

LING131 Final Project

Sibo Zhu, Yuchen Zhang, Daniel Zhang, Wenxiao Xiao

11 December 2018

Abstract

Our final project is Two Sigma: Using News to Predict Stock Movements, and we used the content of news analytic to predict stock price performance. The ubiquity of data today enables investors at any scale to make better investment decisions. The challenge is ingesting and interpreting the data to determine which data is useful, finding the signal in this sea of information. We present our machine learning algorithm the LightGBM, which provides a solid performance on the Kaggle Two Sigma competition. We developed a comprehensive feature engineering process that is tailored towards the specific environment of predicting stock price performance along with our construction of the LightGBM algorithm. Altogether, it brings our leaderboard ranking to be top 10%, with the public score of 0.69540.

Introduction

Data Analysis

Since this is a Kaggle exclusive dataset, all the data can only be accessed via the Kaggle server and couldn't be downloaded. Therefore all the data analysis below is done on the Kaggle server:

Market Data

We have an interesting dataset which contains stock prices for many companies over a decade! For now let's have a look at the data itself. We can see long-term trends, appearing and declining companies and many other things.

```
print(f'{market_train_df.shape[0]} samples and {market_train_df.shape[1]} features in the training market dataset.')
```

```
4072956 samples and 16 features in the training market dataset.
```

Now, let's look at these price drops in details.

```

market_train_df['price_diff'] = market_train_df['close'] - market_train_df['open']
grouped = market_train_df.groupby('time').agg({'price_diff': ['std', 'min']}).reset_index()
print(f"Average standard deviation of price change within a day in
      {grouped['price_diff']['std'].mean():.4f}.")

```

Average standard deviation of price change within a day in 1.0335.

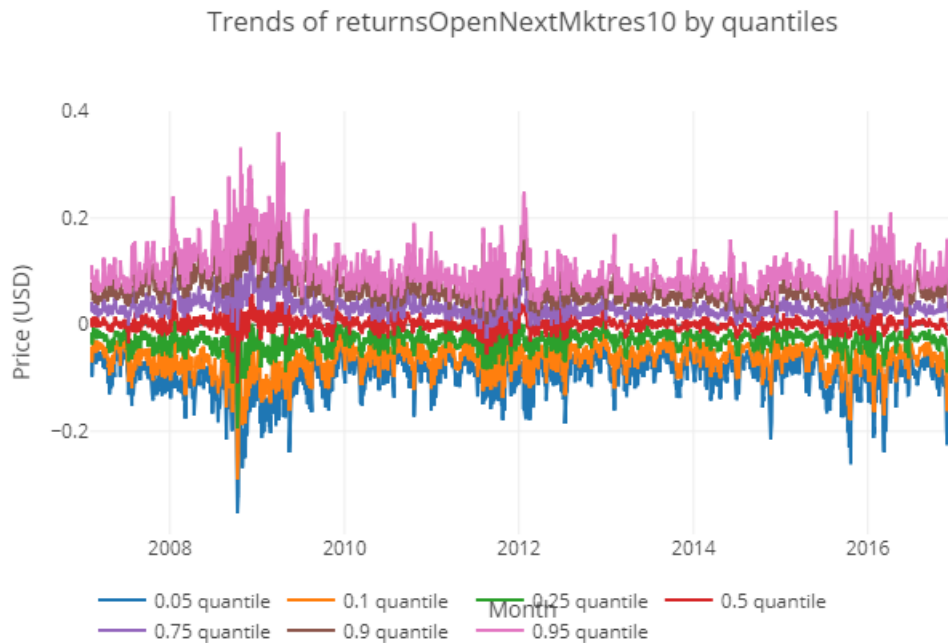
Now let's take a look at our target variable.

```

data = []
for i in [0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95]:
    price_df = market_train_df.groupby('time')['returnsOpenNextMktres10'].quantile(i).reset_index()

    data.append(go.Scatter(
        x = price_df['time'].dt.strftime(date_format='%Y-%m-%d').values,
        y = price_df['returnsOpenNextMktres10'].values,
        name = f'{i} quantile'
    ))
layout = go.Layout(dict(title = "Trends of returnsOpenNextMktres10 by quantiles",
                        xaxis = dict(title = 'Month'),
                        yaxis = dict(title = 'Price (USD)'),
                        legend=dict(
                            orientation="h"),
))
py.iplot(dict(data=data, layout=layout), filename='basic-line')

```



News Data

Also, let's take a look at the overview of News data:

```
print(f'{news_train_df.shape[0]} samples and {news_train_df.shape[1]} features in the training news dataset.')
```

9328750 samples and 35 features in the training news dataset.

Let's see a wordcloud of the last 100000 headlines.

```
text = ' '.join(news_train_df['headline'].str.lower().values[-1000000:])
wordcloud = WordCloud(max_font_size=None, stopwords=stop, background_color='white',
                      width=1200, height=1000).generate(text)
plt.figure(figsize=(12, 8))
plt.imshow(wordcloud)
plt.title('Top words in headline')
plt.axis("off")
plt.show()
```



Also, we want to see the summary of sentiment analysis:

```
for i, j in zip([-1, 0, 1], ['negative', 'neutral', 'positive']):
    df_sentiment = news_train_df.loc[news_train_df['sentimentClass'] == i, 'assetName']
    print(f'Top mentioned companies for {j} sentiment are:')
    print(df_sentiment.value_counts().head(5))
    print('')
```

```

Top mentioned companies for negative sentiment are:
Apple Inc                22518
JPMorgan Chase & Co      20647
BP PLC                   19328
Goldman Sachs Group Inc  17955
Bank of America Corp     17704
Name: assetName, dtype: int64

Top mentioned companies for neutral sentiment are:
HSBC Holdings PLC        19462
Credit Suisse AG       14632
Deutsche Bank AG       12959
Barclays PLC           12414
Apple Inc              10994
Name: assetName, dtype: int64

Top mentioned companies for positive sentiment are:
Apple Inc              19020
Barclays PLC           18051
Royal Dutch Shell PLC  15484
General Electric Co    14163
Boeing Co              14080
Name: assetName, dtype: int64

```

Light GBM

We built up the LightGBM model for gradient boosting that uses tree-based learning algorithm.

Light GBM grows tree vertically which is leaf-wise while other algorithms grow trees horizontally. It will choose the leaf with max delta loss to grow. When growing the same leaf, Leaf-wise algorithm can reduce more loss than a level-wise algorithm.

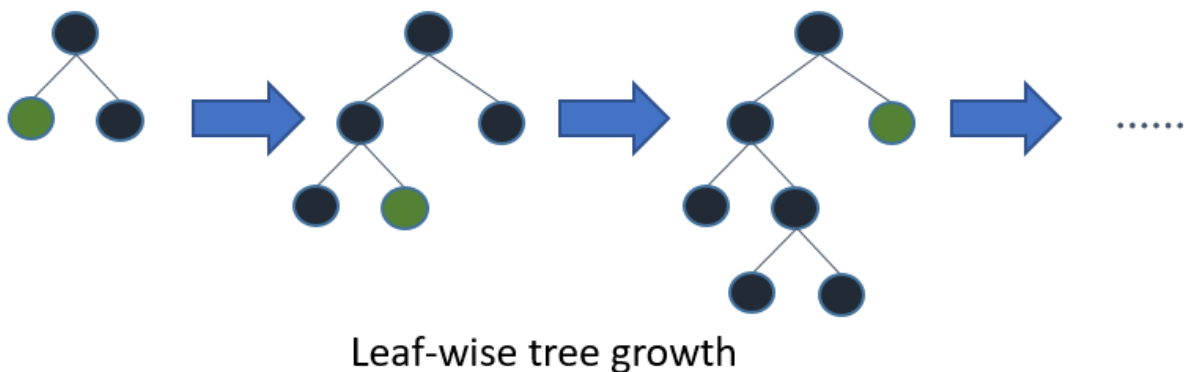


Figure 1: Light GBM

Light GBM is prefixed as 'Light' because of its high speed. Light GBM can handle the large size of data and takes lower memory to run. Another reason why we use lightGBM is that it

focuses on the accuracy of the results. It produces much more complex trees by following leaf wise split approach rather than a level-wise approach which is the main factor in achieving higher accuracy.

```
def run_lgb(train_X, train_y, val_X, val_y,args):
    params = {
        "objective" : "binary",
        "metric" : "binary_logloss",
        "num_leaves" : args['num_leaves'],
        "min_child_samples" : args['min_child_samples'],
        "learning_rate" : args['learning_rate'],
        "bagging_fraction" : 0.7,
        "feature_fraction" : 0.66,
        "bagging_frequency" : 5,
        "bagging_seed" : 2018,
        "verbosity" : -1
    }

    lgtrain = lgb.Dataset(train_X, label=train_y)
    lgval = lgb.Dataset(val_X, label=val_y)
    model = lgb.train(params, lgtrain, args['n_estimators'], valid_sets=[lgval], early_stopping_rounds=50,
        verbose_eval=100)

    # pred_test_y = model.predict(test_X, num_iteration=model.best_iteration)
    pred_val_y = model.predict(val_X, num_iteration=model.best_iteration)
    confidence_valid = model.predict(X_valid)*2 -1
    score = accuracy_score(confidence_valid > 0 , y_valid)
    print(score)
    mse = mean_squared_error(confidence_valid > 0, y_valid.astype(float))
    print("MSE", mse)
    print("args",args)
    return model, mse
```

Implementation Details

First: Load essential libraries

```
import gc
from datetime import datetime, timedelta
import numpy as np
import pandas as pd
from sklearn.metrics import log_loss
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
import seaborn as sns
import matplotlib.pyplot as plt
from kaggle.competitions import twosigmanews
pd.set_option('max_columns', 50)

env = twosigmanews.make_env()
market_train, news_train = env.get_training_data()
```

In this section, we first load all libraries including Pandas DataFrame, Xgboost framework and LightGBM framework for training our model. After comparing their efficiency and performance, we decided to use LightGBM instead of Xgboost, which will be discussed in detail in the model selection. Then we get two training datasets from Kaggle.

Second: Process Data and Build feature engineering

```
news_train['time'] = news_train['time'].dt.floor('d')
cols = ['sentimentNegative', 'sentimentNeutral', 'sentimentPositive', 'relevance', 'companyCount', 'bodySize',
        'sentenceCount', 'wordCount', 'firstMentionSentence',
        'sentimentWordCount', 'takeSequence', 'sentimentClass', 'noveltyCount12H',
        'noveltyCount24H', 'noveltyCount3D', 'noveltyCount5D', 'noveltyCount7D',
        'volumeCounts12H', 'volumeCounts24H', 'volumeCounts3D', 'volumeCounts5D', 'volumeCounts7D']

news_total = news_train[['time', 'assetName'] + cols].copy()
del news_train
gc.collect()
news_train = news_total
```

Here we use group all news published in the same day, then select usable features extracted from news, and align the features by time and assetNames.

```
def get_news_train(raw_data, days = 5):
    news_last = pd.DataFrame()
    rate = 1.0
    for i in range(days):
        cur_train = raw_data[cols] * rate
        rate *= 0.7
        cur_train['time'] = raw_data['time'] + datetime.timedelta(days = i, hours=22)
        cur_train['key'] = cur_train['time'].astype(str) + raw_data['assetName'].astype(str)
        cur_train = cur_train[['key'] + cols].groupby('key').sum()
        cur_train['key'] = cur_train.index.values
        news_last = pd.concat([news_last, cur_train[['key'] + cols]])
    del cur_train
    gc.collect()
    print("after concat the shape is:", news_last.shape)
    news_last = news_last.groupby('key').sum()
    news_last['key'] = news_last.index.values
    print("the result shape is:", news_last.shape)

    del news_last['key']
    return news_last

news_last = get_news_train(news_train)

market_train['key'] = market_train['time'].astype(str) + market_train['assetName'].astype(str)
market_train = market_train.join(news_last, on = 'key', how='left')

cat_cols = ['assetCode', 'assetName']
num_cols = ['volume', 'close', 'open', 'returnsClosePrevRaw1', 'returnsOpenPrevRaw1', 'returnsClosePrevMktres1',
            'returnsOpenPrevMktres1', 'returnsClosePrevRaw10', 'returnsOpenPrevRaw10', 'returnsClosePrevMktres10',
            'returnsOpenPrevMktres10', 'sentimentNegative', 'sentimentNeutral', 'sentimentPositive', 'relevance',
            'companyCount', 'bodySize', 'sentenceCount', 'wordCount', 'firstMentionSentence', 'sentimentWordCount',
            'takeSequence', 'sentimentClass', 'noveltyCount12H', 'noveltyCount24H', 'noveltyCount3D',
            'noveltyCount5D', 'noveltyCount7D', 'volumeCounts12H', 'volumeCounts24H', 'volumeCounts3D',
            'volumeCounts5D', 'volumeCounts7D']

def encode(encoder, x):
    len_encoder = len(encoder)
    try:
        id = encoder[x]
    except KeyError:
        id = len_encoder
    return id

encoders = [{i} for i in range(len(cat_cols))]
for i, cat in enumerate(cat_cols):
    print('encoding %s ...' % cat, end=' ')
```

```

encoders[i] = {l: id for id, l in enumerate(market_train.loc[train_indices, cat].unique())}
market_train[cat] = market_train[cat].astype(str).apply(lambda x: encode(encoders[i], x))
print('Done')

embed_sizes = [len(encoder) + 1 for encoder in encoders]

from sklearn.preprocessing import StandardScaler
import matplotlib

market_train['close'].clip(upper = 1000, inplace = True)
market_train['open'].clip(upper = 1000, inplace = True)
market_train['volume'].clip(upper = 1e+08, inplace = True)
print('scaling numerical columns')
scaler = StandardScaler()
col_mean = market_train[num_cols].mean()
market_train[num_cols] = market_train[num_cols].fillna(col_mean)
scaler = StandardScaler()
market_train[num_cols] = scaler.fit_transform(market_train[num_cols])

```

This is the most challenging part because we want to consider the impact of a piece of news on the stock price for a period of time. At first, we try to attach all news related to an asset with one week in every sample. However, the space complexity is too high, so we had to find an alternative. The best way we can think of is multiplying a decay factor(0.7 for one day) to all the features from one sample and add to the following dates(we find 5 days works best).

And we added the organized news data to the market data if the two samples in news dataset and market data set have the same assetName and time.

And we encode all categorical features into numeric by assigning a number to each category, for example, if one feature has 3 categories, then we will assign 0, 1 and 2 to each one. And we scale the price data and fill in the missing values by taking the overall average of this feature.

```

from sklearn.model_selection import train_test_split
train_indices, val_indices = train_test_split(market_train.index.values, test_size=0.25, random_state=23)

def get_input(market_train, indices):
    X = market_train.loc[indices, num_cols]
    for cat in cat_cols:
        X[cat] = market_train.loc[indices, cat].values
    y = (market_train.loc[indices, 'returnsOpenNextMktres10'] >= 0).values
    r = market_train.loc[indices, 'returnsOpenNextMktres10'].values
    u = market_train.loc[indices, 'universe']
    d = market_train.loc[indices, 'time'].dt.date
    return X, y, r, u, d

X_train, y_train, r_train, u_train, d_train = get_input(market_train, train_indices)

X_valid, y_valid, r_valid, u_valid, d_valid = get_input(market_train, val_indices)
print(X_valid.shape)

```

Finally we split the combined data set into training and test sets.

Third: Training Gradient Boosting Model

```

def train_predict(df_train, df_test, target = "winPlacePerc"):
    from xgboost import XGBClassifier
    import lightgbm as lgb
    from functools import partial
    from hyperopt import hp, fmin, tpe
    from sklearn.metrics import mean_squared_error
    algo = partial(tpe.suggest, n_startup_jobs=10)

def run_lgb(train_X, train_y, val_X, val_y, args):
    params = {
        "objective" : "binary",
        "metric" : "binary_logloss",
        "num_leaves" : args['num_leaves'],
        "min_child_samples" : args['min_child_samples'],
        "learning_rate" : args['learning_rate'],
        "bagging_fraction" : 0.7,
        "feature_fraction" : 0.66,
        "bagging_frequency" : 5,
        "bagging_seed" : 2018,
        "verbosity" : -1
    }

    lgtrain = lgb.Dataset(train_X, label=train_y)
    lgval = lgb.Dataset(val_X, label=val_y)
    model = lgb.train(params, lgtrain, args['n_estimators'], valid_sets=[lgval], early_stopping_rounds=50,
        verbose_eval=100)

    pred_val_y = model.predict(val_X, num_iteration=model.best_iteration)
    confidence_valid = model.predict(X_valid)*2 -1
    score = accuracy_score(confidence_valid > 0 , y_valid)
    print(score)
    mse = mean_squared_error(confidence_valid > 0, y_valid.astype(float))
    print("MSE", mse)
    print("args", args)
    return model, mse

def auto_tuning(args):
    model, mse = run_lgb(X_train, y_train.astype(int), X_valid, y_valid.astype(int), args)
    return mse

space = {"n_estimators":hp.choice('n_estimators', range(100,1000)),
        "num_leaves":hp.choice('num_leaves', range(20,100)),
        "min_child_samples":hp.choice("min_child_samples", range(20,2000)),
        'learning_rate':hp.loguniform('learning_rate', 0.01,0.3),
        'max_depth': hp.choice('max_depth', range(3,8))
    }
print(fmin)
best = fmin(auto_tuning, space, algo=algo, max_evals=100)
print(best)

args = {'learning_rate': 1.0958730495793214, 'max_depth': 7, 'min_child_samples': 301, 'n_estimators': 439,
        'num_leaves': 43}
model, _ = run_lgb(X_train, y_train.astype(int), X_valid, y_valid.astype(int), args)
gc.collect()

# calculation of actual metric that is used to calculate final score
confidence_valid = model.predict(X_valid)*2 -1
r_valid = r_valid.clip(-1,1) # get rid of outliers. Where do they come from??
x_t_i = confidence_valid * r_valid * u_valid
data = {'day' : d_valid, 'x_t_i' : x_t_i}
df = pd.DataFrame(data)
x_t = df.groupby('day').sum().values.flatten()
mean = np.mean(x_t)
std = np.std(x_t)
score_valid = mean / std
print(score_valid)
market_train.describe()

```


We use auto turning function of the LightGBM to find the best parameter for this model and training our final model based these parameters. Then check the score based on the test set.

Fourth: Define an MLP model and train it

```

days = env.get_prediction_days()

n_days = 0
predicted_confidences = np.array([])
from collections import deque
news_pre = deque()
news_all = pd.DataFrame()
BaseMod = 50
for (market_obs_df, news_obs_df, predictions_template_df) in days:
    n_days +=1
    print(n_days,end=' ')
    news_all = pd.concat([news_all,news_obs_df])
    if n_days >= BaseMod and n_days % BaseMod >= 0 and n_days % BaseMod < 8:
        news_pre.append(news_obs_df)
    elif n_days >= BaseMod and n_days % BaseMod == 8:
        del news_all
        gc.collect()
        news_all = pd.DataFrame()
        for item in news_pre:
            news_all = pd.concat([news_all,item])
        news_pre.clear()

news_last = get_news_train(news_all)

market_obs_df['key'] = market_obs_df['time'].astype(str) + market_obs_df['assetName'].astype(str)
market_obs_df = market_obs_df.join(news_last,on = 'key',how='left')

market_obs_df['close'].clip(upper = 1000, inplace = True)
market_obs_df['open'].clip(upper = 1000, inplace = True)
market_obs_df['volume'].clip(upper = 1e+08, inplace = True)

market_obs_df[num_cols]=market_obs_df[num_cols].fillna(col_mean)
market_obs_df[num_cols] = scaler.transform(market_obs_df[num_cols])
X_test = market_obs_df[num_cols]
X_test['assetCode'] = market_obs_df['assetCode'].apply(lambda x: encode(encoders[0], x)).values
X_test['assetName'] = market_obs_df['assetName'].apply(lambda x: encode(encoders[1], x)).values

market_prediction = model.predict(X_test)*2 -1
predicted_confidences = np.concatenate((predicted_confidences, market_prediction))

preds = pd.DataFrame({'assetCode':market_obs_df['assetCode'],'confidence':market_prediction})

predictions_template_df =
    predictions_template_df.merge(preds,how='left').drop('confidenceValue',axis=1).fillna(0).rename(columns={'confidence':'confidenceValue'})
env.predict(predictions_template_df)
del news_last
gc.collect()

env.write_submission_file()

```

In the end pull the days required by the competition, and make predictions with our model.

What worked?

In this Kaggle competition, since the database is enormous and it is impossible for people who use laptops to run our code, so we added two additional helper function to reduce the memory usage.

For function “reduce mem usage(df),” it is used to iterate through all the columns of a data frame and modify the data type to reduce memory usage.

```
def reduce_mem_usage(df):
    start_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))

    for col in df.columns:
        col_type = df[col].dtype
        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)

    end_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
    print('Decreased by {:.1f}%'.format(
        100 * (start_mem - end_mem) / start_mem))
    return df
```

What didn't work

model selection

Before the selection of LightGBM, some algorithms were also tried for this project. Firstly we used random forest which is also a tree-based algorithm generating trees based on different samples of data and use the result of different trees to vote for the final result. Usually, the performance of random forest is not bad, but perhaps because the data is so large with kind of high dimension of both continuous and categorical data, the AUC we got is only 0.75 which is much lower than the performance of XGBOOST and LightBGM. So we decided to do more parameter tuning and feature engineering work on another two algorithms.

As for the comparison between XGBOOST and LightGBM, they are similar and popular since both of them belong to the idea of boosting. Specifically, each round, they train the data with some kind of weak classifier (e.g., decision tree, logistic regression, etc.) and then

train the error part with the same classifier until it reaches a preset accuracy or it cannot reach a higher accuracy after many rounds training. However, the way they are generating trees is very different from what the following two charts shows.

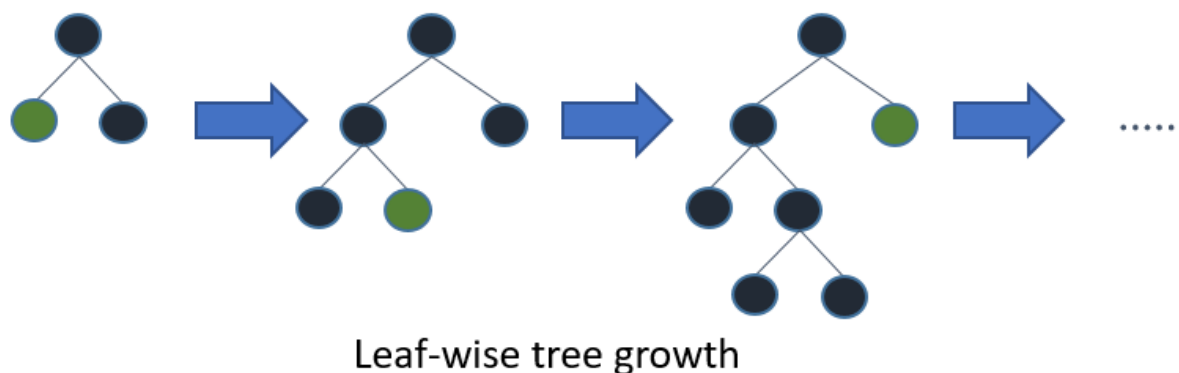


Figure 2: Light GBM

The way LightGBM generating trees is Leaf-wise which means it only split the node which bring the highest information gain on each level which is efficient in this kind of boosting method because firstly many nodes in the lower level of the tree will not contribute much information gain even you split them. Secondly, since this is boosting algorithm and error part will be relearned in the subsequent round of training. However, it also has a disadvantage which is also because of the way it is generating trees. it is easy to lead to an over-fitting problem since sometimes the algorithm may keep splitting on some branch and always go deeper and may ignore the information at other branches. So we need to use the max-depth parameter to control this problem.



Figure 3: XGBOOST

Regarding XGBOOST algorithm, the way it is generating trees is level-wise which means that each time it splits all the nodes in the same level, and this method is convenient for multi-threading optimization and could avoid the over-fitting problem on some extent compared with LightGBM. However, one issue of this is that it wastes CPU especially when it split all the nodes in a lower level of the tree because some nodes in the lower level could only contribute a little to the total accuracy.

	XGBoost
parameter used	max_depth: 50 learning_rate: 0.01 min_child_weight: 1 n_estimators: 200
training AUC	0.87
training Time	21642s

Figure 4: Result of XGBOOST

	LightGBM
parameter used	max_depth: 50 learning_rate: 0.01 num_leaves: 30 n_estimators: 200
training AUC	0.89
training Time	16755s

Figure 5: Result of Light BGM

So it could be seen that different algorithms all have their advantages and disadvantages. So we selected according to the accuracy and time efficiency. It could be seen that after using Grid-Search to try different parameter combination, here is the one which brings the best accuracy. The AUC got from XGBOOST is 0.87 which is a little lower than that of LightGBM which is 0.89. However, the time XGBOOST spend is much longer than that of LightGBM.

Feature Extraction


Another challenge we met is the feature extraction part for news data. Since a relatively long time range of stock price has to be learned if we want to learn the cycle or pattern of change of stock price. But the news data to match every day's price change is so large. Only the news from Reuters and belong to the period between 2006 and 2017 exceeds 9 million.

So it's hard to deal with that amount of data on ourselves' computer. So we decided to use the news data provided by 2 Sigma institution which is more convenient to do NLP analysis.

Conclusion

With our comprehensive feature engineering process and help from agglomerative bagging procedure, we successfully push our score from the initial commit of 0.57 to the final 0.6954 and ranked top10% out of 2317 teams. From this final project, Stock Price Prediction with News Data, we not only deeply understood and practiced the knowledge that Professor Verhagen taught us in class, such as building models and train models, dealing with tokens and analyzing using Bigrams, etc, but also learned how to use deep learning(neural network) and specifically use of LightGBM model to train our data. Besides, team collaboration also enables people to do great things that an individual couldn't.

Here see the figure below is part of our final submission and just for showing the format of the output.



	time	assetCode	confidenceValue
1	2017-01-03	A.N	0.37488194467172775
2	2017-01-03	AA.N	0.09431669088806305
3	2017-01-03	AAL.O	0.6470238220035371
4	2017-01-03	AAN.N	0.6624697640029094
5	2017-01-03	AAP.N	0.41331150521358406
6	2017-01-03	AAPL.O	0.12036844312941142
7	2017-01-03	ABB.N	0.44839587522928004
8	2017-01-03	ABBV.N	0.2912816598989687
9	2017-01-03	ABC.N	-0.1435037157184036
10	2017-01-03	ABCO.O	0.06009776635136399
11	2017-01-03	ABEV.N	0.26644055
12			

204	new	1有須的沸尔		0.69540	1	2u
205	▲769	XueFeng Si		0.69540	8	21h
206	new	rm -rf		0.69540	2	7m
Your Best Entry ↑ Your submission scored 0.69540, which is an improvement of your previous score of 0.59260. Great job! Tweet this!						
207	▼44	Dataggle		0.69540	2	1mo

Future Work

Due to the time limitation of the project, there's still many ideas haven't got a chance to deploy. If there's a chance in the future, we would firstly explore more sources of news data such as Financial Times and Yahoo Finance. Also, we want to explore some social media source since it could reflect users' real emotion and attitude to the brand or product of the company. Secondly, we'd like to figure out more ways to do feature engineering on news data to lead to a better result. finally, we want to try to separate different model for price data and natural language data to see could we get better accuracy in prediction

Contribution

Sibo Zhu: Developing LightGBM, Feature engineering, Researching related works, Extracting useful ideas from the wisdom of internet, Code Processing and Team management.


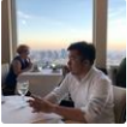
Yuchen Zhang: Helping with developing LightGBM and designing Feature Engineering, contributing to the final presentation and final report, researching on related works.

Daniel Zhang: Developed a descriptive analysis and worked on parameter tuning for different models.

Wenxiao Xiao: Preprocessing data, merge market and news datasets, designing Feature Engineering.

Code Running

Due to the fact that all the data are exclusive on Kaggle server and cannot be downloaded, agreed by Professor Verhagen, we can only submit all the codes we used, and the screenshot of our codes successfully ran on the Kaggle server. We strongly recommend whoever read this report to join the competition as well and test run our code in your private kernel. Pictures below are the proofs of our code passed the Kaggle server:

Sibozhu

LING131 Project(Team rm -rf)

last run 10 hours ago · IPython Notebook HTML
 using data from [Two Sigma: Using News to Predict Stock Movements](#) · Private [Make Public](#)

0 voters

[Notebook](#)
[Code](#)
[Data \(1\)](#)
[Output](#)
[Comments \(0\)](#)
[Log](#)
[Versions \(1\)](#)
[Forks](#)
[Options](#)
[Fork Notebook](#)
[Edit Notebook](#)

Run Info


Succeeded	True	Run Time	8316.4 seconds
Exit Code	0	Queue Time	0 seconds
Docker Image Name	kaggle/python(Dockerfile)	Output Size	0
Timeout Exceeded	False	Used All Space	False
Failure Message			

Log [Download Log](#)

```

Time   Line #  Log Message
4.0s   1      [NbConvertApp] Converting notebook script.ipynb to html
4.0s   2      [NbConvertApp] Executing notebook with kernel: python3
556.0s 3      [LightGBM] [Warning] Unknown parameter: bagging_frequency
8303.7s 4      [NbConvertApp] WARNING | Timeout waiting for IOPub output
8315.6s 5      [NbConvertApp] Support files will be in __results__files/
[NbConvertApp] Making directory __results__files
8315.7s 6      [NbConvertApp] Writing 1819280 bytes to __results__.html
8315.7s 7
8315.7s 8      Complete. Exited with code 0.


```



Featured Code Competition

Two Sigma: Using News to Predict Stock Movements

Use news analytics to predict stock price performance

 Two Sigma · 2,378 teams · 22 days to go (16 days to go until merger deadline)

\$100,000
Prize Money

[Overview](#)
[Data](#)
[Kernels](#)
[Discussion](#)
[Leaderboard](#)
[Rules](#)
[Team](#)
[My Submissions](#)
[Submit Predictions](#)

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
submission.csv	a few seconds ago	1 seconds	11 seconds	0.64535

Complete

[Jump to your position on the leaderboard](#)

References

<https://www.kaggle.com/artgor/eda-feature-engineering-and-everything>
<https://www.kaggle.com/mjbahmani/linear-algebra-for-data-scientists>
<https://www.kaggle.com/bguberfain/a-simple-model-using-the-market-and-news-data>
<https://www.kaggle.com/dster/two-sigma-news-official-getting-started-kernel>
<https://www.kaggle.com/bguberfain/a-simple-model-using-the-market-and-news-data>
<https://www.kaggle.com/qqgeogor/eda-script-67>