part3-final

January 3, 2019

0.1 # News Analytics and Stock Price Performance: Model Tuning and Selection (part 3)

Can we use news analytics and market data to predict stock price performance? There is no doubt that the ubiquity of data today enables investors at any scale to make better investment decisions but to truly harness this power, we must be able to distinguish signal from noise.

This is a 3 part walkthrough of a Kaggle competition by Two Sigma, with the end result being a model that predicts a signed confidence of an assets fluctuation over a ten-day window.

$$\widehat{y}_{ti} \in [-1, 1]$$

Initially, I was a little confused with the evaluation process here. Most people think of stock market predictions as being regression problems but this seemed like a binary classification problem to me. An asset either has a positive or a negative return, with the signed confidence being used to indicate both the direction and the magnitude of this move.

as mentioned above, the signed confidence interval needs to be between [-1 and 1]. Binary classification models are going to output a probability, naturally being a number between 0 and 1. So in order to get the output of my model to conform to this structure, I decided I would multiply my predicted value by 2 and then subtract 1. If the predicted probability is 0, this will cause the output to be -1 and if the predicted probability is 1, then the output will be 1.

For each day in the evaluation time period, we calculate:

$$x_t = \sum_i \widehat{y}_{ti} r_{ti} u_{ti}$$

$$score = \frac{\bar{x}_t}{\sigma(x_t)}$$

where r_{ti} is the 10-day market-adjusted leading return for day t for instrument i, and u_{ti} is a 0/1 universe variable that controls whether a particular asset is included in scoring on a particular day.

Your submission score is then calculated as the mean divided by the standard deviation of your daily x_t values:

If the standard deviation of predictions is 0, the score is defined as 0.

Two sources of data for this competition:

Market data (2007 to present) provided by Intrinio - contains financial market information such as opening price, closing price, trading volume, calculated returns, etc.

News data (2007 to present) Source: Thomson Reuters - contains information about news articles/alerts published about assets, such as article details, sentiment, and other commentary.

There are 3 notebooks for this walkthrough. The first includes all of the EDA for both datasets. In the second, I walk through feature preprocessing, exploration and engineering. Finally, in the third notebook, I build, test and tune multiple machine learning models.

```
In [1]: import numpy as np
        import pandas as pd
        import os
        from kaggle.competitions import twosigmanews
        import matplotlib.pyplot as plt
        import seaborn as sns; sns.set()
        from sklearn.model_selection import train_test_split
        from sklearn import ensemble
        from sklearn.metrics import accuracy_score
        from xgboost import XGBClassifier
        from lightgbm import LGBMClassifier
        import category_encoders as ce
        from xgboost import plot_importance
        import warnings
        import gc
        import psutil
        warnings.filterwarnings('ignore')
        %matplotlib inline
```

```
In [2]: # Function to keep track of memory usage.  These DataFrames represent 10+ million rows
        def cpuStats():
            pid = os.getpid()
            py = psutil.Process(pid)
            memoryUse = py.memory_info()[0] / 2. ** 30
            return 'memory GB:' + str(np.round(memoryUse, 2))
```

```
In [3]: cpuStats()

Out[3]: 'memory GB:0.26'
```

```
In [4]: # Load training data from API
        env = twosigmanews.make_env()
        (market_train_df, news_train_df) = env.get_training_data()

Loading the data... This could take a minute.
Done!
```

```
In [5]: cpuStats()

Out[5]: 'memory GB:7.05'
```

```
In [6]: print(f'market_train_df: {market_train_df.shape}')
        market_train_df.head()
```

```
market_train_df: (4072956, 16)
```

```
Out[6]:                        time   ...   universe
        0 2007-02-01 22:00:00+00:00   ...        1.0
        1 2007-02-01 22:00:00+00:00   ...        0.0
        2 2007-02-01 22:00:00+00:00   ...        1.0
        3 2007-02-01 22:00:00+00:00   ...        1.0
        4 2007-02-01 22:00:00+00:00   ...        1.0

        [5 rows x 16 columns]
```

```
In [7]: print(f'news_train_df: {news_train_df.shape}')
        news_train_df.head()
```

```
news_train_df: (9328750, 35)
```

```
Out[7]:                        time      ...       volumeCounts7D
        0 2007-01-01 04:29:32+00:00      ...                    7
        1 2007-01-01 07:03:35+00:00      ...                    3
        2 2007-01-01 11:29:56+00:00      ...                   17
        3 2007-01-01 12:08:37+00:00      ...                   15
        4 2007-01-01 12:08:37+00:00      ...                    0

        [5 rows x 35 columns]
```

A simple sanity check inspection of the data.

0.2 # Helper Functions

```
In [60]: def label_cat(df, col):
             return df[col].astype('category').cat.as_ordered()

         def bin_encode(df, cols_to_bin):
             ce_bin = ce.BinaryEncoder(cols = cols_to_bin)
             return ce_bin.fit_transform(df)

         # Find all features with NaN's
         def get_null_features(df):
             return df.columns[df.isna().any()].tolist()

         # Returns either month, day or year of date -> expedites feature engineering
         def get_date_feature(df, col, date_type):
             if (date_type == 'year'):
                 return pd.to_datetime(df[col]).dt.year
             elif (date_type == 'month'):
                 return pd.to_datetime(df[col]).dt.month
             elif (date_type == 'quarter'):
```

```python
        return pd.to_datetime(df[col]).dt.quarter
    else:
        return pd.to_datetime(df[col]).dt.dayofweek

# Helper function to print accuracy
def print_accuracy(model, x, y, x_val, y_val):
    print("Training Accuracy Score: ", accuracy_score(model.predict(x), y))
    print("Validation Accuracy Score: ", accuracy_score(model.predict(x_val), y_val))

# Helper function to get feature importances
def get_feature_imp(model, x):
    return pd.DataFrame(model.feature_importances_,
                                   index = x.columns,
                                    columns=['importance']).sort_values('importance',

# Calculate the RSI
def RSI(series, period):
    delta = series.diff().dropna()
    u = delta * 0
    d = u.copy()
    u[delta > 0] = delta[delta > 0]
    d[delta < 0] = -delta[delta < 0]
    u[u.index[period-1]] = np.mean( u[:period] ) #first value is sum of avg gains
    u = u.drop(u.index[:(period-1)])
    d[d.index[period-1]] = np.mean( d[:period] ) #first value is sum of avg losses
    d = d.drop(d.index[:(period-1)])
    rs = u.ewm(com=period-1, adjust=False).mean() / d.ewm(com=period-1, adjust=False)
    return 100 - 100 / (1 + rs)

# 80%, 20%
def templоral_split(df, labels):
    X_train, X_val = np.split(df, [int(.8*len(df))])
    y_t, y_v = np.split(labels, [int(.8*len(labels))])

    y_train = y_t > 0
    y_val = y_v > 0

    return [X_train, y_train, X_val, y_val]

# Join 2 dataframes
def join_df(left, right, left_on, right_on=None, suffix='_y'):
    if right_on is None: right_on = left_on
    return left.merge(right, how='left', left_on=left_on, right_on=right_on,
                      suffixes=("", suffix))

# Random search for temporal data
def random_search(model, param_grid, X_train, y_train, X_val, y_val):
    best_acc = 0
```

```
    for i in range(10):
        params = {k: np.random.choice(v) for k, v in param_grid.items()}
        params['n_jobs'] = -1
        new_model = model(**params)
        new_model.fit(X_train, y_train)
        acc = accuracy_score(new_model.predict(X_val), y_val)

        if acc > best_acc or best_acc == 0:
            best_acc = acc
            best_params = params
    print(f'Best Score: {best_acc}')
    print(f'Best Paramas: {best_params}')
    return best_params, best_acc
```

0.3 Preparing the Data

Here I have compiled all of the data preprocessing steps into a few functions. I had to remove a few processes here. Imputing and normalizing the data caused issues with the test set provided by Kaggle. The test set is pulled in via a provided python generator and the results I was getting on the provided test set were bizarre after normalizing. I suspect this is because the python generator pulled in data one day at a time, so I was essentially normalizing the data using the value ranges for a given day, instead of all values for the entire dataset. I also removed the tf-idf feature that I experimented with in part 2. It didn't seem to add much to the model performance and it greatly increased my data preprocessing time. it was causing my test set process at the bottom of this notebook to take around 30 minutes.

```
In [9]: # Prep market data
        # input: market dataframe
        # output: preprocessed and feature engineered dataframe
        def prep_market(df):
            # We will be using the assetCode
            df.drop(['assetName'], axis=1, inplace=True)
            _ = gc.collect()

            df['time'] = df.time.dt.strftime("%Y%m%d").astype(int)

            # Create year, month, day features
            df['month'] = get_date_feature(df, 'time', 'month')
            df['day'] = get_date_feature(df, 'time', 'day')
            df['quarter'] = get_date_feature(df, 'time', 'quarter')

            # Some feature engineering -> moving averages
            for n in [14, 30, 50, 200]:
                # Create the moving averages
                df['close_ma' + str(n)] = df['close'].rolling(window=n).mean()

            # Create RSI -> only 14 was useful during feature exploration
            df['rsi14'] = RSI(df['close'], 14)
```

```
        # 10 day pct change in volume
        df['vol_pct_change'] = df['volume'].pct_change()
        df['vol_pct_10'] = df['vol_pct_change'].rolling(window=10).mean()

        # drop 200 rows -> this is because of the moving average calculations
        df.dropna(inplace=True)
        _ = gc.collect()

        # These were identified in part 2 -> returnsOpenNextMktres10 is the dependant var
        train_cols = ['returnsClosePrevRaw10',
                      'returnsClosePrevMktres10',
                      'close',
                      'rsi14',
                      'assetCode',
                      'month',
                      'close_ma14',
                      'close_ma200',
                      'close_ma30',
                      'volume',
                      'close_ma50',
                      'returnsClosePrevMktres1',
                      'returnsClosePrevRaw1',
                      'returnsOpenPrevRaw1',
                      'vol_pct_10',
                      'day',
                      'quarter',
                      'time']

        # This is for the final trainin set contidion
        if 'returnsOpenNextMktres10' in df.columns:
            train_cols = train_cols + ['returnsOpenNextMktres10']

        df = df[train_cols]

        return df
```

In [10]:
```
# Prep news data
# input: news dataframe
# output: preprocessed and feature engineered dataframe
def prep_news(df):

    drop_list = [
        'audiences', 'subjects', 'assetName',
        'firstCreated', 'sourceTimestamp',
    ]
    df.drop(drop_list, axis=1, inplace=True)
    _ = gc.collect()
```

```
        # convert the date
        df['time'] = df.time.dt.strftime("%Y%m%d").astype(int)

        # convert the assets codes to a usable format
        df['assetCode'] = df['assetCodes'].map(lambda x: list(eval(x))[0])

        # encode provider
        df['provider'] = label_cat(df, 'provider').cat.codes

        # determine the proportion of the news item discussing the asset
        df['coverage'] = df['sentimentWordCount'] / df['wordCount']

        # relative position of the first mention in the item
        df['position'] = df['firstMentionSentence'] / df['sentenceCount']

        # Drop some unnecessary news features
        droplist = ['takeSequence','headlineTag',
                    'assetCodes', 'headline', 'marketCommentary']

        df.drop(droplist, axis=1, inplace=True)
        _ = gc.collect()

        # combine multiple news reports for same assets on same day
        newsgp = df.groupby(['time','assetCode'], sort=False).aggregate(np.mean).reset_inc

        return newsgp
```

```
In [11]: # Function for all data processing
    # input: market and news dataframe
    # output: preprocessed, feature engineered, grouped and joined dataframe
    def prep_data(market_train_df, news_train_df):
        market_train = prep_market(market_train_df)
        news_train = prep_news(news_train_df)

        joined = join_df(market_train, news_train, ['time', 'assetCode'], ['time', 'asset(

        # many assets that will have many days without news data
        joined[news_train.columns[2:].values] = joined[news_train.columns[2:].values].fil
        joined.drop(['time'], axis=1, inplace=True)
        joined['assetCodeT'] = label_cat(joined, 'assetCode').cat.codes

        del market_train
        del news_train
        _ = gc.collect()

        return joined
```

```
In [12]: ts_df = prep_data(market_train_df, news_train_df)
```

```
In [13]: print(f'Shape: {ts_df.shape}')
         ts_df.head()
```

```
Shape: (3978176, 44)
```

```
Out[13]:    returnsClosePrevRaw10     ...      assetCodeT
         0               0.050947     ...               0
         1               0.049460     ...               1
         2               0.033058     ...               6
         3               0.005546     ...               7
         4               0.025527     ...              14

         [5 rows x 44 columns]
```

```
In [14]: # Save memory
         del market_train_df
         del news_train_df
         _ = gc.collect()
```

0.4 Training

As I have said before, since we are dealing with large DataFrames, I am going to start training my models on a small sample of the data. I think it's important to be able to iterate quickly in the early phases and this will enable me to experiment quickly. I'll start by taking the last (most recent) 500,000 records with an 80/20 split.

Let's discuss parameter tuning. Using GridSearchCV in a conventional way doesn't make sense here because we are dealing with temporal data. Cross-validation would destroy the temporal nature. What I could have done though is manually created my splits in sequence and then passed an array of tuples containing the *IDX* of these splits for the training data and test data to the *cv* parameter. I didn't realize GridSearchCV could be used this way. Documentation to the rescue!

The issue here is that I am facing serious memory limitations due to the size of the data. GridSearchCV for some reasons was causing my kernel to consistently crash. Instead, I opted for a manual random search for a provided grid of parameters.

```
In [15]: # These cols are needed for the final prediction process
         train_cols = [col for col in ts_df.columns if col not in ['assetCode', 'universe', 're
```

```
In [31]: X_train_sample, y_train_sample, X_val_sample, y_val_sample = temploral_split(ts_df.il
```

```
In [32]: print(f'X_train_sample: {X_train_sample.shape}')
         print(f'X_val_sample: {X_val_sample.shape}')
```

```
X_train_sample: (400000, 42)
X_val_sample: (100000, 42)
```

0.4.1 Random Forest

```
In [33]: rfc = ensemble.RandomForestClassifier(n_estimators=30, n_jobs=-1, random_state=23)
         %time rfc.fit(X_train_sample, y_train_sample)
```

```
CPU times: user 2min 7s, sys: 196 ms, total: 2min 7s
Wall time: 33.9 s
```

```
Out[33]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=None, max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=30, n_jobs=-1,
                     oob_score=False, random_state=23, verbose=0, warm_start=False)
```

```
In [34]: print_accuracy(rfc, X_train_sample, y_train_sample, X_val_sample, y_val_sample)
```

```
Training Accuracy Score:  0.9998975
Validation Accuracy Score:  0.51367
```

Seems to be overfitting quite severely.

0.4.2 Random Forest: Random Search

```
In [41]: rfc = ensemble.RandomForestClassifier
         rfc_params = {'max_features': ['auto', 'sqrt', 'log2'],
                       'min_samples_leaf': [1, 2, 4, 6, 8, 10],
                       'n_estimators': [30, 50, 100, 200]}

         best_params, best_acc = random_search(rfc, rfc_params, X_train_sample, y_train_sample
```

```
Best Score: 0.52831
Best Paramas: {'max_features': 'sqrt', 'min_samples_leaf': 8, 'n_estimators': 200, 'n_jobs': -
```

```
In [42]: rfc = ensemble.RandomForestClassifier(**best_params)
         %time rfc.fit(X_train_sample, y_train_sample)
         print_accuracy(rfc, X_train_sample, y_train_sample, X_val_sample, y_val_sample)
```

```
CPU times: user 12min 5s, sys: 684 ms, total: 12min 5s
Wall time: 3min 4s
Training Accuracy Score:  0.9760125
Validation Accuracy Score:  0.52855
```

Seems to be overfitting quite severely. In my previous notebook, my RF models were hoving around 52% as well. Let's try XGBoost and see if my results differ.

```
In [49]: feature_importances = get_feature_imp(rfc, X_train_sample)
         feature_importances[0:30]
```

```
Out[49]:                              importance
         returnsClosePrevMktres10      0.071520
         returnsClosePrevRaw10         0.070432
         close                         0.065506
         close_ma200                   0.065200
         close_ma14                    0.065175
         rsi14                         0.065100
         close_ma50                    0.063641
         close_ma30                    0.063400
         returnsClosePrevMktres1       0.060641
         assetCodeT                    0.060635
         volume                        0.060545
         vol_pct_10                    0.059114
         returnsOpenPrevRaw1           0.058454
         returnsClosePrevRaw1          0.058150
         sentimentNeutral              0.007116
         sentimentNegative             0.006948
         sentimentWordCount            0.006880
         sentimentPositive             0.006860
         position                      0.006268
         coverage                      0.006231
         wordCount                     0.006132
         bodySize                      0.005962
         volumeCounts7D                0.005897
         sentenceCount                 0.005676
         volumeCounts5D                0.005103
         volumeCounts3D                0.004490
         companyCount                  0.003761
         volumeCounts24H               0.003604
         provider                      0.003423
         relevance                     0.003410
```

Interesting to see 4 of the features I created are near the top of the feature importance list.

0.4.3 XGBoost

```
In [50]: xgb = XGBClassifier(objective='binary:logistic', n_jobs=4, seed=23)
         %time  xgb.fit(X_train_sample, y_train_sample)
```

```
CPU times: user 1min 49s, sys: 380 ms, total: 1min 49s
Wall time: 28 s
```

```
Out[50]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
                max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
```

```
                n_jobs=4, nthread=None, objective='binary:logistic', random_state=0,
                reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=23, silent=True,
                subsample=1)
```

```
In [51]: print_accuracy(xgb, X_train_sample, y_train_sample, X_val_sample, y_val_sample)
```

```
Training Accuracy Score:  0.55403
Validation Accuracy Score:  0.53775
```

Slightly better than the RF model and not overfitting as severely (max_depth is 3). Let's try tweaking the parameters a little.

0.4.4 XGBoost: Random Search

```
In [52]: gbm_params = {
             'min_child_weight': [1, 5, 10],
             'gamma': [0.5, 1, 1.5, 2, 5],
             'subsample': [0.6, 0.8, 1.0],
             'colsample_bytree': [0.6, 0.8, 1.0],
             'max_depth': [3, 4, 5]
         }

         best_gbm_params, best_gbm_acc = random_search(XGBClassifier, gbm_params, X_train_sampl
```

```
Best Score: 0.53845
Best Paramas: {'min_child_weight': 10, 'gamma': 5.0, 'subsample': 1.0, 'colsample_bytree': 0.8
```

```
In [55]: xgb = XGBClassifier(objective='binary:logistic', **best_gbm_params)
         %time  xgb.fit(X_train_sample, y_train_sample)
```

```
CPU times: user 2min 26s, sys: 580 ms, total: 2min 27s
Wall time: 37.6 s
```

```
Out[55]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                colsample_bytree=0.8, gamma=5.0, learning_rate=0.1,
                max_delta_step=0, max_depth=5, min_child_weight=10, missing=None,
                n_estimators=100, n_jobs=-1, nthread=None,
                objective='binary:logistic', random_state=0, reg_alpha=0,
                reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
                subsample=1.0)
```

```
In [56]: print_accuracy(xgb, X_train_sample, y_train_sample, X_val_sample, y_val_sample)
```

```
Training Accuracy Score:  0.570535
Validation Accuracy Score:  0.53845
```

A slight increase in validation accuracy.

0.4.5 LGBM

```
In [47]: lgb = LGBMClassifier(n_jobs=4)
         %time lgb.fit(X_train_sample, y_train_sample)
```

```
CPU times: user 16.9 s, sys: 312 ms, total: 17.2 s
Wall time: 4.89 s
```

```
Out[47]: LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
                 importance_type='split', learning_rate=0.1, max_depth=-1,
                 min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0,
                 n_estimators=100, n_jobs=4, num_leaves=31, objective=None,
                 random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
                 subsample=1.0, subsample_for_bin=200000, subsample_freq=0)
```

```
In [48]: print_accuracy(lgb, X_train_sample, y_train_sample, X_val_sample, y_val_sample)
```

```
Training Accuracy Score:  0.5839275
Validation Accuracy Score:  0.53517
```

The LGBM model trains much faster than XGBoost and we get very similar results.

0.4.6 LGBM: Random Search

```
In [57]: lgbm_grid = {
             'learning_rate': [0.15, 0.1, 0.05, 0.02, 0.01],
             'num_leaves': [i for i in range(12, 90, 6)],
             'n_estimators': [50, 200, 400, 600, 800],
             'min_child_samples': [i for i in range(10, 100, 10)],
             'colsample_bytree': [0.8, 0.9, 0.95, 1],
             'subsample': [0.8, 0.9, 0.95, 1],
             'reg_alpha': [0.1, 0.2, 0.4, 0.6, 0.8],
             'reg_lambda': [0.1, 0.2, 0.4, 0.6, 0.8],
         }

         best_lgbm_params, best_lgbm_acc = random_search(LGBMClassifier, lgbm_grid, X_train_sa
```

```
Best Score: 0.53583
Best Paramas: {'learning_rate': 0.02, 'num_leaves': 48, 'n_estimators': 50, 'min_child_samples
```

```
In [58]: lgb = LGBMClassifier(**best_lgbm_params)
         %time lgb.fit(X_train_sample, y_train_sample)
```

```
CPU times: user 14.5 s, sys: 320 ms, total: 14.8 s
Wall time: 4.24 s
```

```
Out[58]: LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_bytree=0.95,
                importance_type='split', learning_rate=0.02, max_depth=-1,
                min_child_samples=80, min_child_weight=0.001, min_split_gain=0.0,
                n_estimators=50, n_jobs=-1, num_leaves=48, objective=None,
                random_state=None, reg_alpha=0.6, reg_lambda=0.1, silent=True,
                subsample=0.9, subsample_for_bin=200000, subsample_freq=0)
```

```
In [59]: print_accuracy(lgb, X_train_sample, y_train_sample, X_val_sample, y_val_sample)
```

```
Training Accuracy Score:  0.5517225
Validation Accuracy Score:  0.53583
```

Overall, XGBoost and LGBM performed better than Random Forest. XGBoost and LGBM both had similar results, however, training and prediction time for LGBM was much faster. For instance, fitting an LGBM model after a random grid search was 9 times faster than for XGBoost. LGBM also overfit less. So while the XGBoost model was 0.002% higher in accuracy, I think the other benefits of the LGBM model outweigh this slight increase in accuracy.

```
In [61]: # Save memory
         del X_train_sample
         del y_train_sample
         del X_val_sample
         del y_val_sample
         _ = gc.collect()
```

0.5 Final

Time to train a model on the entire dataset.

```
In [62]: X_train, y_train, X_val, y_val = temploral_split(ts_df[train_cols], ts_df['returnsOpe

         del ts_df
         _ = gc.collect()
```

```
In [63]: lgb_final = LGBMClassifier(**best_lgbm_params)
         %time  lgb_final.fit(X_train, y_train)
```

```
CPU times: user 2min 3s, sys: 4.55 s, total: 2min 8s
Wall time: 37 s
```

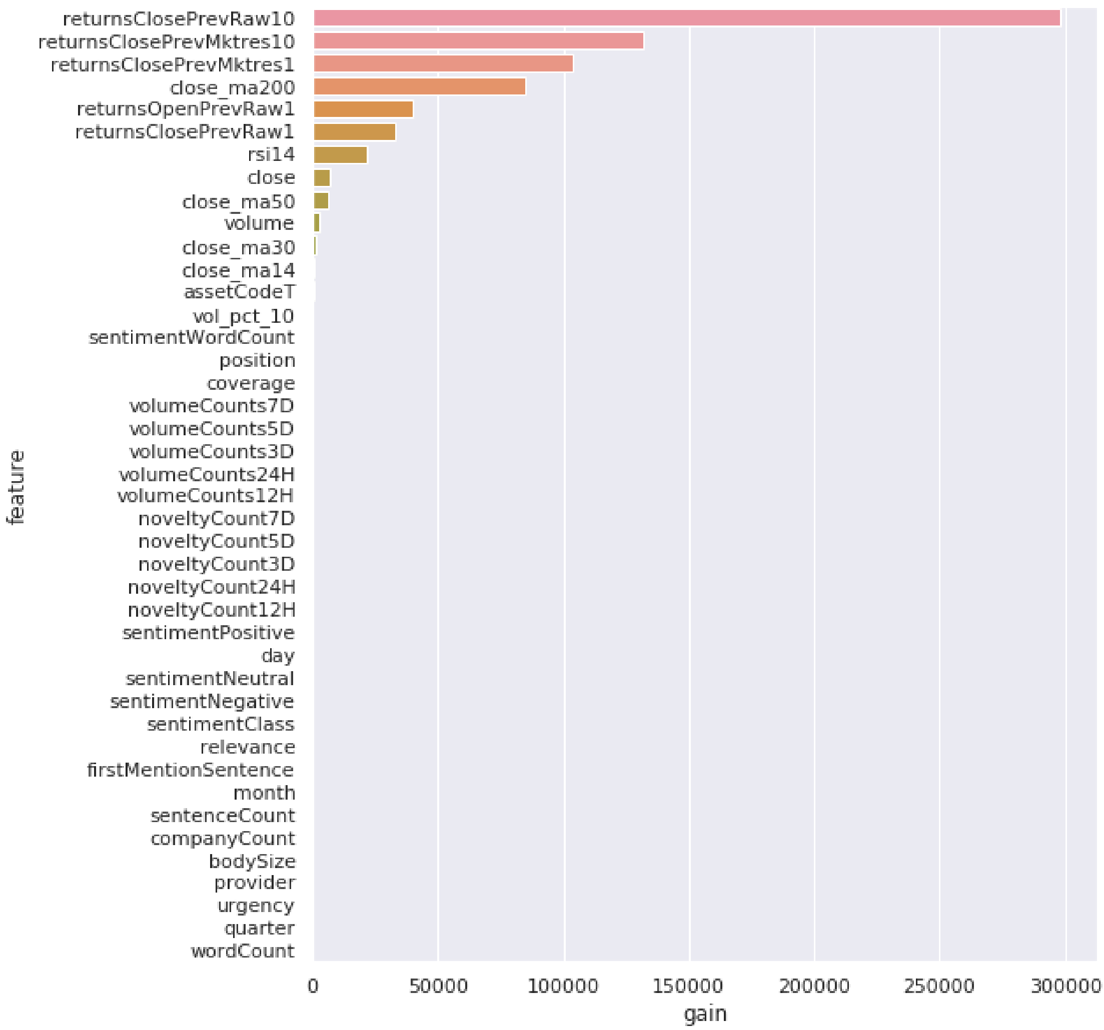
```
Out[63]: LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_bytree=0.95,
                importance_type='split', learning_rate=0.02, max_depth=-1,
                min_child_samples=80, min_child_weight=0.001, min_split_gain=0.0,
                n_estimators=50, n_jobs=-1, num_leaves=48, objective=None,
                random_state=None, reg_alpha=0.6, reg_lambda=0.1, silent=True,
                subsample=0.9, subsample_for_bin=200000, subsample_freq=0)
```

```
In [64]: print("Validation Accuracy Score: ", accuracy_score(lgb_final.predict(X_val), y_val))
```

```
Validation Accuracy Score:  0.5377081982212972
```

0.6 Feature Importances

```
In [65]: feat_importance = pd.DataFrame()
         feat_importance["feature"] = X_train.columns
         feat_importance["gain"] = lgb_final.booster_.feature_importance(importance_type='gain
         feat_importance.sort_values(by='gain', ascending=False, inplace=True)
         plt.figure(figsize=(8,10))
         ax = sns.barplot(y="feature", x="gain", data=feat_importance)
```




```
In [67]: del X_train
         del y_train
         del X_val
         del y_val
         _ = gc.collect()
```

1 Test set

```
In [68]: def write_submission(model, env):
             days = env.get_prediction_days()
             for (market_df, news_df, predictions_template_df) in days:

                 pd_df = prep_data(market_df, news_df)

                 pd_df = pd_df[pd_df.assetCode.isin(predictions_template_df.assetCode)]

                 # Drop cols that are not features
                 feats = [c for c in pd_df.columns if c not in ['assetCode', 'universe']]

                 # (* 2 - 1) used to cap predictions into required confidence intervals
                 preds = lgb_final.predict_proba(pd_df[feats])[:, 1] * 2 - 1
                 sub = pd.DataFrame({'assetCode': pd_df['assetCode'], 'confidence': preds})
                 predictions_template_df = predictions_template_df.merge(sub, how='left').drop
                     'confidenceValue', axis=1).fillna(0).rename(columns={'confidence':'confide

                 env.predict(predictions_template_df)
             env.write_submission_file()

         write_submission(lgb_final, env)
```

```
Your submission file has been saved. Once you `Commit` your Kernel and it finishes running, yo
```