

# model\_evaluation

December 11, 2019

[https://github.com/QuantCS109/TrumpTweets/blob/master/notebooks\\_modelling/model\\_evaluation.ipynb](https://github.com/QuantCS109/TrumpTweets/blob/master/notebooks_modelling/model_evaluation.ipynb)

```
[1]: import sys
sys.path.append('.') #to add top-level to path

import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import r2_score, accuracy_score, f1_score

from pandas.plotting import scatter_matrix
#import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

from modules.project_helper import VolFeatures, FuturesCloseData
```

## 0.1 Assumptions

Our set of features and predictors includes, for each of 13 trading assets, 55 predictors (59 for the commodities such as corn, wheat, and soybeans which include gamma features). Each has a set of 699 trading days, spanning from February 1st, 2017, to November 7th, 2019.

We split the data set into a 80% train and 20% test set, split without shuffling as we are dealing with a time series. A small number of days is removed from the beginning of the test set to prevent data leakage.

We want to predict 1 day market direction for each day in our sample. We predict 1 if the direction

is up, 0 if it's down. We use the performance measures described in the Objectives section.

The trading strategy buys 1 unit of risk of each asset whenever the prediction is 1, and sells short whenever the prediction is 0.

We assume we trade at the futures settlement price every day. In reality, it's impossible to trade at the settlement price, as it usually happens after market close hours. Different assets settle at different times of the day, and have different trading and closing hours. Given the nature of our dataset, we believe this assumption is the one that most closely resembles real trading conditions though.

## 0.2 Models

Logistic Regression (baseline): our initial model included all the predictors discussed in our features section Decision Tress: to preserve model interpretability Random Forest Gradient Boosting It is very easy to overfit a trading model. You will get perfect accuracy in the training set, and testing your strategy in the test will fail miserably. We try to focus on reducing variance and attempting not to overfit.

## 0.3 Best Model Metrics

- The test\_accuracy: overall model one day market direction predicting accuracy. To account for profit & loss and risks involved in any investment strategy we take the following metrics into consideration:
- The sharpe ratio: standardize measures of the excess from the risk-free rate mean return and divided by the standard deviation of returns. Positive sharpe = positive profit Negative sharpe = negative profit
- The f1\_score is the equally weighted harmonic mean of precision and recall. We look into the f1 score because as Lopez de Prado notes in his book, pg 206, "Accuracy may not be an adequate score for meta-labeling applications."
- $\text{precision} = \text{TP} / \text{TP} + \text{FP}$
- $\text{recall} = \text{TP} / \text{TP} + \text{FN}$
- The prediction ratio: number of times the strategy predicted 1 vs number of times it predicted 0.

```
[2]: import pickle
file = open("../data/features/full_features.pkl", 'rb')
full_features = pickle.load(file)
file.close()
```

```
[3]: instrument_list = ['ES', 'NQ', 'CD', 'EC', 'JY', 'MP', 'TY', 'US', 'C', 'S', 'W', 'CL', 'GC']
x_dict={}
y_dict={}
y_returns={}
for inst in instrument_list:
```

```

#y_dict[inst] = 2 * (full_features[inst][inst] >=0) - 1
y_dict[inst] = (full_features[inst][inst]>=0).astype(int)
x_dict[inst] = full_features[inst].drop([inst], axis=1)
returns = full_features[inst][inst]
y_returns[inst] = returns[[i in x_dict[inst].index for i in returns.index]]
if sum(y_returns[inst].index != x_dict[inst].index)!=0:
    raise Exception('Returns and X indices dont match')

```

```

[4]: class MLModel:
    def __init__(self,model,inst,x_dict,y_dict,y_returns,hyper_parameters={}):
        self.inst = inst
        self.x = x_dict[inst]
        self.y = y_dict[inst]
        self.y_returns = y_returns[inst]
        self.hyper_parameters=hyper_parameters

        self.model = model
        self.accuracy_train = None
        self.accuracy_test = None
        self.sharpe = None
        self.f1_test = None
        self.prediction_ratio = None

        self.strat_rets = None
        self.strat_rets_cum = None

        self.train_predictions = None
        self.test_predictions = None
        self.position = None

        self.train_class_balance = None
        self.test_class_balance = None

    def split_data(self):
        self.X_train, \
        self.X_test, \
        self.y_train, \
        self.y_test,\
        self.y_returns_train,\
        self.y_returns_test = train_test_split(self.x, self.y, self.y_returns,
        ↪test_size=0.20, shuffle=False)

    def train_model(self):
        #self.model = OLS(self.y_train, self.X_train)
        #self.model = self.model.fit()

```

```

        self.model = self.model(random_state=0,**self.hyper_parameters).
→fit(self.X_train,self.y_train)

    def evaluate_sharpe(self, cutoff=0.50):
        rets = self.strategy_returns(cutoff)[0]
        self.sharpe = np.sqrt(252)*np.mean(rets) / np.std(rets)

    def get_position(self, cutoff=0.50):
        # converting predictions from {0,1} to {-1,1}, short/long
        self.position = 2 * self.model.predict(self.X_test) - 1
        self.position[self.model.predict_proba(self.X_test).max(axis=1) <=
→cutoff] = 0
        return self.position

    def strategy_returns(self,cutoff=0.50):
        x = self.get_position(cutoff=cutoff)[: -1]
        y = self.y_returns_test[: -1] #make sure returns are logs
        self.strat_rets = x * y
        self.strat_rets_cum = self.strat_rets.cumsum()
        return self.strat_rets, self.strat_rets_cum

    def evaluate_model(self):
        self.accuracy_train = self.model.score(self.X_train, self.y_train)
        self.accuracy_test = self.model.score(self.X_test, self.y_test)
        self.f1_test = f1_score(self.y_test,self.test_predictions)
        self.evaluate_sharpe()
        self.prediction_ratio = np.mean(self.test_predictions)

    def generate_predictions(self):
        self.train_predictions = self.model.predict(self.X_train)
        self.test_predictions = self.model.predict(self.X_test)

        self.train_class_balance = np.mean(self.train_predictions)
        self.test_class_balance = np.mean(self.test_predictions)

class AssetModels:
    def __init__(self,inst,x_dict,y_dict,y_returns,hyper_parameters):
        #self.model = None
        self.logistic_model =
→MLModel(LogisticRegression,inst,x_dict,y_dict,y_returns,

```

```

                                hyper_parameters.get('logistic') if
↪hyper_parameters.get('logistic') else {}
        self.rf_model =
↪MLModel(RandomForestClassifier,inst,x_dict,y_dict,y_returns,
                                hyper_parameters.get('rf') if
↪hyper_parameters.get('rf') else {})
        self.tree_model =
↪MLModel(DecisionTreeClassifier,inst,x_dict,y_dict,y_returns,
                                hyper_parameters.get('tree') if
↪hyper_parameters.get('tree') else {})
        self.boosted_tree_model =
↪MLModel(GradientBoostingClassifier,inst,x_dict,y_dict,y_returns,
                                hyper_parameters.get('boosted_tree') if
↪hyper_parameters.get('boosted_tree') else {})

        self.ml_models = {'logistic':self.logistic_model,
                            'rf':self.rf_model,
                            #'tree':self.tree_model,
                            'boosted_tree':self.boosted_tree_model
                            }

        self.best_model_name = None
        self.best_model = None

        self.best_model_accuracy = None
        self.best_model_sharpe = None

        self accuracies_train = None
        self accuracies_test = None
        self.sharpe_values = None
        self.f1_scores = None
        self.prediction_ratios = None

    def get_best_model(self):
        self accuracies_test = pd.DataFrame.from_dict(
            {model_name:model.accuracy_test for model_name,model in self.
↪ml_models.items()},
            orient='index',columns=['test_accuracy']
        )
        self accuracies_train = pd.DataFrame.from_dict(
            {model_name:model.accuracy_train for model_name,model in self.
↪ml_models.items()},
            orient='index',columns=['test_accuracy']
        )

        self.sharpe_values = pd.DataFrame.from_dict(

```

```

        {model_name:model.sharpe for model_name,model in self.ml_models.
→items()}},
        orient='index',columns=['sharpe']
    )

    self.f1_scores = pd.DataFrame.from_dict(
        {model_name:model.f1_test for model_name,model in self.ml_models.
→items()}},
        orient='index',columns=['f1_score']
    )

    self.prediction_ratios = pd.DataFrame.from_dict(
        {model_name:model.prediction_ratio for model_name,model in self.
→ml_models.items()}},
        orient='index',columns=['prediction_ratio']
    )


    self.best_model_name = self.sharpe_values.idxmax().tolist()[0]
    self.best_model = self.ml_models.get(self.best_model_name)

    self.best_model_accuracy = self accuracies_test[self.accuracies_test.
→index==self.best_model_name]
    self.best_model_accuracy.index.name = 'best_model'
    self.best_model_accuracy.reset_index(inplace=True)

    self.best_model_sharpe = self.sharpe_values[self.sharpe_values.
→index==self.best_model_name]
    self.best_model_f1 = self.f1_scores[self.f1_scores.index==self.
→best_model_name]
    self.best_model_prediction_ratio = self.prediction_ratios[self.
→prediction_ratios.index==self.best_model_name]

    #self.best_model_sharpe.index.name = 'best_model'
    #self.best_model_sharpe.reset_index(inplace=True)


def run(self):
    {model.split_data() for model in self.ml_models.values()}
    {model.train_model() for model in self.ml_models.values()}
    {model.generate_predictions() for model in self.ml_models.values()}
    {model.evaluate_model() for model in self.ml_models.values()}
    self.get_best_model()

```

```

class ModelBuilder:
    def
    ↪__init__(self,x_dict,y_dict,y_returns,instrument_list,hyper_parameters={}):
        self.x_dict = x_dict
        self.y_dict = y_dict
        self.hyper_parameters = hyper_parameters
        self.instrument_list = instrument_list
        self.asset_models = {inst: AssetModels(inst,x_dict,y_dict,y_returns,
                                                hyper_parameters.get(inst) if hyper_parameters.
    ↪get(inst) else {})\
                                for inst in instrument_list}

        self accuracies_best = pd.DataFrame()
        self accuracies_all = pd.DataFrame()

    def get_accuracies(self):
        for inst in instrument_list:
            accuracy_df = self.asset_models[inst].best_model_accuracy
            accuracy_df.index = [inst]
            sharpe_df = self.asset_models[inst].best_model_sharpe
            sharpe_df.index = [inst]
            f1_df = self.asset_models[inst].best_model_f1
            f1_df.index = [inst]
            prediction_ratio_df = self.asset_models[inst].
    ↪best_model_prediction_ratio
            prediction_ratio_df.index = [inst]

            accuracy_df = accuracy_df.join(sharpe_df).join(f1_df).
    ↪join(prediction_ratio_df)
            self accuracies_best = self accuracies_best.append(accuracy_df)

            all_accuracy_df = self.asset_models[inst].accuracies_test
            all_sharpe_df = self.asset_models[inst].sharpe_values
            all_f1_df = self.asset_models[inst].f1_scores
            all_prediction_ratio_df = self.asset_models[inst].prediction_ratios

            all_accuracy_df = all_accuracy_df.join(all_sharpe_df).
    ↪join(all_f1_df).join(all_prediction_ratio_df)
            all_accuracy_df.index.name = 'model'
            all_accuracy_df = all_accuracy_df.reset_index()
            all_accuracy_df['asset'] = inst
            self accuracies_all = self accuracies_all.append(all_accuracy_df)
            self accuracies_all = self accuracies_all.set_index('asset')

```

```

def run(self):
    {inst: model.run() for inst,model in self.asset_models.items()}
    self.get accuracies()

```

```

[5]: hp = {
    'ES':{
        'logistic':{
            'C':0.1
        },
        'rf':{
            'max_depth':4,
            'max_features':13,
        },
        'boosted_tree':{
            'max_depth':6,
            'max_features':16,
        },
    },
    'NQ':{
        'logistic':{
            'C':0.01
        },
        'rf':{
            'max_depth':4,
            'max_features':3,
        },
        'boosted_tree':{
            'max_depth':14,
            'max_features':5,
        },
    },
    'CD':{
        'logistic':{
            'C':0.01
        },
        'rf':{
            'max_depth':13,
            'max_features':6,
        },
        'boosted_tree':{

```



```

        'max_depth':8,
        'max_features':6,
    },
},
'EC':{
    'logistic':{
        'C':1
    },
    'rf':{
        'max_depth':12,
        'max_features':6,
    },
    'boosted_tree':{
        'max_depth':11,
        'max_features':3,
    },
},
},
'JY':{
    'logistic':{
        'C':0.001
    },
    'rf':{
        'max_depth':4,
        'max_features':22,
    },
    'boosted_tree':{
        'max_depth':13,
        'max_features':16,
    },
},
},
'MP':{
    'logistic':{
        'C':10000
    },
    'rf':{
        'max_depth':8,
        'max_features':19,
    },
    'boosted_tree':{
        'max_depth':11,
        'max_features':22,
    },
},
},
'TY':{
    'logistic':{
        'C':0.001
    },
    'rf':{
        'max_depth':4,
        'max_features':22,
    },
    'boosted_tree':{
        'max_depth':13,
        'max_features':16,
    },
},
},

```

```


    'rf':{
        'max_depth':7,
        'max_features':13,
    },
    'boosted_tree':{
        'max_depth':8,
        'max_features':16,
    },
},
'US':{
    'logistic':{
        'C':0.001
    },
    'rf':{
        'max_depth':5,
        'max_features':3,
    },
    'boosted_tree':{
        'max_depth':11,
        'max_features':16,
    },
},
'C':{
    'logistic':{
        'C':0.01
    },
    'rf':{
        'max_depth':8,
        'max_features':19,
    },
    'boosted_tree':{
        'max_depth':11,
        'max_features':5,
    },
},
'S':{
    'logistic':{
        'C':1
    },
    'rf':{
        'max_depth':13,
        'max_features':10,
    },
    'boosted_tree':{
        'max_depth':6,
        'max_features':10,
    },
},

```

```

},
'W':{
  'logistic':{
    'C':10,
  },
  'rf':{
    'max_depth':4,
    'max_features':3,
  },
  'boosted_tree':{
    'max_depth':12,
    'max_features':6,
  },
},
},
'CL':{
  'logistic':{
    'C':0.1
  },
  'rf':{
    'max_depth':4,
    'max_features':4,
  },
  'boosted_tree':{
    'max_depth':5,
    'max_features':22,
  },
},
},
'GC':{
  'logistic':{
    'C':10000,
  },
  'rf':{
    'max_depth':4,
    'max_features':3,
  },
  'boosted_tree':{
    'max_depth':11,
    'max_features':25,
  }
}
}
}

```

```
[6]: model_builder = 
      ↪ ModelBuildier(x_dict,y_dict,y_returns,instrument_list,hyper_parameters=hp)
      model_builder.run()
```

## 0.4 Best Model Metrics

```
[7]: model_builder accuracies_best
```

```
[7]:
```

	best_model	test_accuracy	sharpe	f1_score	prediction_ratio
ES	boosted_tree	0.571429	1.863502	0.673913	0.764286
NQ	rf	0.535714	0.616839	0.691943	0.964286
CD	rf	0.485714	1.107758	0.462687	0.428571
EC	logistic	0.492857	-0.057362	0.219780	0.171429
JY	boosted_tree	0.514286	0.801115	0.260870	0.157143
MP	boosted_tree	0.535714	0.513795	0.628571	0.721429
TY	logistic	0.507143	0.811234	0.591716	0.621429
US	boosted_tree	0.521429	0.602155	0.544218	0.464286
C	boosted_tree	0.485714	-0.327765	0.555556	0.650000
S	boosted_tree	0.514286	1.210550	0.595238	0.692857
W	logistic	0.514286	1.014732	0.381818	0.271429
CL	boosted_tree	0.471429	0.745956	0.455882	0.471429
GC	rf	0.535714	0.781256	0.619883	0.650000

## 1 All Model Metrics

```
[8]: model_builder accuracies_all
```

```
[8]:
```

	asset	model	test_accuracy	sharpe	f1_score	prediction_ratio
ES	logistic	0.557143	0.910835	0.710280	0.978571	
ES	rf	0.521429	0.333081	0.666667	0.885714	
ES	boosted_tree	0.571429	1.863502	0.673913	0.764286	
NQ	logistic	0.542857	0.443255	0.703704	1.000000	
NQ	rf	0.535714	0.616839	0.691943	0.964286	
NQ	boosted_tree	0.478571	-1.755228	0.621762	0.835714	
CD	logistic	0.478571	-0.093572	0.075949	0.035714	
CD	rf	0.485714	1.107758	0.462687	0.428571	
CD	boosted_tree	0.485714	0.465284	0.320755	0.228571	
EC	logistic	0.492857	-0.057362	0.219780	0.171429	
EC	rf	0.450000	-1.009133	0.306306	0.314286	
EC	boosted_tree	0.471429	-0.283435	0.288462	0.264286	
JY	logistic	0.492857	0.535686	0.360360	0.292857	
JY	rf	0.514286	0.032064	0.128205	0.057143	
JY	boosted_tree	0.514286	0.801115	0.260870	0.157143	
MP	logistic	0.385714	-2.400286	0.426667	0.542857	
MP	rf	0.478571	-2.138776	0.522876	0.564286	

MP	boosted_tree	0.535714	0.513795	0.628571	0.721429
TY	logistic	0.507143	0.811234	0.591716	0.621429
TY	rf	0.471429	-0.960823	0.543210	0.571429
TY	boosted_tree	0.507143	-1.478931	0.566038	0.550000
US	logistic	0.521429	0.241914	0.637838	0.735714
US	rf	0.478571	-0.635984	0.425197	0.321429
US	boosted_tree	0.521429	0.602155	0.544218	0.464286
C	logistic	0.492857	-0.851619	0.628272	0.857143
C	rf	0.471429	-1.733111	0.471429	0.492857
C	boosted_tree	0.485714	-0.327765	0.555556	0.650000
S	logistic	0.485714	-0.415592	0.604396	0.792857
S	rf	0.485714	-0.808832	0.526316	0.578571
S	boosted_tree	0.514286	1.210550	0.595238	0.692857
W	logistic	0.514286	1.014732	0.381818	0.271429
W	rf	0.485714	-1.060103	0.478261	0.471429
W	boosted_tree	0.478571	-1.591892	0.406504	0.364286
CL	logistic	0.500000	-0.682313	0.666667	1.000000
CL	rf	0.471429	-0.862049	0.622449	0.900000
CL	boosted_tree	0.471429	0.745956	0.455882	0.471429
GC	logistic	0.485714	-0.677300	0.234043	0.100000
GC	rf	0.535714	0.781256	0.619883	0.650000
GC	boosted_tree	0.485714	-0.272016	0.409836	0.300000

```
[9]: def feat_imp(inst):
    try: df = pd.DataFrame(model_builder.asset_models[inst].best_model.model.
        ↪feature_importances_,
        index = x_dict[inst].columns,
        columns=['feature_importance'] ).
    ↪sort_values(by='feature_importance', ascending=False).head(10)
    except:
        df = pd.DataFrame(model_builder.asset_models[inst].ml_models['rf'].
        ↪model.feature_importances_,
        index = x_dict[inst].columns,
        columns=['feature_importance'] ).
    ↪sort_values(by='feature_importance', ascending=False).head(10)
    return df
```

```
[10]: feat_imp('ES')
```

```
[10]: feature_importance
svd_2                0.054485
ES_2M_1M_atm_vol    0.051513
intra_diff_15_5     0.049428
negative_proportion_mean 0.046886
combined_score_mean  0.043317
ES_volume_chg        0.033259
combined_score_max    0.032976
```

intra_ret_15	0.032656
ES_min_tweet	0.031317
ES_1M_atm_vol	0.029315

```
[11]: feat_imp('NQ')
```

```
[11]:
```

	feature_importance
combined_score_min	0.091241
NQ_1M_Fly25	0.060682
intra_blend	0.056874
svd_1	0.055587
NQ_daily_tweet	0.051200
positive_proportion_max	0.047721
NQ_max_tweet	0.044579
positive_proportion_min	0.039651
intra_ret_1	0.035541
neutral_proportion_mean	0.033770

```
[12]: feat_imp('CD')
```

```
[12]:
```

	feature_importance
neutral_proportion_mean	0.058555
positive_proportion_mean	0.043355
CD_1M_Fly25	0.041829
CD_1M_atm_vol	0.037757
CD_2M_Fly25	0.034282
combined_score_max	0.034085
combined_score_mean	0.033373
CD_1M_RR25	0.031666
CD_daily_tweet	0.030896
negative_proportion_mean	0.029563

```
[13]: feat_imp('EC')
```

```
[13]:
```

	feature_importance
EC_2M_Fly25	0.038656
intra_ret_15	0.038455
positive_proportion_mean	0.037400
combined_score_max	0.035604
EC_2M_1M_atm_vol	0.034939
neutral_proportion_min	0.033756
EC_max_tweet	0.033604
positive_proportion_max	0.033325
EC_volume_chg	0.033193
EC_daily_tweet	0.032027

```
[14]: feat_imp('JY')
```

```
[14]: feature_importance
      intra_diff_15_5      0.044658
      JY_max_tweet      0.043817
      combined_score_max      0.043595
      neutral_proportion_mean      0.038497
      combined_score_mean      0.036835
      combined_score_min      0.036501
      neutral_proportion_min      0.036479
      JY_volume_chg      0.034836
      JY_2M_RR25      0.034813
      JY_1M_atm_vol      0.034300
```

```
[15]: feat_imp('MP')
```

```
[15]: feature_importance
      MP_volume_chg      0.057362
      MP_2M_1M_atm_vol      0.055473
      MP_1M_atm_vol      0.051847
      MP_1M_RR25      0.038343
      intra_ret_15      0.037930
      intra_blend      0.035972
      MP_2M_RR25      0.035197
      svd_2      0.033308
      MP_daily_tweet      0.032682
      MP_min_tweet      0.032210
```

```
[16]: feat_imp('TY')
```

```
[16]: feature_importance
      TY_volume_chg      0.074339
      TY_2M_1M_atm_vol      0.066684
      intra_ret_1      0.049537
      TY_max_tweet      0.048582
      TY_1M_atm_vol      0.044206
      neutral_proportion_mean      0.038549
      TY_min_tweet      0.033162
      combined_score_max      0.031735
      TY_1M_RR25      0.031057
      intra_ret_15      0.030877
```

```
[17]: feat_imp('US')
```

```
[17]: feature_importance
      US_2M_RR25      0.043350
      intra_diff_15_5      0.037405
      US_max_tweet      0.034962
      intra_ret_15      0.034850
```

US_2M_1M_atm_vol	0.034387
combined_score_max	0.033417
US_min_tweet	0.033117
US_volume_chg	0.032601
svd_2	0.031988
US_1M_atm_vol	0.031915

```
[18]: feat_imp('C')
```

```
[18]:
```

	feature_importance
C_min_tweet	0.039365
C_down_diff_5	0.038048
C_max_tweet	0.034869
C_1M_Fly25	0.034855
C_2M_Fly25	0.034220
C_1M_RR25	0.033160
neutral_proportion_mean	0.031977
C_volume_chg	0.030803
intra_diff_15_5	0.030464
C_1M_atm_vol	0.028692

```
[19]: feat_imp('C')
```

```
[19]:
```

	feature_importance
C_min_tweet	0.039365
C_down_diff_5	0.038048
C_max_tweet	0.034869
C_1M_Fly25	0.034855
C_2M_Fly25	0.034220
C_1M_RR25	0.033160
neutral_proportion_mean	0.031977
C_volume_chg	0.030803
intra_diff_15_5	0.030464
C_1M_atm_vol	0.028692

```
[20]: feat_imp('S')
```

```
[20]:
```

	feature_importance
S_1M_Fly25	0.046787
svd_1	0.041096
S_2M_Fly25	0.040064
combined_score_min	0.038636
S_2M_1M_atm_vol	0.035160
S_volume_chg	0.034285
neutral_proportion_mean	0.033537
intra_blend	0.029619
S_2M_RR25	0.029456



intra_diff_15_5	0.028888
-----------------	----------

```
[21]: feat_imp('W')
```

```
[21]:
```

	feature_importance
intra_diff_15_5	0.078262
W_down_diff_5	0.064605
W_volume_chg	0.055811
W_max_tweet	0.054567
combined_score_mean	0.043809
W_1M_Fly25	0.041629
W_2M_1M_atm_vol	0.040373
svd_2	0.039497
W_up_diff_5	0.039379
neutral_proportion_max	0.039229

```
[22]: feat_imp('GC')
```

```
[22]:
```

	feature_importance
intra_ret_1	0.074310
combined_score_min	0.063572
svd_1	0.054905
intra_blend	0.052400
svd_2	0.043386
GC_2M_Fly25	0.041977
GC_1M_RR25	0.040624
GC_daily_tweet	0.040218
topic_8	0.037597
GC_2M_1M_atm_vol	0.036825