# Project Report

# Stock Recommender System

**CS 410: Text Information System**

**University of Illinois, Urbana-Champaign**

## Team Members

|  |  |  |
| --- | --- | --- |
| **Name** | **NetID** | **Email ID** |
| Ezra Schroeder | [ezras2](mailto:ezras2@illinois.edu) | [ezras2@illinois.edu](mailto:ezras2@illinois.edu) |
| Rasbihari Pal | [pal9](mailto:pal9@illinois.edu) | [pal9@illinois.edu](mailto:pal9@illinois.edu) |
| ****Sandeep Kumar**** | [kumar64](mailto:kumar64@illinois.edu) | [kumar64@illinois.edu](mailto:kumar64@illinois.edu) |

Team captain marked in **BOLD.**

[ [Team Members](#_Team_Members) ] [ [Project Overview](#_Project_Overview) ] [ [Abstract](#_Abstract) ] [ [Motivation](#_Motivation) ] [ [Introduction](#_Introduction) ] [ [High Level Design](#_High_Level_Design) ] [ [Component View (Deployment Perspective)](#_Component_View_(Deployment) ] [ [SRE Front-end (UI)](#_Component_View_(Deployment) ] [ [SRE Back-end (Rest API)](#_SRE_Back-end_(Rest) ] [ [Implementation details](#_Implementation_details) ] [ [Rating System](#_Rating_System) ] [ [Sentiment Analysis](#_Sentiment_Analysis) ] [ [Recommender System](#_Recommender_System) ] [ [Libraries and Datasets](#_Libraries_and_Datasets) ] [ [Verification](#_Verification) ] [ [Conclusion and Future Work](#_Conclusion_and_Future) ] [ [References](#_References) ]

## Project Overview

## Abstract

Our project is a novel text-mining application which combines 3 different functionalities around rating stocks into one application. Namely, it consists of a base analysis which is a two-tiered summary of sentiment of stock analyst ratings who are well known in the industry and rate stock ticker symbols, a twitter sentiment analysis aspect which scrapes tweets off of Twitter and analyzes them for sentiment about a particular stock symbol, and a recommendation engine which recommends stock symbols similar to a user-provided query stock symbol. There are many conceivable use-cases for an app such as ours upon further embellishment and improvements, such as individual investors in the stock market who want automated and instantaneous advice and suggestions that incorporates both analyst ratings from well-known analysts across the internet, public sentiment as embodied by tweets, and that produces not only concise summaries of these analyst ratings and Twitter sentiment but recommends similar stocks to their stock symbol (e.g. AAPL) of interest. Imaginably there may well also be corporate interest in incorporating an application such as ours into a larger pipeline which could be serving a huge myriad of purposes but that leverages actionable knowledge about the stock market into a larger purpose. Although our application is alpha version, it is not inconceivable that it could springboard academic research into these areas.

## Motivation

In the year 2020, there has been huge surge in securities trading, driven by retail investors. The increase in the trading activity can be attributed primarily to the easy-to-use mobile based trading apps, offered by several FinTech companies such as eToro, Robinhood and InteractiveBrokers etc. The retail investors of today might lack time for in-depth research and/or even lack the necessary knowledge to analyze the financial standing of a company. In such a situation, many of the retail investors either rely on word of mouth or blindly following other investors on such trading platforms. This leads to investment decisions beyond the risk profile and/or risk appetite of the investors.

## Introduction

We propose “Stock recommender system” as a solution to enable retail investors easy access to information, relevant for informed investment decisions. Based on user’s preference of a stock, the stock recommender proposes a cumulated rating for the stock and also proposes other stocks with similar ratings. The recommender system combines

* stock rating data from various market analyst,
* market sentiments and
* company profile

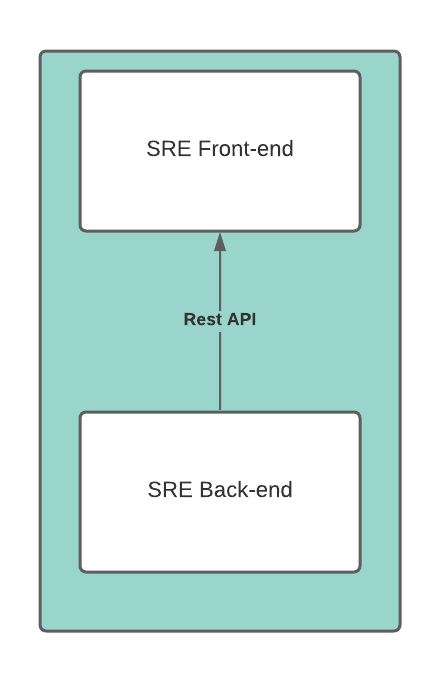
for determination of the cumulated rating and curation of the recommendation list.

## High Level Design

### Component View (Deployment Perspective)

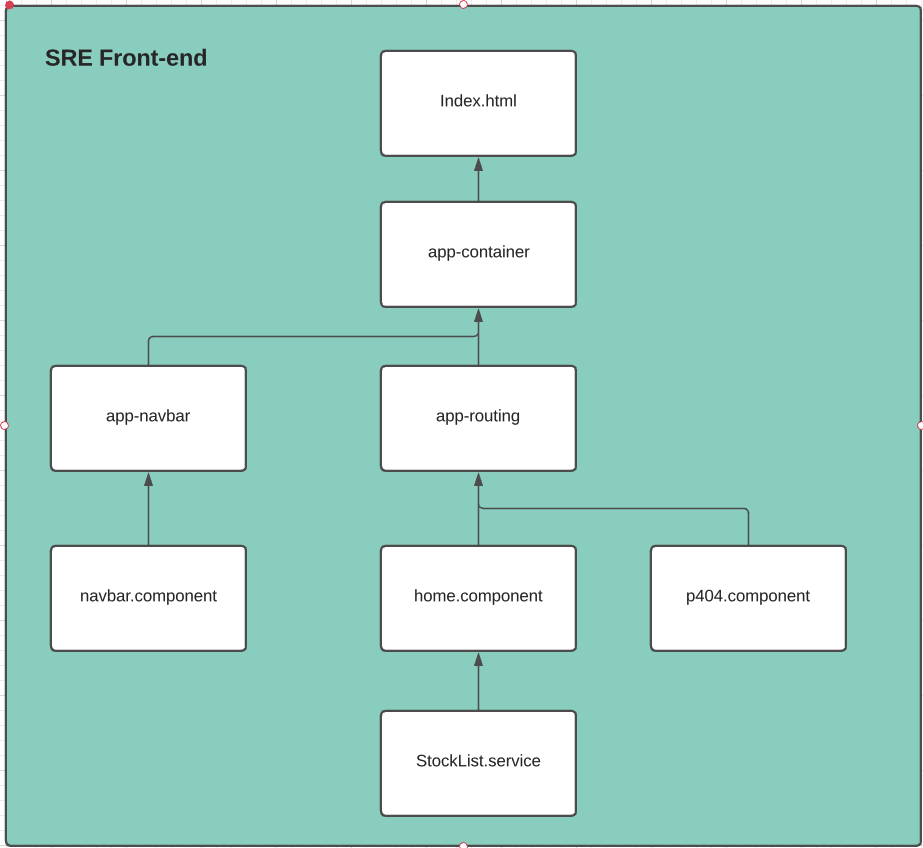
Stock Recommendation System consists of two components - SRE Front-end and SRE Back-end.

* The SRE Front-end is the UI component for user interaction, developed using Angular. The Front-end relies on SRE Back-end for all its data needs for providing various user centric functions.
* The SRE Back-end is the main component, which implements all the necessary algorithm and business rules and finally exposes the data related to ratings, recommendations and tweets via Rest APIs.



### SRE Front-end (UI)

The SRE Front-end is developed in Angular framework and is a single page application. It consists of the following components:



**A.    app-container**

The main component to render the Stock Recommendation System web page

**B.    app-navbar**

The component responsible for top navigation bar

**C.    app-routing**

The component dealing with url based routing and takes care of routing to the home page or canonical “page-not-found“ page.

**D.    home.component**

This is the main component which takes user input, calls SRE backend services to fetch relevant data.

**E.    p404.component**

This is the component for handling “page not found” scenario in case user enters incorrect URL not supported by SRE Front-end.

### SRE Back-end (Rest API)

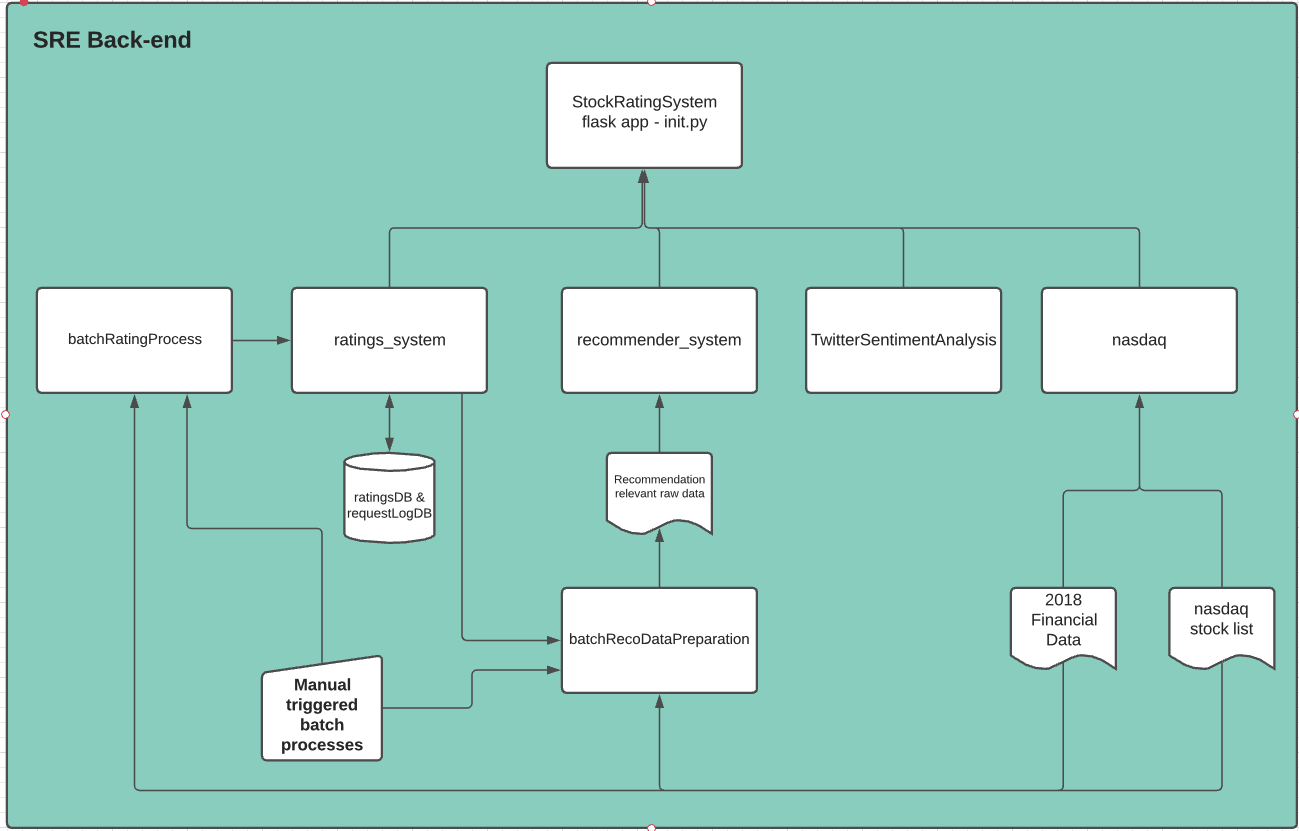
The SRE Back-end is a Flask based app, developed using Python and TinyDB which exposes various Rest APIs for the SRE Front-end.

List of Rest APIs:

|  |  |
| --- | --- |
| **Rest API** | **Sample Response** |
| Get list of all stocks in the corpus (output abridged for succinctness)  GET /stock/all | [  {  "Symbol":"AAPL",  "Security Name": "Apple Inc",  "Market": "NASDAQ",  "Sector": "Technology"  }  ] |
| Get ratings for a given stock from a given market  GET /stock/ratings/<market>/<stock\_symbol> | [  {  "stockSymbol": "AAPL",  "marketPlace": "NASDAQ",  "refreshData": "2020-11-25",  "overallRating": "HOLD",  "analystsRatings": [  {  "level\_0": 0,  "index": 1,  "ratingDate": "2020-11-19",  "ratingAgency": "The Goldman Sachs Group",  "ratingAssigned": "Sell",  "newRatings": -1,  "scaledRatings": "SELL"  }  ]  }  ] |
| Get a list of recommended stocks, similar to given stock  GET /stock/recommendation/<stock\_symbol> | [  {  "seq": 1,  "stockSymbol": "HAFC",  "stockName": "Hanmi Financial Corporation",  "sector": "Finance",  "rating": "SELL"  },  ] |
| Get Twitter sentiment for given stock  1: Positive,  0: Neutral, -1: Negative  GET /stock/sentiments/<market>/<stock\_symbol> | {  "stockSymbol": "AAPL",  "refreshDate": "2020-11-25",  "sentiment": 1  } |
| Get list of user request log  GET /requests/all | [  {  "datetime": "2020-11-25T21:55:52.924706",  "symbol": "TSLA",  "market": "NASDAQ"  }  ] |

List of the Rest APIs exposed out of the backend can be found on GitHub project repository: <https://github.com/MLwithSandy/CourseProject/tree/main/ProjectCode/StockRatingsSystem>

The SRE Back-end is comprised of the following components and/or modules:



**A.    Flask app**

This is the main component which exposes various Rest APIs for external consumers - in our case SRE Front-end. It integrates all other components such as ratings\_system, recommender\_system etc. to provide the necessary services. In addition, it also logs all requests to requestLogDB.

**B.    ratings\_system**

The ratings\_system is the component which calculates the overall rating for the user selected (searched) stock, by scraping the required ratings data from a variety of market analyst websites and aggregating them. It also provides the list of our selected latest ten ratings from an assortment of various market analysts as a reference.

**C.    recommender\_system**

The recommender\_system takes a stock symbol as input and recommends 5 stocks matching the profile of the user selected (searched) stock and a pre-defined similarity function.

**D.    TwitterSentimentAnalysis**

TwitterSentimentAnalysis component fetches latest tweets for the user selected (searched) stock from Twitter which by making use of the Twitter API, performs sentiment analysis and returns overall sentiment for the given stock. It also provides the latest five tweets as a reference.

**E.    nasdaq**

The nasdaq component is primarily used for preparation of data. It combines the nasdaq listed stocks with the dataset we used for identifying various feature parameters of a stock. By cleaning and combining our data sources, we were able to create a final dataset of 1462 stocks for use across our systems in our app (analyst ratings, twitter sentiment, recommender system).

**F.    batchRatingProcess**

The batchRatingProcess is used to update ratings of all 1462 stocks in background. This is to ensure that the user does not face any significant delay in fetching the data from third party websites while interacting with our SRE app’s UI. In addition, the ratings of a stock is not updated frequently and batch processing fits well from a requirements and solution design perspective.

**G.    batchRecoDataPreparation**

The batchRecoDataPreparation is another batch process to prepare the data for the recommendation system as the recommendation engine relies on an assortment of analyst ratings of stocks as a feature in the similarity function.

## Implementation details

### Rating System

**A.    Approach**

To calculate the ratings, the following algorithm is used:

* First of all, rating for the selected(searched) stock is searched in the ratings database.
  + Case 1: ratings data is already available in the ratings database. In such a case, ratings data is fetched from the database.
  + Case 2: ratings data is not available in ratings database. In this case, following step is executed to get the ratings.
    - Read the website marketbeat.com which lists the ratings from various market analysts and scrape the page section containing ratings relevant date
    - Data is cleansed, validated and structured as per requirement
    - In case of more than 10 ratings, only latest 10 ratings are selected.
    - Initially data from only current and previous months were considered but to increase the data quantity, this restriction is switched off in final product
    - Final list of ratings is stored in the database
* Once the rating data is available, ratings from various market analysts are scaled to following ratings scale: {-1: SELL, 0: HOLD, 1: BUY}
* Finally, overall rating is calculated based on the selected scaled ratings with possible outcome as SELL, HOLD or BUY.

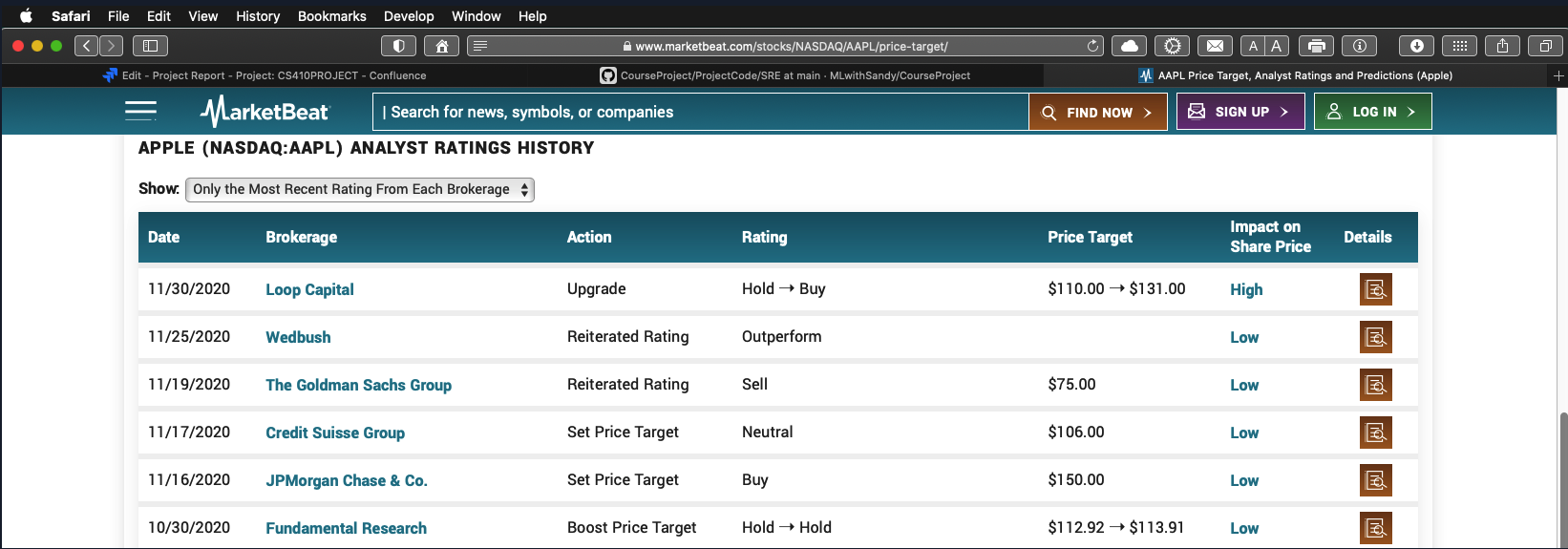
**B.    Webpage Scraper**

Webpage scraper was developed using the learnings from MP 2.1 of the course CS 410: Text Information System. Python library BeautifulSoup and Chrome driver were primarily used to fetch the web document.

Following URL builder code were used to get URL for various stocks:

|  |
| --- |
| # MarketBeat URL builder  def get\_mb\_url(market, stock\_symbol):  base\_url =  'https://www.marketbeat.com/stocks/{market}/{stock\_symbol}/price-target/'  final\_url = base\_url.format(market=market, stock\_symbol=stock\_symbol)  return final\_url |

e.g. <https://www.marketbeat.com/stocks/NASDAQ/AAPL/price-target/>



As part of the exercise, following additional aspects were required to be taken care: 1. the time delay in reading the complete web document, and 2. the URL redirects - in case a particular stock was not found on the website, website would redirect the URL request to a default page.

|  |
| --- |
| # create a webdriver object and set options for headless browsing  def load\_webdriver():  if str(filePath).find("CS410") == -1:  options = webdriver.ChromeOptions()  options.add\_argument('--no-sandbox')  options.add\_argument('--headless')  options.add\_argument("--disable-dev-shm-usage")  driver = webdriver.Chrome(chrome\_options=options)  else:  options = Options()  options.headless = True  driver = webdriver.Chrome(filePath / 'chromedriver', options=options)  return driver |

|  |
| --- |
| # read web document using beutifulsoup  def get\_js\_soup(url\_web, driver):  driver.get(url\_web)  time.sleep(5)  if url\_web != redirected\_url:  print("redirected URL : " + redirected\_url)  new\_url = redirected\_url + "price-target/"  print("new URL for Analysts rating : " + new\_url)  if new\_url.find('NASDAQ/price-target/') == -1:  driver.get(url\_web)  else:  return |

Using the html div class identifier, relevant data were extracted from the webpage data and subsequently processed in panda dataframe.

table = soup\_obj.find("table",

attrs={"class": "scroll-table sort-table fixed-left-column fixed-header"})

if table is None:

table = soup\_obj.find("table",

attrs={"class": "scroll-table sort-table fixed-header"})

if table is None:

table = soup\_obj.find("table",

attrs={"class": "scroll-table sort-table"})

if table is None:

print(stock\_symbol + ": No table found for market analyst rating")

else:

table\_body = table.find('tbody')

rows = table\_body.find\_all('tr')

for row in rows:

cols = row.find\_all('td')

cols = [ele.text.strip() for ele in cols]

data.append([ele.split("➝")[-1].strip() for ele in cols if ele])

**C.    Uniform Scaling of Ratings**

Uniform scaling of ratings was required to unify the ratings from various market analysts and ensure that the calculation of overall rating is without any bias and is not affected by some higher or lower ratings from some market analysts, only due to the fact that they use different scales for rating a stock.

Various ratings were mapped to {-1: SELL, 0: HOLD, 1: BUY} using following mapping dictionary is used.

ratings\_dict = {

"SELL": -1, "STRONG SELL": -1, "BEARISH": -1, "UNDERPERFORM": -1,

"SECTOR UNDERPERFORM": -1, "MODERATE SELL": -1, "WEAK HOLD": -1,

"UNDERWEIGHT": -1, "REDUCE": -1,

"HOLD": 0, "NEUTRAL": 0, "AVERAGE": 0,

"MARKET PERFORM": 0, "SECTOR PERFORM": 0, "SECTOR WEIGHT": 0,

"PEER PERFORM": 0, "EQUAL WEIGHT": 0, "IN-LINE": 0,

"MARKET OUTPERFORM": 1,

"OUTPERFORM": 1, "MODERATE BUY": 1, "ACCUMULATE": 1, "OVER-WEIGHT": 1,

"OVERWEIGHT": 1, "STRONG-BUY": 1, "ADD": 1, "BULLISH": 1, "BUY": 1,

"POSITIVE": 1, "STRONG BUY": 1, "TOP PICK": 1, "CONVICTION-BUY": 1,

"OUTPERFORMER": 1

}

There were other options also considered e.g., scaling of all ratings on a scale of 1-5 or 1-3, weighted scale to reflect strong ratings e.g., strong buy or strong sell. However, based on our need to combine the analyst rating with twitter sentiment, scaling to {-1: SELL, 0: HOLD, 1: BUY} were selected for this exercise.

**D.    Overall Ratings Calculation**

First of all, aggregated ratings of all ratings from various market analyst is calculated based on arithmetic mean of all selected scaled ratings. e.g., aggregated rating for following five scaled ratings {-1: SELL, 1: BUY, 0: HOLD, 1: BUY, -1: SELL} from various market analyst will be {0: HOLD}.

Once the aggregated rating is calculated, it is combined with Twitter Sentiment analysis result as per below table for Overall ratings.

|  |  |  |
| --- | --- | --- |
| **Aggregated Rating** | **Twitter Sentiment** | **Overall Ratings** |
| -1: SELL | POSITIVE | HOLD |
| -1: SELL | NEUTRAL | SELL |
| -1: SELL | NEGATIVE | SELL |
| 0: HOLD | POSITIVE | BUY |
| 0: HOLD | NEUTRAL | HOLD |
| 0: HOLD | NEGATIVE | SELL |
| 1: BUY | POSITIVE | BUY |
| 1: BUY | NEUTRAL | BUY |
| 1: BUY | NEGATIVE | HOLD |

### Sentiment Analysis

Sentiment Analysis, also known as Opinion Mining, refers to the use of Natural Language Processing to computationally determine opinions and emotions of an opinion holder for an opinion target. A common use for this technology is to discover how people feel about certain topics, particularly through users’ textual posts in Social Media space. To perform Sentiment Analysis to provide stock recommendation, Twitter has been considered as the Social Media space where users post their opinion as their tweets. As most of the elements in the opinion representation such as the opinion holder (Twitter users in this case) and opinion target (Stock to be recommended in this case) and the content and the context of the opinion (financial context) are already known, the main task is to decide opinion sentiment. So, this is a case of just using sentiment classification for understanding opinion where the input is opinionated text object, the output is a sentiment label i.e., polarity analysis with predefined categories such as positive, negative, or neutral. For this project, the sentiment analysis has been done through the process outlined below.

**Process Description**

**A.    Preparing the Data Set**

To build the model, training and testing data set is needed. Ideally for optimal performance the financial tweets need to be downloaded from tweeter and should be human evaluated to create the labels which can be used for training the model and later for testing the model. However, one needs to spend considerable amount of manual effort to build such data. To optimize time and resource for this project, a readily available downloadable training set (polarity dataset from Cornell university -) has been used. The data set contain 2000 processed down-cased text files used in Pang/Lee ACL 2004 [[1](#b1)]; the names of the two subdirectories in that folder, "pos" and "neg", indicate the true classification (sentiment) of the component files according to the automatic rating classifier the tweets of the data set have been all labeled as positive or negative, depending on the content.  The data set have been persisted into pickle file (a binary representation of python structure) to optimize performance of the subsequent run to build the classifier.

import pickle

from sklearn.datasets import load\_files

#nltk.download('stopwords')

#Import Dataset -> generate two classes one each for each sub directorties

dataset = load\_files('txt\_sentoken/')

X,y = dataset.data, dataset.target

#store as pickle file, these are byte type file

with open('X.pickle', 'wb') as f:

pickle.dump(X,f)

with open('y.pickle', 'wb') as f:

pickle.dump(y,f)

The same data has been split into training and testing data set following a 80-20 rule where 80% of the downloaded pre-labeled data has been used as training data set and 20% of the same downloaded data has been used as testing data set.

**B.    Preprocessing the Data Set**

The downloaded data set has been preprocessed before feeding into the program to create the Classifier to remove all the non-word characters, to convert into lower case, to remove single characters (e.g. i, a etc.).

import re

import pickle

with open('X.pickle', 'rb') as f:

X=pickle.load(f)

with open('y.pickle', 'rb') as f:

y=pickle.load(f)

corpus = []

for i in range(0,len(X)):

data = re.sub(r'\W', ' ', str(X[i]))

data = data.lower()

data = re.sub(r'\s+[a-z]\s+', ' ', data)

data = re.sub(r'^[a-z]\s+', ' ', data)

data = re.sub(r'\s+', ' ', data)

corpus.append(data)

**C.   Building the BOW, TF-IDF and Logistic Regression Classifier**

Scikit-learn library (a free machine learning library for Python) has been used to create the Classifier.

At first, the bag of words model has been created and later the bag of words model would be converted into TF-IDF model.  To covert the data into bag of words model, classes from Scikit-learn has been used. First, a max feature has been set to 2000 which means 2000 most frequent words would be used as features. The min document frequency would ensure to exclude a word to be considered as 2000 features which appear 3 or less documents (to prevent a word to become a feature which is rare into the set but very popular within a certain document) and the max document frequency would ensure to exclude a word to be considered as 2000 features which appear 60% or more documents (to exclude the most common words like the, an etc.). Then Stop words has been removed that is defined in nltk corpus. Now to form the Bag of Words model the corpus from above steps has been used.

from nltk.corpus import stopwords

from sklearn.feature\_extraction.text import CountVectorizer

from nltk.corpus import stopwords

nltk.download('stopwords')

vectorizer = CountVectorizer(stop\_words=stopwords.words('english'), max\_df = 0.6, min\_df = 3, max\_features = len(X))

X = vectorizer.fit\_transform(corpus).toarray()#will generate 2D array [len(X),len(X)], total number of docs = number of features = len(x)

Finally, TfidfTransformer from sklearn is used to create TF-IDF model from Bag of Words model created earlier.

from sklearn.feature\_extraction.text import TfidfTransformer

# convert BOW to TF-IDF

transformer = TfidfTransformer()

X = transformer.fit\_transform(X).toarray()

Logistic Regression (a Discriminative Classifier) is a classification algorithm (learning algorithm) which is used for binary classification problem. The task of sentiment analysis for this project can be thought as a binary classification problem where the goal is to predict positive or negative sentiment from a given tweets (a sentence in particular). Hence for the purpose of this project, a logistics regression classifier has been built where negative and positive sentiments have been denoted as 0 and 1 respectively.

So, the Binary Response Variable Y Є {0,1} needs to be calculated from the predictor X where X = { x1, x2 ….. x2000} (all the 2000 features)

Hence, for



Logistics Regression can be represented as below:



So essentially using training data T, , parameters  (M=2000) needs to be estimated.

Hence the conditional likelihood estimate is:



and



The goal of Logistic Regression algorithm is to optimize the parameters using training data set. Once the optimal values of the parameters are found by the algorithm, y can be calculated for any unknown sentiment of a new sentence. If y>=0.5, then that sentiment is positive sentiment and if y<0.5, then that sentiment is negative sentiment.

For this project, LogistricRegression from sklearn has been used to build the classifier.

First the input data set has been split as - 80% of available data has been considered as training data and 20% of the available data has been considered as test data

from sklearn.model\_selection import train\_test\_split

text\_train,text\_test,sentiment\_train,sentiment\_test = train\_test\_split(X,y,test\_size=0.2,random\_state=0)#80% training and 20% testing data

Then classifier is built:

from sklearn.linear\_model import LogisticRegression

# create the classifier using logistic regression

classifier = LogisticRegression()

classifier.fit(text\_train,sentiment\_train)

Just to showcase the model performance, confusion\_matrix class from sklearn has been used. With the input data, close to 85% accuracy has been achieved.

from sklearn.metrics import confusion\_matrix

#testing model performance

sentiment\_prediction = classifier.predict(text\_test)

cm = confusion\_matrix(sentiment\_test,sentiment\_prediction)# [[predicted as 0 and actually 0, predicted as 0 and actually 1],

#[predicted as 1 and actually 0, predicted as 1 and actually 1]]

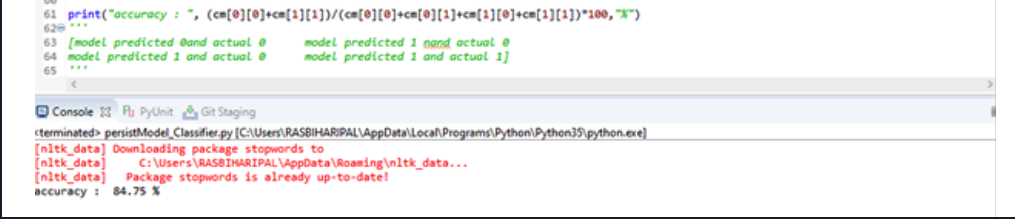
print("accuracy : ", (cm[0][0]+cm[1][1])/(cm[0][0]+cm[0][1]+cm[1][0]+cm[1][1])\*100,"%")

'''

[model predicted 0and actual 0 model predicted 1 nand actual 0

model predicted 1 and actual 0 model predicted 1 and actual 1]

'''



Finally, the model and vectorizer have been stored as pickle file (binary representation of python objects) so that while calculating the real time tweets, the saved model can be used as it is.

from sklearn.feature\_extraction.text import TfidfVectorizer

#store the classifier ....pickle file

with open('classifier.pickle', 'wb') as f:

pickle.dump(classifier,f)

#store the TFIDF vectorizer

vectorizerTFIDF = TfidfVectorizer(stop\_words=stopwords.words('english'), max\_df = 0.6, min\_df = 3, max\_features = len(X))

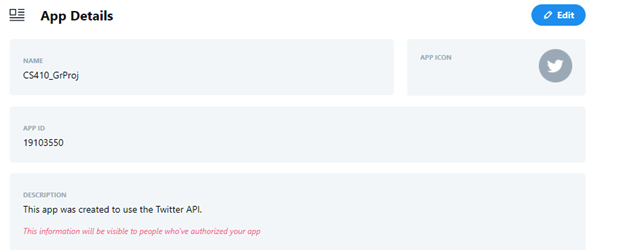
X\_TDIDF = vectorizerTFIDF.fit\_transform(corpus).toarray()

with open('vectorizerTFIDF.pickle', 'wb') as f:

pickle.dump(vectorizerTFIDF,f)

**D.   Fetching the real time tweets**

A developer app has been created in Twitter for the purpose of this project.



The OAuth2 mechanism has been used to make API calls to Twitter API for fetching the recent tweets.

ConsumerKey and ConsumerSecret from the created app are used to generate bearer token in the runtime API call using <https://api.twitter.com/oauth2/token>

Finally, the bearer token is used to call API to perform recent search (returns Tweets from the last 7 days that match a search query) using “recent search” API. <https://developer.twitter.com/en/docs/twitter-api/tweets/search/api-reference/get-tweets-search-recent>

For this project, Max result of 100 has been set to perform recent search. This is to ensure throttling of number of results fetched (as for the basic access, twitter account has 500000 search results/month limitation)

**E.    Performing Sentiment analysis of the fetched tweets**

To perform sentiment analysis in real time, firstly the Logistic Regression Classifier model and TF-IDF vectorizer is loaded from the saved pickle file.

import pickle

import re

with open('classifier.pickle', 'rb') as f:

clf=pickle.load(f)

with open('vectorizerTFIDF.pickle', 'rb') as f:

vectorizer=pickle.load(f)

For a given stock the “recent search” API is used to search recent tweets. After fetching related tweets (max = 100), each tweet is preprocessed to create bag of words and to be represented as TF-IDF vectorizer. Then, prebuilt Logistic Regression Classifier is used to predict the sentiment of each tweets.

for t in tweets\_fetched:

t = re.sub(r'^https://t.co/[a-zA-Z0-9]\*\s',' ',t)

t = re.sub(r'\s+https://t.co/[a-zA-Z0-9]\*\s',' ',t)

t = re.sub(r'\s+https://t.co/[a-zA-Z0-9]\*$',' ',t)

t = t.lower()

t = re.sub(r"that's",'that is',t)

t = re.sub(r"there's",'there is',t)

t = re.sub(r"what's",'what is',t)

t = re.sub(r"where's",'where is',t)

t = re.sub(r"it's",'it is',t)

t = re.sub(r"who's",'who is',t)

t = re.sub(r"i'm",'i am',t)

t = re.sub(r"she's",'she is',t)

t = re.sub(r"he's",'he is',t)

t = re.sub(r"they're",'they are',t)

t = re.sub(r"who're",'who are',t)

t = re.sub(r"shouldn't",'should not',t)

t = re.sub(r"wouldn't",'would not',t)

t = re.sub(r"couldn't",'could not',t)

t = re.sub(r"can't",'can not',t)

t = re.sub(r"won't",'will not',t)

t = re.sub(r'\W', ' ', t)

t = re.sub(r'\d', ' ', t)

t = re.sub(r'\s+[a-z]\s+', ' ', t)

t = re.sub(r'\s+[a-z]$', ' ', t)

t = re.sub(r'^[a-z]\s+', ' ', t)

t = re.sub(r'\s+', ' ', t)

sentiment = clf.predict(vectorizer.transform([t]).toarray())

Finally, total positive and negative sentiments are calculated for total tweets fetched in runtime and the final sentiment of a particular stock has been calculated as positive = 1 (if the positive percentage > 65%), negative = -1 (if the positive percentage < 35%) or neutral = 0 (if the positive percentage is in between 35% and 65%).

if (tot\_positive+tot\_negetive)>0 :

positive\_percentage = tot\_positive/(tot\_positive+tot\_negetive)

print("positive\_percentage :",positive\_percentage\*100, "%")

if positive\_percentage>0.65 :

print("stock is buy")

return 1

elif positive\_percentage<0.35 and positive\_percentage>0:

print("stock is sell")

return -1

else:

print("stock is neutral")

return 0

Following end point has been built to provide twitter sentiment analysis result to a stock.

Definition

* Get Twitter sentiment for given stock - 1: Positive, 0: Neutral, -1: Negative

GET /stock/sentiments/<market>/<stock\_symbol>

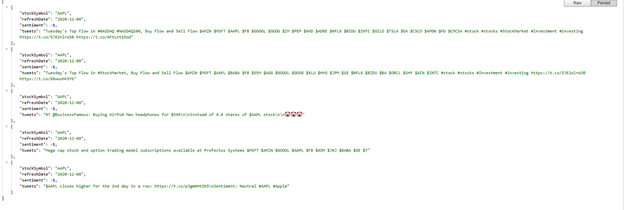
e.g. <http://localhost:5000/stock/sentiments/NASDAQ/AAPL>

Response

* 200 OK on success

json { "stockSymbol": "AAPL", "refreshDate": "2020-11-25", "sentiment": 1 }

Sample run:



### ****Recommender System****

**Approach**

The basic approach to the recommendation engine is to use a similarity function to compare pre-identified features of all stocks in our corpus against user-provided stock symbol features. While doing so, we are implementing one of the methods for building recommendation engine - Content based recommendation or Item similarity, which was part of Week 6 lecture of course CS 410 Text Information System.

Recommender System recommends to the user the top 5 stock symbols which are most similar to the user provided stock symbol in terms of those underlying features and the similarity function. For the initial, ***Cosine similarity*** is used as similarity function and the following financial/economic features were selected as underlying features for similarity calculation.

1. company is in S&P 500
2. company profitability over last three years
3. revenue growth for last three years
4. current market analyst ratings
5. sector
6. gross profit per market cap

Analyst ratings are calculated by rating\_system and same is reused also for recommendation system. Other features for the stock are calculated or derived using the financial data of the companies from year 2018. Source: Kaggle dataset available at <https://www.kaggle.com/cnic92/200-financial-indicators-of-us-stocks-20142018>.

As part of the exercise, significant amount of effort was used to identify a single data source for financial or economic data and also to understand various financial terms to find the right feature. Based on our current understanding of a company performance and its relation to stock price, we used above listed features. The features might be further optimized with right guidance from a financial analyst or person with insight of stock market and factors influencing the investor decisions.

## ****Libraries and Datasets****

**Python Libraries**

Flask

flask\_cors

markdown

beautifulsoup4

selenium

pandas

pymongo

tinydb

misaka

nltk

requests

sklearn

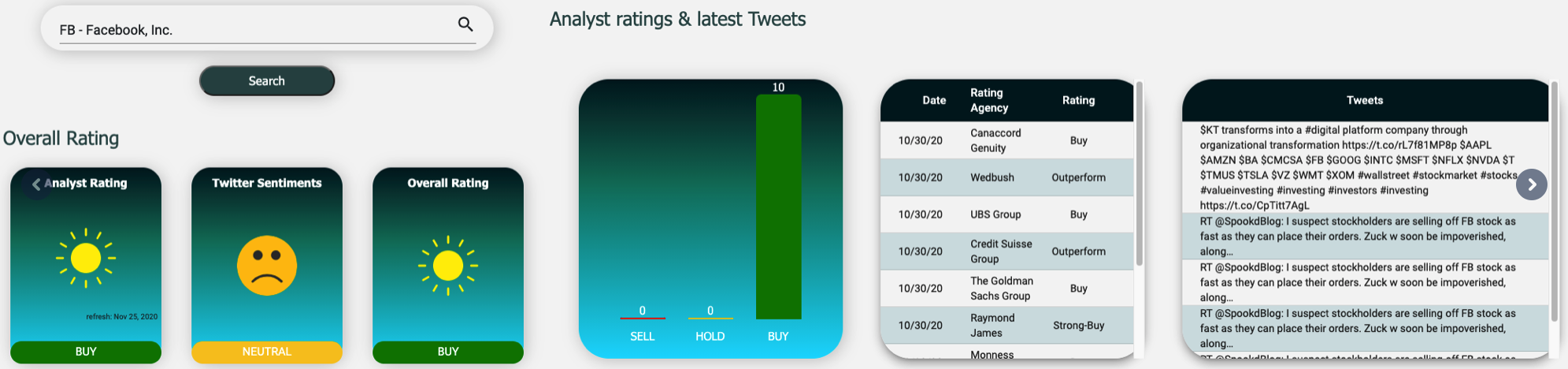
flask-jsonpify

**Datasets**

* <https://www.marketbeat.com/>
* <https://www.cs.cornell.edu/people/pabo/movie-review-data/>
* <https://www.kaggle.com/cnic92/200-financial-indicators-of-us-stocks-20142018?select=2018_Financial_Data.csv>
* <https://developer.twitter.com>

## ****Verification****

The result of the recommender system was verified based on manual verification process by selecting some stocks at random and checking the result for individual section e.g., ratings, sentiments and overall ratings. For example, Overall rating for Facebook (FB) stock is BUY. Looking at individual ratings component, the result seems reasonable.

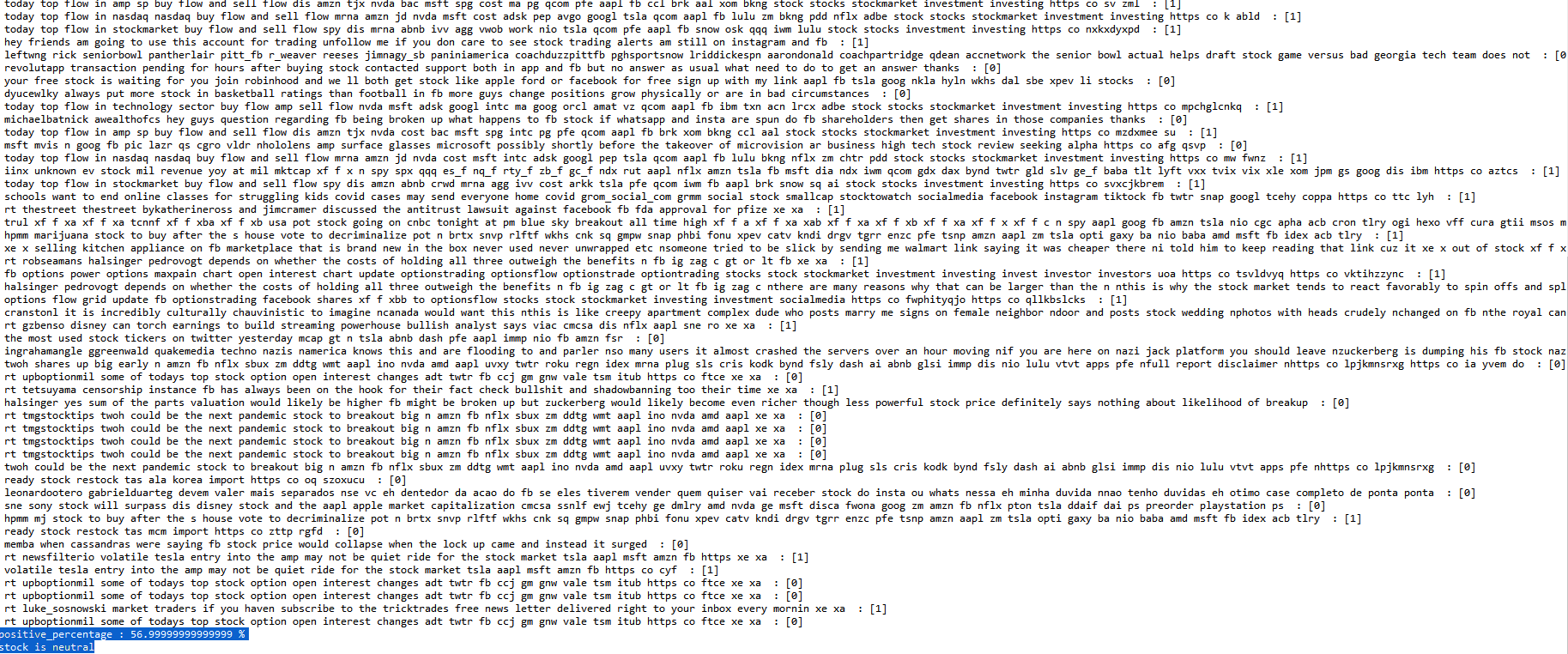


**Analyst Ratings**: BUY

**Verification**: In the list of market analyst ratings, all 10 analysts have rated it as BUY. Therefore, the aggregated rating is BUY.

**Tweet Sentiments**: Neutral

**Verification**: From the latest 100 tweets fetched (as shown in below screenshot), it seems reasonable to have a NEUTRAL sentiment



**Overall Rating**: BUY

**Verification**: Combining Analyst Ratings “Buy” with Twitter Sentiments “NEUTRAL”, as per listed decision table, the result is BUY.

## ****Conclusion and Future Work****

Overall, the Stock recommending system as described in this project report has yielded satisfactory result in recommending rating (e.g., buy, hold, sell) for the user entered Stock symbol and recommending five similar stocks. The result has been validated by some popular stocks. One of key challenges faced in building the system is the access on right financial data as well as good training and testing set to train and test the model for sentiment analysis. The choice we had to manually create the required data. However, to optimize the time and resource available, it has been decided to use readily available data in the web which might have not yielded the perfect recommendation.

There are several interesting directions for the future version of recommender system. First, the overall functionality can be improved by considering user input of sectors and providing recommendation and trends specific to that sector. A second direction involves defining the test and train data and possibly human labeling twitter feed (for sentiment analysis) just for tweets related to stock market and use the same to train and test the model. Given the current trend of machine learning algorithms, a third interesting research direction is to explore the timeseries data for stock adjacency. Finally, overall score can be improved further to consider more analysts’ reports and microblogging websites and come up with more recommender categories i.e., Strong Buy, Buy, Hold, Sell, Strong Sell etc.

At the end, it has been a great journey of ideation and learning in a collaborative manner to design and develop the stock recommender system. We thank different analysts’ websites to make their ratings available publicly and twitter to grant access of real time tweets through their public API and last but not the least, we thank professor ChengXiang Zhai and all our TAs and all the reviewers to give the direction needed and to provide the valuable feedback.

## ****References****

[1] Bo Pang and Lillian Lee. 2004. A sentimental education: sentiment analysis using subjectivity summarization based on minimum cuts. In Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics (ACL '04). Association for Computational Linguistics, USA, 271–es. DOI:<https://doi.org/10.3115/1218955.1218990>

[2] Carbone, N. (2020, January 18). 200+ Financial Indicators of US stocks (2014-2018). Retrieved December 11, 2020, from <https://www.kaggle.com/cnic92/200-financial-indicators-of-us-stocks-20142018>