Sentiment Analysis and Financial Applications

Yuwen Jin 10455173

1. Overview

In this project, I apply term frequency - inverse document frequency method to transform text to digital vectors, and use stacked generalization to predict sentiment label. Then I use this method to classify sentiments of some financial related twitters. By comparing the tweets sentiment and the coming market movement I can say how reliable these twitter may be and which seem to be more reliable among the group I chose. On the other hand, I put twitter sentiment information into a time series model together with market indexes in order to better predict stock direction.

2. Data

2.1 Data collection

One part of my data is indexes and stock price downloaded with 'yfinance' package in Python. Indexes indicates stock market situation, which stock price(movement) is what we actually care about.

The other part is text data from some financial related twitter accounts during the target period. I got 30 candidates and finally chose 6 twitters to analyze: MarketWatch, business, WSJ, TheEconomist, nytimes, and MorganStanley. The twitter I select are all financial related, so I assume that they can represent a segment of market perspective.

Besides, I got some corpus from different fields: movie review, twitter text and

financial contents. Because I'm analyzing twitter text, and I did try for some times, finally I use the twitter corpus as training data set.

2.2 Data process

To put text into classifiers, I need to do vectorization first. There are lots of ways to realize it such as Word2Vec and ELMo. In this project I use Term Frequency - Inverse Document Frequency method because it works faster and is friendly to starter. In the future I could try some other advanced methods to see if I could improve accuracy.

Also, in order to pair sentiment score with market movements, I first group tweets by trading days, then calculate mean score of each group as the sentiment of the coming trading day.

For stock data I calculate log returns because mostly they are stationary and can still represent the stock movements.

3. Model

3.1 Model selection

In sentiment analysis, I use stacked generalization to predict the sentiment label. In the first layer of my stacking model, I use 5 base models: SVM, Random Forest, Gradient Boosting, KNN and Naive Bayesian. Then in the second layer I use Decision Tree as my meta model. All the models here are common and useful classifiers.

Stacked generalization is one way to realize ensemble learning. With this method, on the one hand I could include several models in one time and collect their advantages, on the other hand the concept here is trying to void overfitting. Meta-learner instead

of voting/averaging in Bagging and Boosting could reduce bias and variance.

Then, in stock trend predicting, I use boosting model. Because it works well in my previous project.

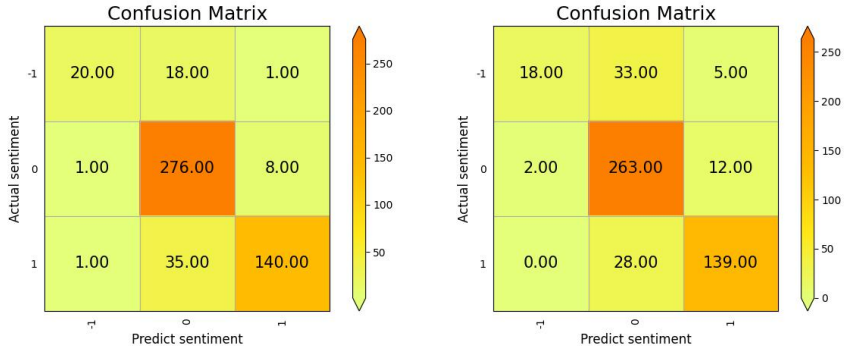
3.2 Fit model

In stacked generalization model, I use k fold cross validation to avoid overfitting. Initially I wrote 2 separate functions to realize the sentiment analysis, then in order to 'save model' and save time, I transform in into a class. Then every time I fit a model, I don't have to train it again when do classification on different test sets.

While predicting stock movements using boosting model, the learning rate is selected based on the accuracy score on validation set during iteration, and I set max_features as 2 because generally it's square root of feature amount. Since my data is time series, I divide training set in order instead of using random validation while fitting model.

3.3 Model test

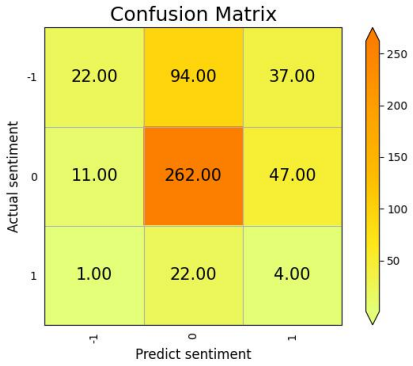
To see how my model performs on twitter sentiment analysis before further application, I use it to predict another presorted twitter data set and plot a confusion matrix. Correct classification are on the diagonal.



In the 2 attempts I show above, my accuracy is.86.9% and 85% respectively. Meanwhile, the distribution of observation and predictions are not completely

concentrated in one class. Then I have reason to believe that this model can classify sentiment of twitter contents to some extend.

However, it doesn't work that well when I use it to classify other financial related texts. In the attempt below, I only get 57.6% correct classification



3.4 Classify and predict

For each twitter records in testing data set I give it a classification label generated by my stacking model. '1' stands for positive, '-1' means negative and '0' is neutral. When a twitter has more than 1 tweets in one counting period(between 2 trading days), I use mean value of the polarities as its sentiment score for that day. In this case I could pair sentiment record with stock movement and see their relationship, also it can be used to do prediction.

This is the data frame of average polarized sentiment I got.

```
>>> sentiment_df
            MarketWatch  business  ...   nytimes  MorganStanley
2019-10-01     0.074380  0.125984  ...  0.118812       0.250000
2019-10-02     0.025000  0.121795  ...  0.120879       0.571429
2019-10-03     0.135135  0.106061  ...  0.172973       0.500000
2019-10-04     0.037037  0.113924  ...  0.179775       0.571429
2019-10-07     0.141304  0.088571  ...  0.073171       0.000000
               ...       ...  ...       ...            ...
2020-09-23     0.076923  0.053459  ...  0.222222       0.500000
2020-09-24     0.084249  0.073200  ...  0.114833       0.333333
2020-09-25     0.107143  0.065217  ...  0.112360       0.285714
2020-09-28     0.023438  0.100000  ...  0.071429       0.363636
2020-09-29     0.035714  0.089404  ...  0.023810       0.000000

[252 rows x 6 columns]
```

4. Result and Analysis

4.1 Sentiment analysis

To get an appropriate model, I tried 3 corpus One is movie reviews, one is financial related texts, and the other one is twitter data. It takes long to fit one time so I only use 10,000 records in corpus as training data sets. The twitter corpus works the best on accuracy among the 3 when test data also comes from twitter. So there may be some certain commonalities in what people say on Twitter.

As I describe in model test part, my sentiment analysis model works well in classifying tweets, but its scope of application is limited because of the corpus I'm using, the train data size I have, and may because of the model I choose.

The vectorization method I use cares more about the frequency that the word show up in the document, but will ignore contextual semantics. Considering topic model(such as LDA), and word embedding(such as word2vec) should help me develop more accurate and efficient model with a wider range of use. As far as I know, BERT model, which I learned from classmates' presentation and tried to get it run on my computer in this 2 days, can reach accuracy of 88% right after the first epoch. However, it takes much longer even we use GPU through Google Colaboratory, and we'll need its pre trained models.

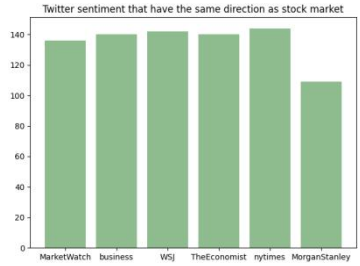
Also, my model may cause some loss on information since it only gives polarized label but doesn't provide exact scores like what 'snowmlp' package does.

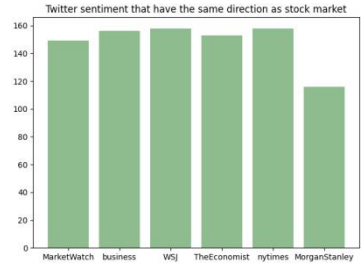
4.2 Twitter account reliability

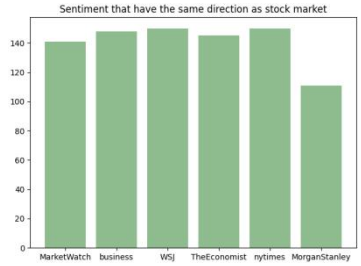
The idea here is to see the relationship between stock movement and sentiment label

of tweets. If they are highly related then we have reason to believe in the tested twitter accounts to some extend.

Below is the bar plot showing how many days the twitter account's sentiment has the same direction label as stock market(S&P 500, NASDAQ and DOW in order) in one year(252 trading days).



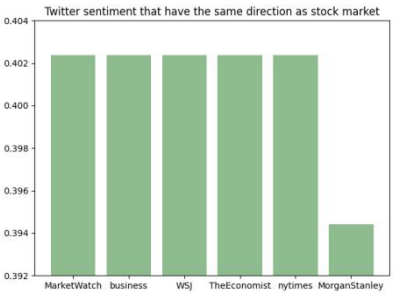




My consider is that these tweets may not be in sync with the market, but still need some other work to see whether it's lagging behind or ahead depending on the. Or we could get some impressions on this from the next part, stock movement prediction.

What's interesting here is that most of the probability I got is around 40%, among which Morgan Stanley is slightly farther away from 50% so I'm using Morgan Stanley data as one variable in stock predicting part. Because it's a binary options and it's not 50% or very close to 50%, the attitude of these twitter accounts do have some relationship with market trend and may be useful if we consider it as one of the explanatory variables while analyzing stock movement.

The diagram below shows the percentage of positive relationship.



4.3 **Predict stock movement**

If I use the same day data to fit the time series model, we see that these variables(target stock and 4 indexes together twitter sentiment score) are highly related, because I get high model accuracy. Aside from market indexes, the 3 model below takes my sentiment score, snowNLP sentiment, and no sentiment respectively.

Confusion Matrix:
[[15 4]
[1 17]]
Classification Report

	precision	recall	f1-score	support
-1.0	0.94	0.79	0.86	19
1.0	0.81	0.94	0.87	18
accuracy			0.86	37
macro avg	0.87	0.87	0.86	37
weighted avg	0.88	0.86	0.86	37

Confusion Matrix:
[[15 4]
[2 16]]
Classification Report

	precision	recall	f1-score	support
-1.0	0.88	0.79	0.83	19
1.0	0.80	0.89	0.84	18
accuracy			0.84	37
macro avg	0.84	0.84	0.84	37
weighted avg	0.84	0.84	0.84	37

Confusion Matrix:
[[16 3]
[2 16]]
Classification Report

	precision	recall	f1-score	support
-1.0	0.89	0.84	0.86	19
1.0	0.84	0.89	0.86	18
accuracy			0.86	37
macro avg	0.87	0.87	0.86	37
weighted avg	0.87	0.86	0.86	37

Also I did one day ahead prediction with the same independent variables as the previous three. Models below are arranged in the same order as the previous row, too.

Confusion Matrix:
[[14 6]
[5 13]]
Classification Report

	precision	recall	f1-score	support
-1.0	0.74	0.70	0.72	20
1.0	0.68	0.72	0.70	18
accuracy			0.71	38
macro avg	0.71	0.71	0.71	38
weighted avg	0.71	0.71	0.71	38

Confusion Matrix:
[[17 3]
[12 6]]
Classification Report

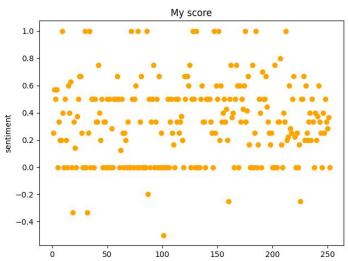
	precision	recall	f1-score	support
-1.0	0.59	0.85	0.69	20
1.0	0.67	0.33	0.44	18
accuracy			0.61	38
macro avg	0.63	0.59	0.57	38
weighted avg	0.62	0.61	0.58	38

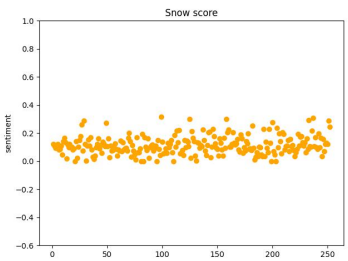
Confusion Matrix:
[[14 6]
[7 11]]
Classification Report

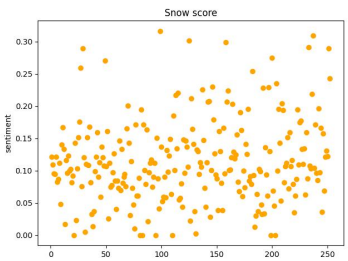
	precision	recall	f1-score	support
-1.0	0.67	0.70	0.68	20
1.0	0.65	0.61	0.63	18
accuracy			0.66	38
macro avg	0.66	0.66	0.66	38
weighted avg	0.66	0.66	0.66	38

The accuracy is not bad, and adding my polarity average does improve model accuracy. Therefore, I'll say my sentiment analysis method works, and sentiment information can help in predicting stock movements.

For some reason snownlp doesn't work very well this time. Observe the scatter plot, sentiment score with snownlp is more concentrated on a relatively small range than the score I got, and it looks more disordered even though I put it in order of date.







5. Future work

5.1 I should try other method to do vectorization, and try other models as well so that I may do better in sentiment analysis.

5.2 I could write a trading strategy to see how much I can earn or will lose if I trade based models in this project.

5.3 I should expand my 'StackedGeneralization' class more and make it more intelligent. For example I could allow it to decide which sub group of the input candidate models to use.

Appendix

(some core code in my project)

1. get_data.py
2. stacking.py
3. Apply1.py (Twitter account reliability)
4. Apply2.py (stock movement predict)

```
import yfinance as yf
import pandas as pd
import snscrape.modules.twitter as tw
import sys
import string


candidates = ['MarketWatch', 'business', 'YahooFinance', 'TechCrunch', 'WSJ', 'Forbes', 'FT', 'TheEconomist', 'nytimes',
                'Reuters', 'GerberKawasaki', 'jimcramer', 'TheStreet', 'TheStalwart', 'TruthGundlach', 'CarlCIcahn',
                'ReformedBroker', 'benbernanke', 'bespokeinvest', 'BespokeCrypto', 'stlouisfed', 'federalreserve',
                'GoldmanSachs', 'ianbremmer', 'MorganStanley', 'AswathDamodaran', 'mcuban', 'muddywatersre', 'StockTwits',
                'SeanaNSmith']


def get_tweet_txt(id_list: list, start: str = '2020-06-01', end: str = '2020-10-01'):
    for ID in id_list:
        f = open('./intermediate file/' + ID + '.txt', 'w', encoding='utf-8')

        for tweet in tw.TwitterSearchScraper(
                query="from:" + ID + " since:" + start + " until:" + end).get_items():
            date_str = tweet.date.strftime("%Y-%m-%d %H:%M:%S%z")
            date_str = date_str[:-2] + ":" + date_str[-2:]
            f.write(date_str + "|" + tweet.content + "\n")
        f.close()


def get_tweet_csv(id_list: list, start: str = '2020-06-01', end: str = '2020-10-01', save=1):
    for ID in id_list:
        output_df = pd.DataFrame(data=None, columns=['Twitter', 'Date', 'Content'])
        date, content = [], []
        for tweet in tw.TwitterSearchScraper(
                query="from:" + ID + " since:" + start + " until:" + end).get_items():
            date_str = tweet.date.strftime("%Y-%m-%d %H:%M:%S%z")
            date_str = date_str[:10]
            date.append(date_str)
            content.append(tweet.content)
        output_df['Date'], output_df['Content'] = date, content
        output_df['Twitter'] = ID
        if save == 1:
            output_df.to_csv('./intermediate/' + ID + '.csv', index=False)
    return output_df


def extract_url(content: str):
    out_index, n = [], 0
    lst = content.split()
    for i in range(len(lst)):
        if lst[i].find('https://') == 0:
            out_index.append(i)
    for j in out_index:
        lst.pop(j-n)
        n += 1
    return lst


def clean_punctuation(content: str):
    lst = content.split()
    for i in range(len(lst)):
        while lst[i][0] in string.punctuation:
            lst[i] = lst[i][1:]
        while lst[i][-1] in string.punctuation:
            lst[i] = lst[i][:-1]
    return ' '.join(lst)


def combine_csv(name_list: list):
    merged_df, temp_df = pd.DataFrame(data=None, columns=['Twitter', 'Date', 'Content']), None
    for ID in name_list:
        try:
            temp_df = pd.read_csv('./intermediate/' + ID + '.csv')
        except IOError:
            sys.stderr.write("You don't have corresponding file.")
            exit(1)
        for i in range(temp_df.shape[0]):
            raw_content = extract_url(temp_df['Content'][i])
            temp_df['Content'][i] = ' '.join(raw_content)
        merged_df = merged_df.append(temp_df)
    merged_df.to_csv('./output data/merged.csv', index=False)
    return merged_df


def clean_punctuation_new(content):
    out = []
    for word in content:
        while word[0] in string.punctuation:
            word = word[1:]
        while word[-1] in string.punctuation:
```

```
                word = word[:-1]
            out.append(word)
    return out


def clean_csv(ID: str):
    temp_df = pd.read_csv('./intermediate/' + ID + '.csv')
    for i in range(temp_df.shape[0]):
        raw_content = extract_url(temp_df['Content'][i])
        temp_df['Content'][i] = ' '.join(raw_content)
    return temp_df


if __name__ == "__main__":
    tweet_id = ['MarketWatch', 'business', 'WSJ', 'TheEconomist', 'nytimes', 'MorganStanley']
    get_tweet_csv(tweet_id, '2019-10-01', '2020-10-01')
    combine_tweet_csv = combine_csv(tweet_id)
```

back

```
from predict import *
import numpy as np
from sklearn.model_selection import KFold
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.tree import DecisionTreeClassifier


class StackedGeneralization:
    def __init__(self, classifier, meta=DecisionTreeClassifier(), kf_n=5):
        self.classifier = classifier
        self.meta = meta
        self.to_vector = TfidfVectorizer(min_df=5, max_df=0.8, sublinear_tf=True, use_idf=True)
        self.kf_n = kf_n
        self.kf = KFold(n_splits=kf_n, random_state=49, shuffle=True)
        self.meta_train = None
        self.meta_test = None
        self.base_memory = None
        self.y_train = None

    def Base_learner_train(self, classifier, x_train, y_train):
        second_train = np.zeros((len(x_train)))
        temp_x_train = self.to_vector.fit_transform(x_train)
        temp_y_train = y_train
        for i, (train_index, test_index) in enumerate(self.kf.split(temp_x_train)):
            kf_x_train = temp_x_train[train_index]
            kf_y_train = temp_y_train[train_index]
            kf_x_test = temp_x_train[test_index]
            classifier.fit(kf_x_train, kf_y_train)
            second_train[test_index] = classifier.predict(kf_x_test)
        self.meta_train = second_train.reshape(-1, 1)
        return second_train.reshape(-1, 1)

    def Base_learner_test(self, classifier, x_test):
        second_test = np.zeros((len(x_test)))
        second_test_prep = np.empty((5, len(x_test)))
        temp_x_test = self.to_vector.transform(x_test)
        for i in range(self.kf_n):
            second_test_prep[i, :] = classifier.predict(temp_x_test)
        second_test[:] = second_test_prep.mean(axis=0)
        self.meta_test = second_test.reshape(-1, 1)
        return second_test.reshape(-1, 1)

    def fit(self, x_train=None, y_train=None, new: str = 'no'):
        self.y_train = y_train
        train_sets= []
        for classifier in self.classifier:
            train_set = self.Base_learner_train(classifier, x_train, y_train)
            train_sets.append(train_set)
        second_layer_train = np.concatenate([result_set.reshape(-1, 1) for result_set in train_sets], axis=1)
        self.base_memory = second_layer_train

    def predict(self, x_test):
        test_sets = []
        for classifier in self.classifier:
            test_set = self.Base_learner_test(classifier, x_test)
            test_sets.append(test_set)
            second_layer_test = np.concatenate([y_test_set.reshape(-1, 1) for y_test_set in test_sets], axis=1)

        # second level model
        self.meta.fit(self.base_memory, self.y_train)
        prediction = self.meta.predict(second_layer_test)
        return prediction



# back
```

```
from get_data import *
from predict import *
import yfinance as yf
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB
import matplotlib.pyplot as plt
from snownlp import SnowNLP
from stacking import *

#
# class StackedGeneralization:
#     def __init__(self, classifier, meta=DecisionTreeClassifier(), kf_n=5):
#         self.classifier = classifier
#         self.meta = meta
#         self.to_vector = TfidfVectorizer(min_df=5, max_df=0.8, sublinear_tf=True, use_idf=True)
#         self.kf_n = kf_n
#         self.kf = KFold(n_splits=kf_n, random_state=49, shuffle=True)
#
#     def Base_learner_output(self, classifier, x_train, y_train, x_test):
#         second_train = np.zeros((len(x_train)))
#         second_test = np.zeros((len(x_test)))
#         second_test_prep = np.empty((5, len(x_test)))
#         temp_x_train = self.to_vector.fit_transform(x_train)
#         temp_y_train = y_train
#         temp_x_test = self.to_vector.transform(x_test)
#         for i, (train_index, test_index) in enumerate(self.kf.split(temp_x_train)):
#             kf_x_train = temp_x_train[train_index]
#             kf_y_train = temp_y_train[train_index]
#             kf_x_test = temp_x_train[test_index]
#             classifier.fit(kf_x_train, kf_y_train)
#             second_train[test_index] = classifier.predict(kf_x_test)
#             second_test_prep[i, :] = classifier.predict(temp_x_test)
#
#         second_test[:] = second_test_prep.mean(axis=0)
#         return second_train.reshape(-1, 1), second_test.reshape(-1, 1)
#
#     def stacking(self, x_train, y_train, x_test):
#         train_sets, test_sets = [], []
#         # run base leaner
#         for classifier in classifiers:
#             train_set, test_set = self.Base_learner_output(classifier, x_train, y_train, x_test)
#             train_sets.append(train_set)
#             test_sets.append(test_set)
#
#         # get input of second layer
#         second_layer_train = np.concatenate([result_set.reshape(-1, 1) for result_set in train_sets], axis=1)
#         second_layer_test = np.concatenate([y_test_set.reshape(-1, 1) for y_test_set in test_sets], axis=1)
#
#         # second level model
#         self.meta.fit(second_layer_train, y_train)
#         prediction = self.meta.predict(second_layer_test)
#         return prediction


def polarize(seq):
    out = seq
    out[out < 0] = -1
    out[out > 0] = 1
    return out


def polarize_2(seq):
    out = seq
    out[out < 0.5] = -1
    out[out == 0.5] = 0
    out[out > 0.5] = 1
    return out


if __name__ == "__main__":
    # corpus
    stack_train = pd.read_csv('tweet_sentiment.csv').dropna(axis=0, how='any')
    stack_train.columns = ['Content', 'Sentiment']
    stack_train.reset_index(drop=True, inplace=True)

    # data process
    x_training = stack_train['Content'][:10000]
    y_training = stack_train['Sentiment'][:10000]
```

```
    stock_df = yf.download("^GSPC", start='2019-10-01', end='2020-09-30')
    timespan = [i.strftime("%Y-%m-%d") for i in stock_df.index.to_series()]

    # classifier
    classifiers = [svm.SVC(kernel='linear'),
                   RandomForestClassifier(),
                   GradientBoostingClassifier(random_state=10),
                   KNeighborsClassifier(n_neighbors=5, p=2, metric='minkowski'),
                   MultinomialNB()]

    # ------------------------------------------------------------------------------

    twitter_id = ['MarketWatch', 'business', 'WSJ', 'TheEconomist', 'nytimes', 'MorganStanley']
    stacking_model = StackedGeneralization(classifiers)
    stacking_model.fit(x_training, y_training)

    sentiment_df = pd.DataFrame(data=None, columns=twitter_id)
    for twitter in twitter_id:
        temp_df = clean_csv(twitter)
        x_testing = temp_df['Content']
        predictions = stacking_model.predict(x_testing)
        temp_df['Sentiment_stacking'] = predictions

        mark, stack = [], []
        day = '2020-08-28'
        for day in timespan:
            lst = temp_df[temp_df['Date'].isin([day])]
            if len(lst) == 0:
                mark.append(-1)
            else:
                mark.append(np.min(lst.index))
                print(str(day) + str(np.min(lst.index)))
        mark.reverse()
        mark.append(len(temp_df))
        for i in range(len(mark)):
            if mark[i] == -1:
                mark[i] = mark[i-1]
        for i in range(len(mark) - 1):
            span = [i for i in range(mark[i], mark[i + 1])]
            lst = temp_df.iloc[span, :]
            if lst is not None:
                stack.append(np.mean(lst['Sentiment_stacking']))
            else:
                stack.append(0)
        sentiment_df[twitter] = stack

    # sentiment_df.to_csv('./output/sentiments.csv', index=False)

    # ------------------------------------------------------------------------------

    sentiment_df = pd.read_csv('./output/sentiments.csv')
    # sentiment_df = pd.read_csv('./output/snow_sent.csv')

    market_return = stock_df['Adj Close'].apply(np.log).diff().dropna()
    market_movement = list(polarize(market_return))

    correct_records = []
    for twitter in twitter_id:
        pred = list(polarize_2(sentiment_df[twitter][1:]))
        cm = confusion_matrix([int(i) for i in market_movement], [int(i) for i in pred], sample_weight=None)
        correct_records.append(np.diag(cm).sum())

    plt.bar(range(len(correct_records)), correct_records, color='darkseagreen', tick_label=twitter_id)
    plt.title('Twitter sentiment that have the same direction as stock market')
    plt.show()

    plt.bar(range(len(correct_records)), [i/251 for i in correct_records], color='darkseagreen', tick_label=twitter_id)
    plt.title('Twitter sentiment that have the same direction as stock market')
    plt.ylim(ymax=0.404, ymin=0.392)
    plt.show()


# back
```

```
import yfinance as yf
import pandas as pd
import numpy as np
from snownlp import SnowNLP
from test_stacking import stacking
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import GradientBoostingClassifier
import math


# function prepare
def get_return(symbol: list, startDay: str = "2020-06-01", endDay: str = "2020-10-01"):
    out = []
    for i in symbol:
        temp_df = yf.download(i, start=startDay, end=endDay)
        temp_return = temp_df['Adj Close'].apply(np.log).diff().dropna()
        out.append(np.array(temp_return).reshape(1, -1))
    return out


def polarize(ser):
    out = []
    for i in ser:
        if i > 0:
            out.append(1)
        if i < 0:
            out.append(-1)
        else:
            out.append(0)
    return out


def fin_model(df, sentiment: int = 1):
    bound1, bound2 = int(df.shape[0]*0.65), int(df.shape[0]*0.8)
    print(bound1)
    print(bound2)
    train = df[:bound2]
    test = df[bound2:]
    y_train = pd.DataFrame()
    y_train['KSS'] = train['KSS']
    train.drop(labels="KSS", axis=1, inplace=True)
    train.drop(labels=train.columns[0], axis=1, inplace=True)

    for i in range(len(y_train)):
        if y_train['KSS'][i] < 0:
            y_train['KSS'][i] = -1
        else:
            y_train['KSS'][i] = 1
    test.drop(labels="KSS", axis=1, inplace=True)
    test.drop(labels=train.columns[0], axis=1, inplace=True)

    if sentiment == 1:
        n = 5
        test.drop(labels="sentiment_snow", axis=1, inplace=True)
        train.drop(labels="sentiment_snow", axis=1, inplace=True)
    elif sentiment == 2:
        n = 5
        test.drop(labels="sentiment_stack", axis=1, inplace=True)
        train.drop(labels="sentiment_stack", axis=1, inplace=True)
    else:
        n = 4
        test.drop(labels="sentiment_stack", axis=1, inplace=True)
        train.drop(labels="sentiment_stack", axis=1, inplace=True)
        test.drop(labels="sentiment_snow", axis=1, inplace=True)
        train.drop(labels="sentiment_snow", axis=1, inplace=True)

    scaler = MinMaxScaler()
    x_train = scaler.fit_transform(train)
    # x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.3)
    x_train_1 = x_train[:bound1]
    x_val = x_train[bound1:]
    y_train_1 = [i[0] for i in y_train.values[:bound1].tolist()]
    y_val = [i[0] for i in y_train.values[bound1:].tolist()]

    learning, accuracy = 0, 0
    lr_list = np.arange(.01, 1, 0.001)
    for learning_rate in lr_list:
        gb_clf = GradientBoostingClassifier(n_estimators=n, learning_rate=learning_rate,
                                            max_features=2, max_depth=6, random_state=0)
        gb_clf.fit(x_train_1, y_train_1)
        if gb_clf.score(x_val, y_val) > accuracy:
            learning = learning_rate
```

```
                accuracy = gb_clf.score(x_val, y_val)

            # print("Learning rate: ", learning_rate)
            # print("Accuracy score (training): {0:.3f}".format(gb_clf.score(x_train, y_train)))
            # print("Accuracy score (validation): {0:.3f}".format(gb_clf.score(x_val, y_val)))

        gb_clf2 = GradientBoostingClassifier(n_estimators=n, learning_rate=learning,
                                             max_features=2, max_depth=6, random_state=0)
        gb_clf2.fit(x_train_1, y_train_1)
        prediction = gb_clf2.predict(x_val)

        print("Confusion Matrix:")
        print(confusion_matrix(y_val, prediction))

        print("Classification Report")
        print(classification_report(y_val, prediction))


if __name__ == "__main__":

        # stock data prepare
        symbols = ["^GSPC", "^IXIC", "^DJI", "VIXY", "KSS"]
        intermediate = get_return(symbols, "2019-10-01", "2020-09-30")
        SP500, NASDAQ, DOW, VIXY, KSS = intermediate[0][0], intermediate[1][0], intermediate[2][0], \
                                        intermediate[3][0], intermediate[4][0]
        stock_df = yf.download("KSS", start="2019-10-01", end="2020-09-30")
        timespan = [i.strftime("%Y-%m-%d") for i in stock_df.index.to_series()]

        # Prepare a data frame to predict
        Analyze = pd.DataFrame(data=KSS, columns=['KSS'])
        Analyze['S&P 500'] = SP500
        Analyze['NASDAQ'] = NASDAQ
        Analyze['DOW'] = DOW
        Analyze['VIXY'] = VIXY
        sentiment = pd.read_csv('./output/sentiments.csv')
        snow = pd.read_csv('./output/snow_sent.csv')
        Analyze['sentiment_snow'] = np.array(snow['MorganStanley'][:-1]).reshape(-1, 1)
        Analyze['sentiment_stack'] = np.array(sentiment['MorganStanley'][:-1]).reshape(-1, 1)
        Analyze.index = timespan[1:]
        Analyze.to_csv('./output/analyze.csv')

        # ------------------------------------------------------------------------

        df_cor = pd.read_csv("./output/analyze.csv")
        # make dependent variable one day ahead
        df_1ahead = pd.DataFrame(Analyze.iloc[:-1, 1:])
        df_1ahead['KSS'] = KSS[1:]
        # df = pd.read_csv("./output data/analyze.csv")

        fin_model(df_cor,0)
        fin_model(df_cor)
        fin_model(df_cor, 2)
        fin_model(df_1ahead, 0)
        fin_model(df_1ahead)
        fin_model(df_1ahead, 2)
```

back