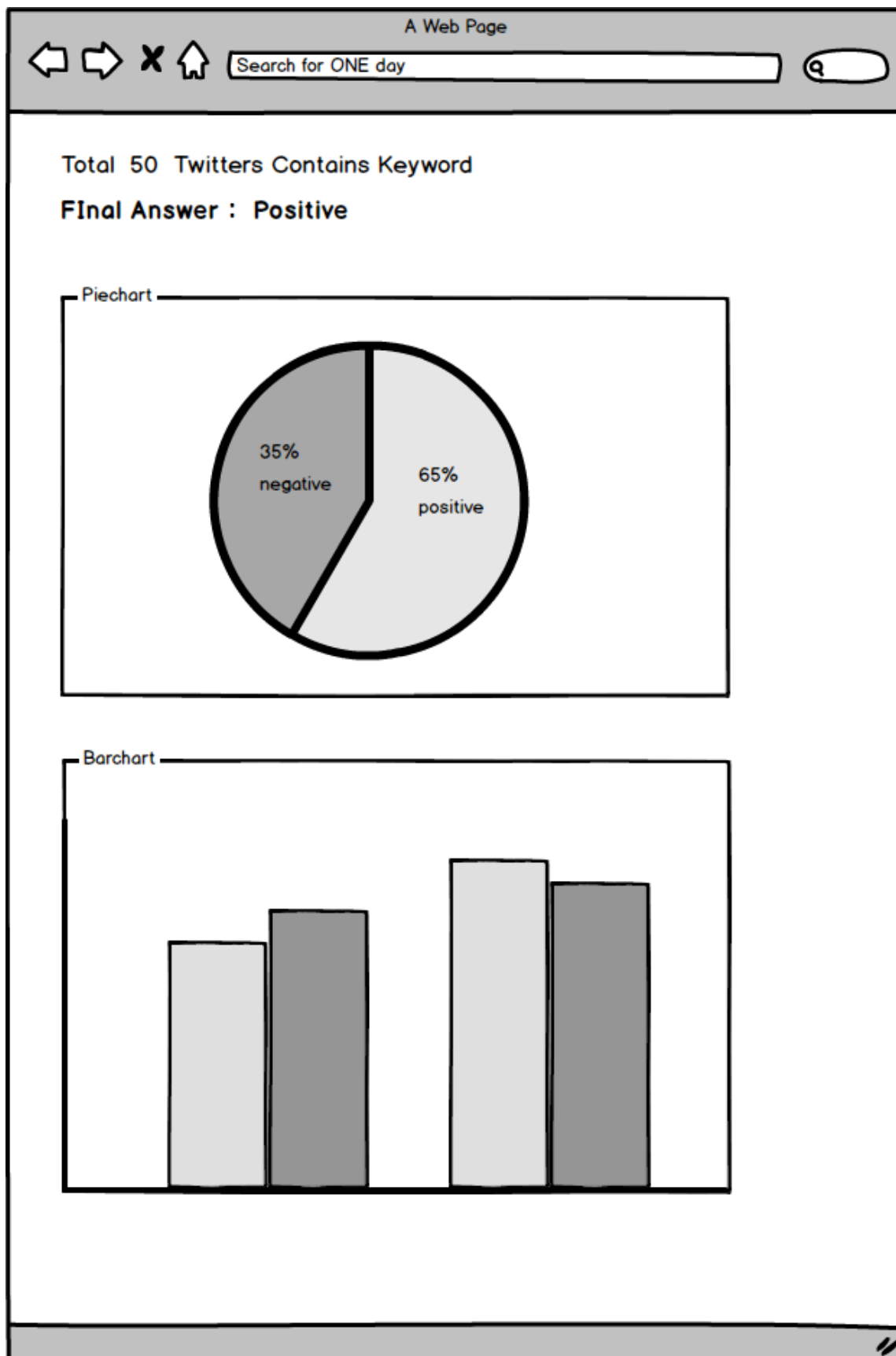


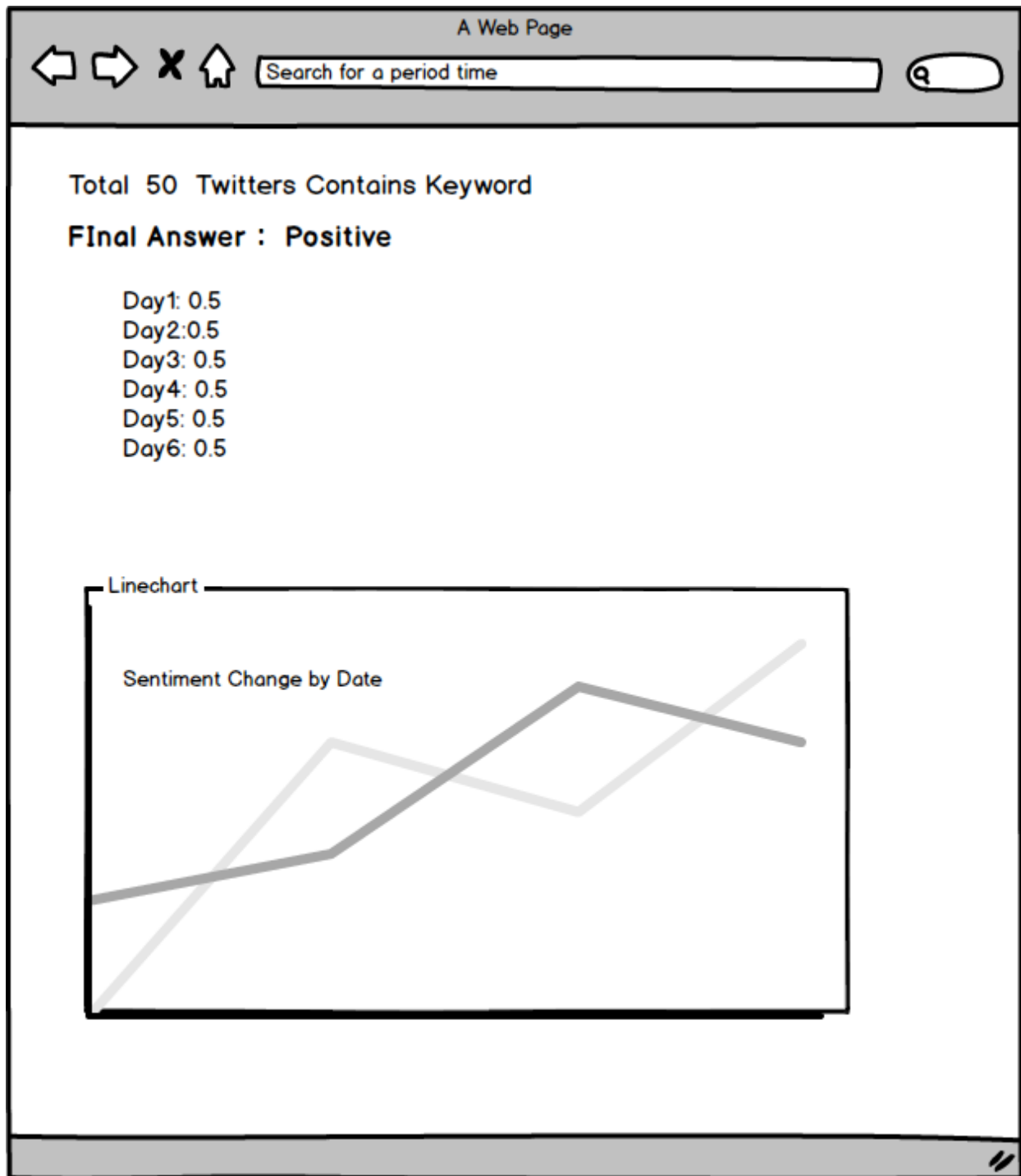
Once the job is running and processing the real-time Twitter stream, we can view the output for sentiment analysis and extend our application to include a customized dashboard over the output using Power BI.



5. Balsamiq Mockup


































6. Web application

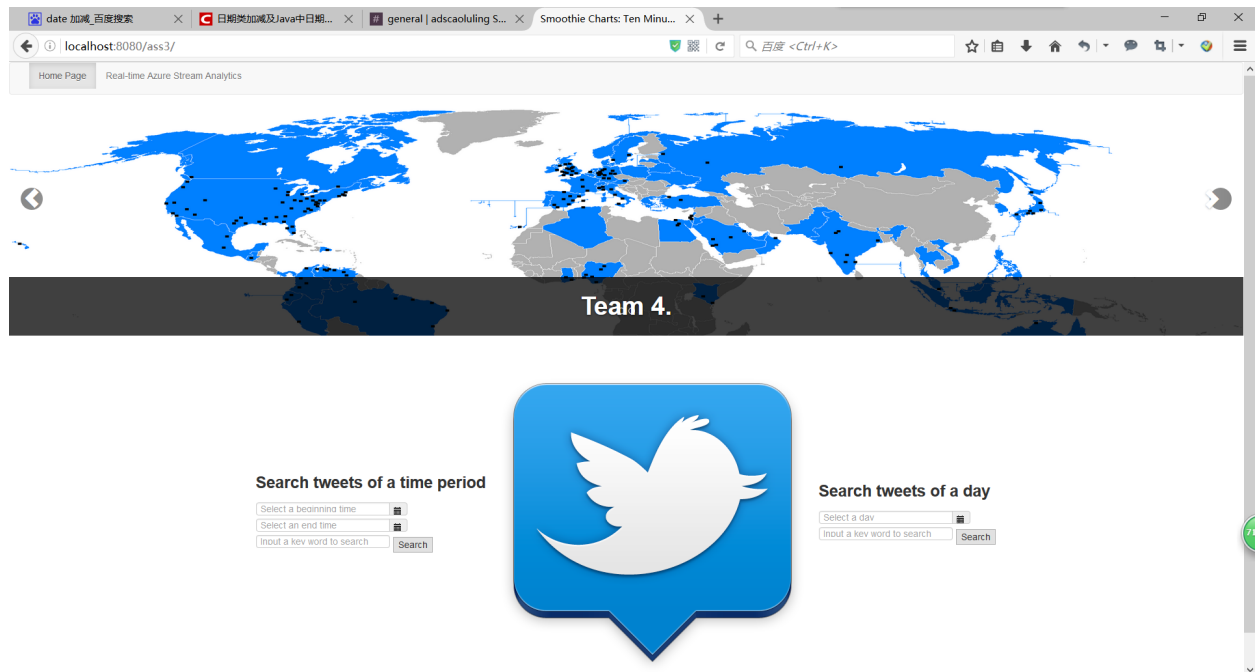
We generated our web application based on Spring MVC and bootstrap, there are two main function in our website, the first one is analyzing tweets according to

keyword in a period of time and generating graph, the second is analyzing one day's tweets by sentimental algorithm and generating graph in details.

To make our web application more friendly, it has three jsp page with bootstrap and javascript, and three Controller to take charge of first function, API and second one respectively.

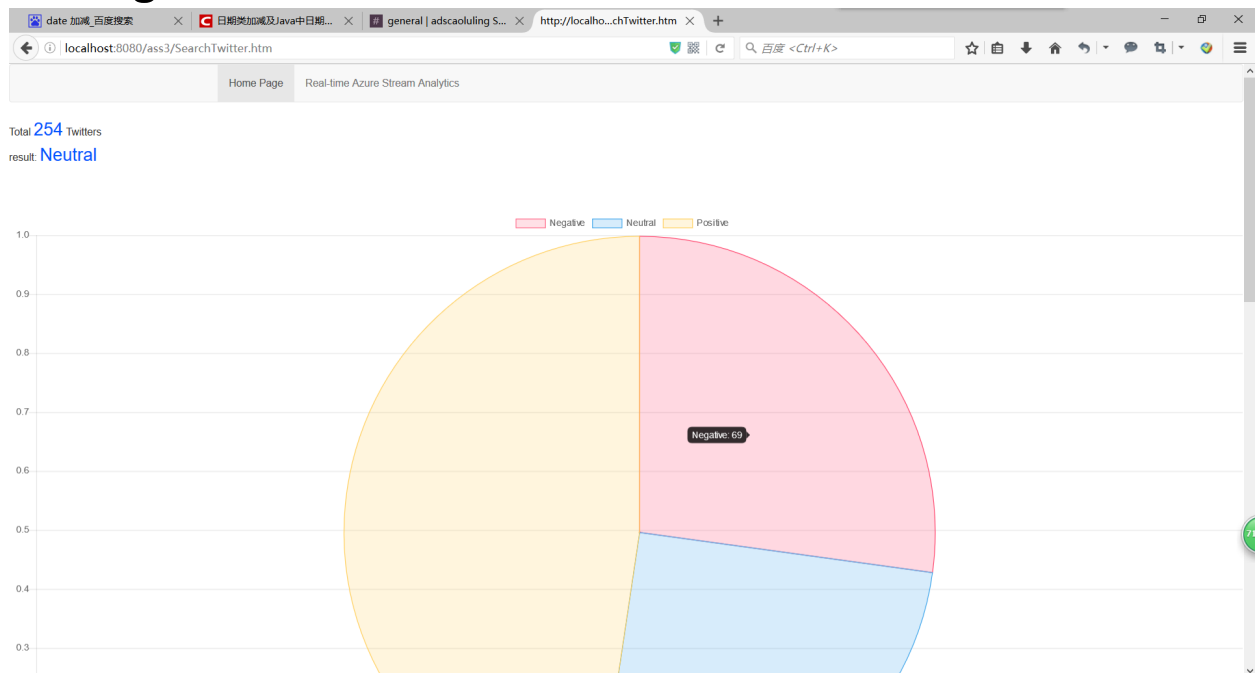
- ▼  adsass3
 - ▼  src/main/java
 - ▼  com.ads.ass3
 - >  HomeController.java
 - >  HttpPostReq.java
 - >  twitterController.java
 - ▼  src/main/resources
 -  META-INF
 -  log4j.xml
 -  twitter4j.properties
 - >  src/test/java
 - >  src/test/resources
 - >  JRE System Library [JavaSE-1.6]
 - >  Maven Dependencies
 - >  Referenced Libraries
- ▼  src
 - ▼  main
 - ▼  webapp
 - >  resources
 - ▼  WEB-INF
 -  classes
 - >  spring
 - ▼  views
 -  home.jsp
 -  res.jsp
 -  result.jsp
 -  web.xml

This is our home page

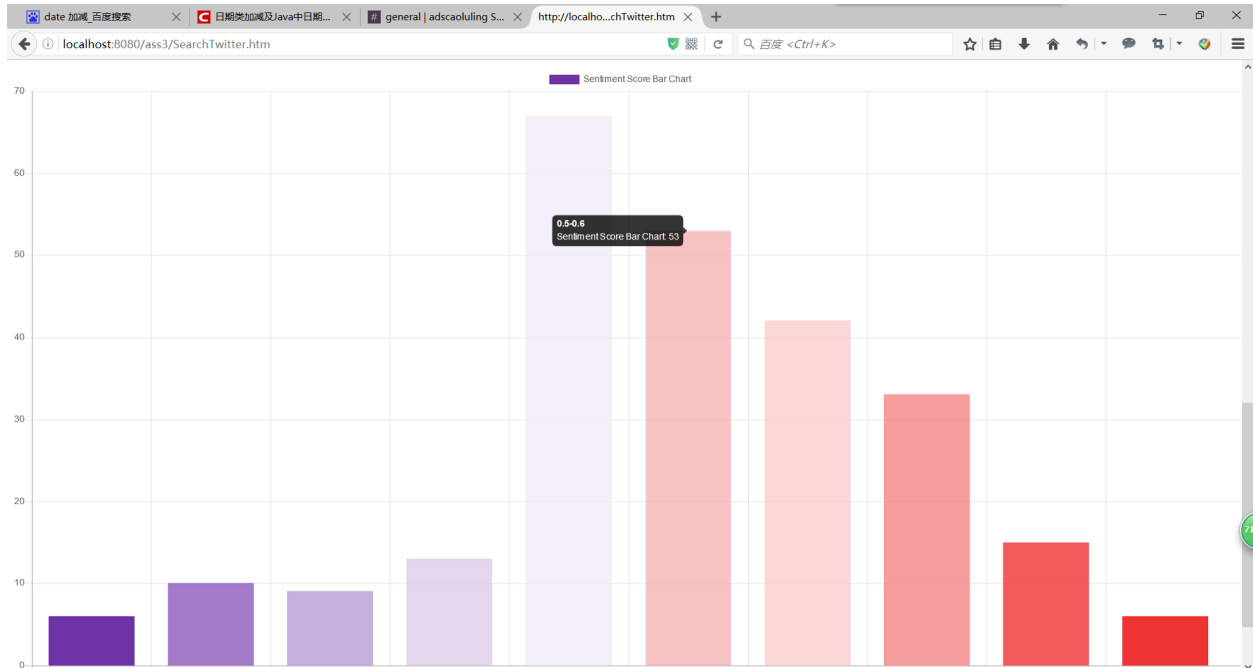


One the left can search tweets based on time period, one the right can search tweets in details

The right hand function:

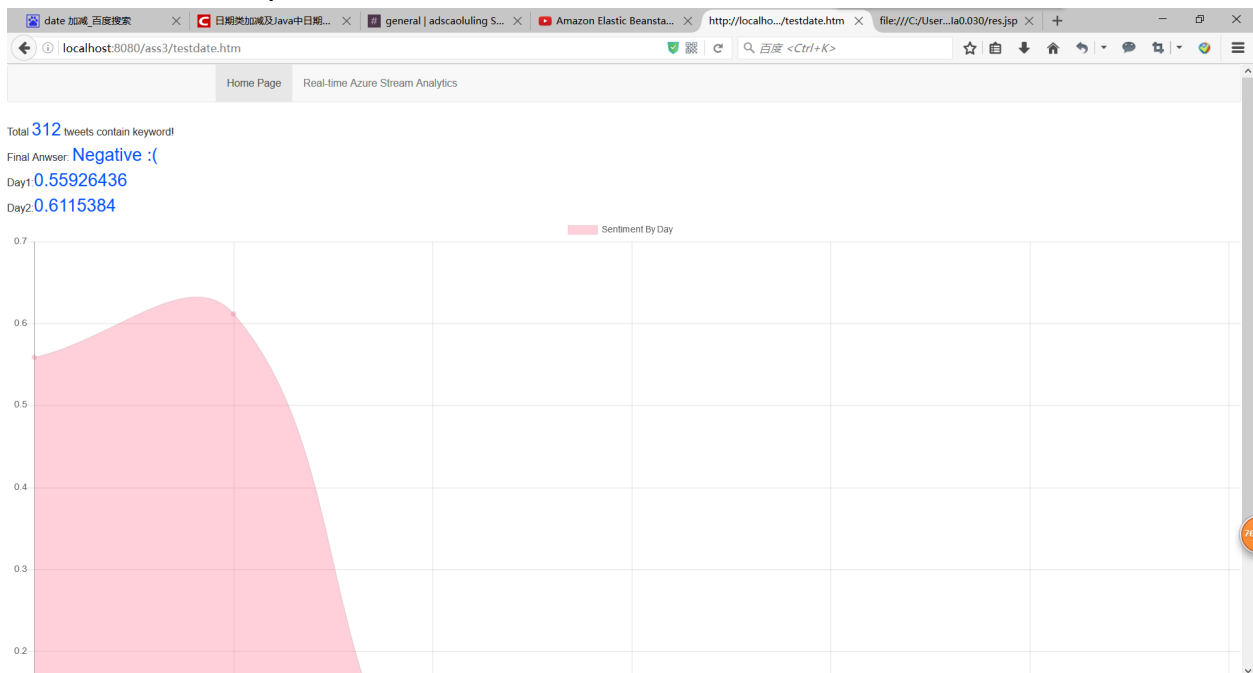


This graph shows the percentage of people's feeling.

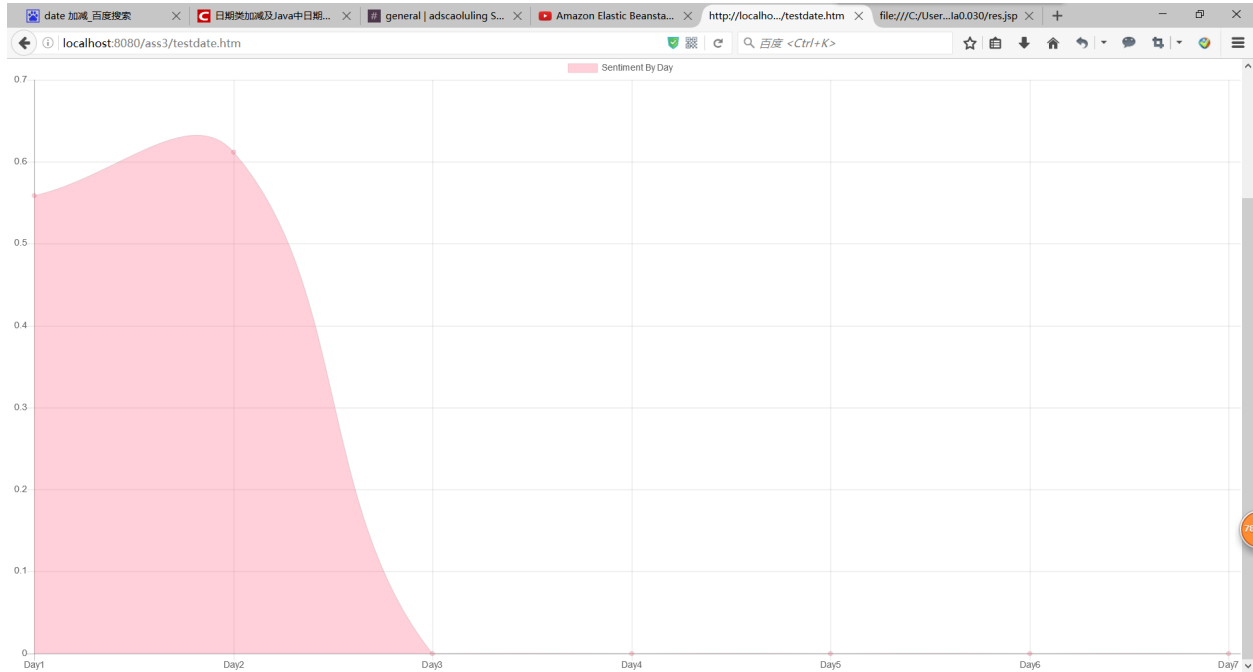


This graph shows the distribution of people's feeling

This is the left hand function page, the information of total tweets, final answer and details of date are shown.



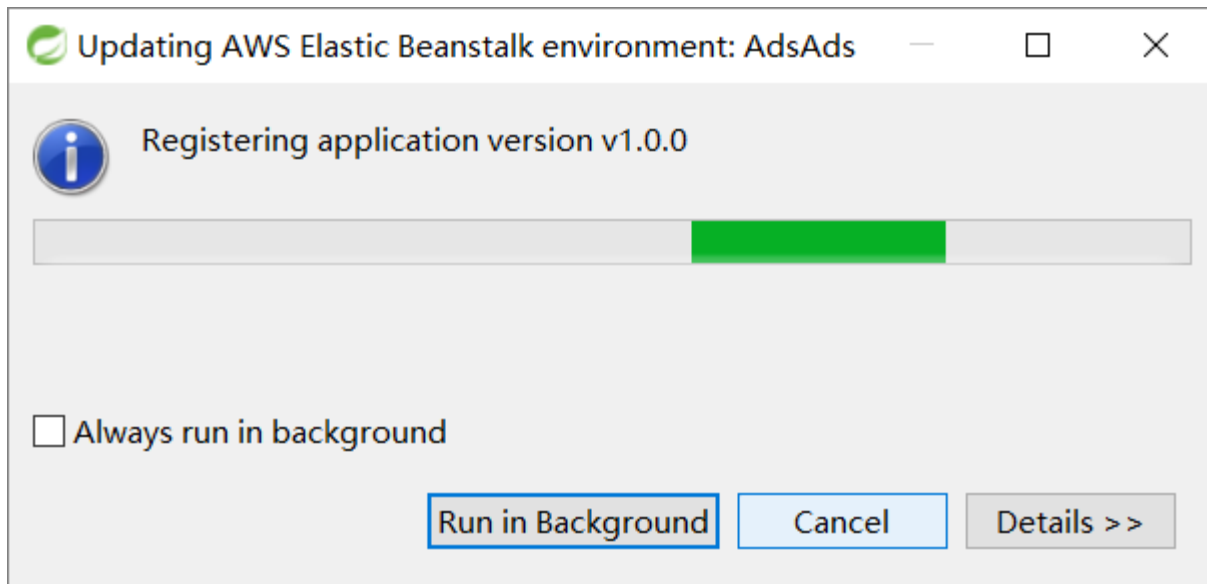
This is the whole graph of trend, x-axis is date, y-axis is score, 0 to 0.45 is negative, 0.46 to 0.55 is neutral, 0.56 and higher is positive.



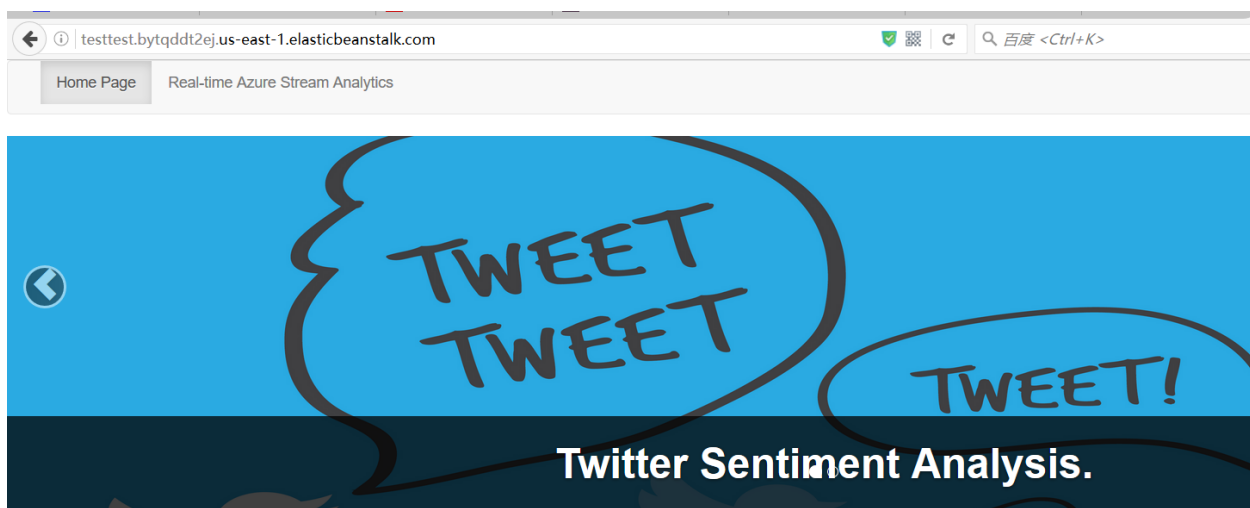
To allow our web application to access by anywhere, we decided to use AWS ToolKit for Eclipse to put our app into AWS Elastic Beanstalk.



Uploading:



After a few minutes, we can access our web by url:
<http://testtest.bytdddt2ej.us-east-1.elasticbeanstalk.com/>



Search tweets of a time period

Select a beginning time



Search tv