Stock Prediction for A-Shares#

1引言

1.1编写目的

实现一个基于Windows系统对于A股的股票预测软件。用户通过输入指定的股票代码，即可得到对于该股票的收益预测。

1.2项目背景

量化分析是金融和人工智能的交叉领域，有着广阔的前景。专业的量化分析人员能够利用金融知识和计算机技术分析行情，从而协助他们做出更有利的判断。

2项目介绍

2.1项目目标

项目目标是实现一个对于中国A股的股票预测软件，为用户提供必要的界面，用户选取股票后给出未来一周的涨幅预测。

2.2项目适用用户

该项目适用于寻求股票参考信息的用户，用户可以通过软件给出的预测适当调整投资策略。

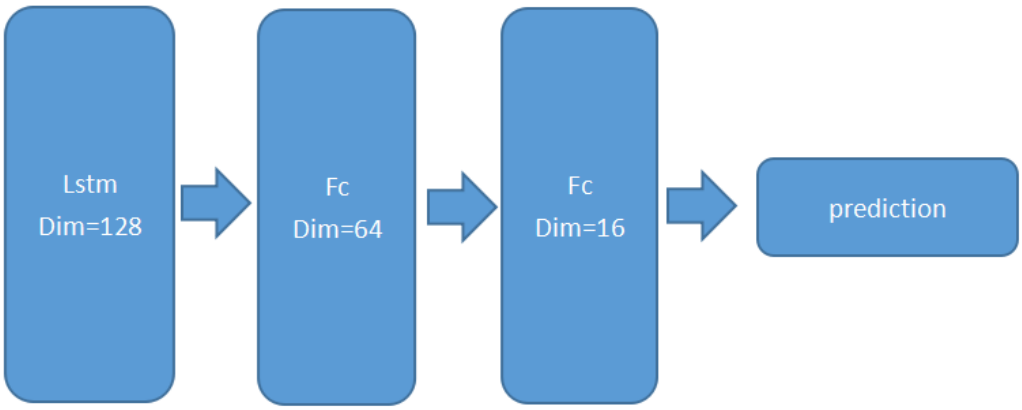
3模块功能

3.1数据获取模块

用户给出指定股票的代码后软件需要得到相应数据，如任意天的开盘价、收盘价、最高价、最低价、成交量，以及基本面信息等。

3.2#股票预测模块

依据获取的数据建立RNN模型，并给出最终的预测。



3.3图形界面

给出必要的用户界面，使得用户能够输入股票代码。

4数据处理

4.1数据输入

数据输入有任意天的开盘价、收盘价、最高价、最低价、成交量，以及一些基本面信息如盈利能力数据、营运能力数据、成长能力数据、偿债能力数据以及现金流量数据。由于以上基本面信息以季度的方式呈现，因此我们将季度的基本面信息扩展到每日。

4.2数据预处理

由于股票停盘等原因导致的数据缺失，需要填充缺失数据。首先对所有输入数据进行归一化，再对于缺失值采用线性插值的方式，依据停盘前后的数据进行插值。

4.3特征选择

由于可能有冗余特征的存在，因此在进行训练前先进行主成分分析，筛选出有用特征。

详细项目设计：

预测方法设计：

1.简介

LSTM Networks是递归神经网络（RNNs）的一种，该算法由Sepp Hochreiter和Jurgen Schmidhuber在Neural Computation上首次公布。后经过人们的不断改进，LSTM的内部结构逐渐变得完善起来。在处理和预测时间序列相关的数据时会比一般的RNNs表现的更好。目前，LSTM Networks已经被广泛应用在机器人控制、文本识别及预测、语音识别、蛋白质同源检测等领域。

2.流程

数据获取与处理：对于时间序列，我们通常会以[X(t-n),X(t-n+1),...,X(t-1),X(t)]这n个时刻的数据作为输入来预测(t+1)时刻的输出。对于股票来说，在t时刻会有若干个features，因此，为了丰富features以使模型更加精确，本文将n(time series)×s(features per time series)的二维向量作为输入。LSTM对于数据标准化的要求很高，因此本文所有input数据均经过z-score标准化处理。
LSTM模型构建：作为循环层的一种神经网络结构，只使用LSTM并不能构建出一个完整的模型，LSTM还需要与其他神经网络层（如Dense层、卷积层等）配合使用。此外，还可以构建多层LSTM层来增加模型的复杂性。
回测：本文进行的回测分为两种，一是直接将LSTM输出结果作为做单信号在个股上进行回测，二是将LSTM的预测结果作为一种择时信号，再配合其他选股模型（如BigQuant平台的StockRanker）进行回测。

3.初探

之前我们做过LSTM应用于股票市场的初步探究，使用方法为利用沪深300前100天的收盘价预测下一天的收盘价。从结果来看，LSTM对未来20天的预测基本上是对过去100天收盘价变化的趋势的总括，因此最终的预测结果以及回测结果都不是很理想。 之后尝试增加了features，效果依然不是很好。
通过对结果进行分析以及阅读研究一些研报，得到的初步结论为：
一是input时间跨度太长（100天的价格走势对未来一天的价格变化影响很小），而待预测数据时间跨度太短；
二是收盘价（Close）是非平稳数据，LSTM对于非平稳数据的预测效果没有平稳数据好。

4.沪深300未来五日收益预测

综合以上两点，本文所使用的输入和输出为利用过去30天的数据预测将来五天的收益。
测试对象：沪深300
数据选择和处理：input的时间跨度为30天，每天的features为['close','open','high','low','amount','volume']共6个，因此每个input为30×6的二维向量。 output为未来5日收益future_return_5（future_return_5>0.2,取0.2;future_return_5<-0.2,取-0.2），为使训练效果更加明显， output=future_return_5×10； features均经过标准化处理(在每个样本内每个feature标准化处理一次)。
训练数据：沪深300 2005-01-01至2014-12-31时间段的数据；
测试数据：沪深300 2015-01-01至2017-05-01时间段数据。
模型构建：鉴于数据较少（训练数据约2500个，预测数据约500个），因此模型构建的相对简单。模型共四层，为一层LSTM层+三层Dense层（图3）。 回测：得到LSTM预测结果后，若LSTM预测值小于0，则记为-1，若大于0，记为1。 每个模型做两次回测，第一次回测（后文简称回测1）为直接以LSTM预测值在沪深300上做单：若LSTM预测值为1，买入并持有5day（若之前已持仓，则更新持有天数），若LSTM预测值为-1，若为空仓期，则继续空仓，若已持有股票，则不更新持有天数； 第二次回测（后文简称回测2）为以LSTM为择时指标，与StockRanker结合在3000只股票做单：若LSTM预测值为1，则允许StockRanker根据其排序分数买入股票，若LSTM预测值为-1，若为空仓期，则继续空仓，若已持有股票，则禁止StockRanker买入股票，根据现有股票的买入时间，5天内清仓；

future_return_5是否二极化处理比较

对于future_return_5的处理分为两种情况，一种为直接将future_return_5作为output进行模型训练，二是将future_return_5二极化（future_return_5>0,取1；future_return_5<=0,取-1），然后将二极化后的数据作为output进行模型训练。
由于模型每次初始化权重不一样，每次预测和回测结果会有一些差别，但经过多次回测统计，直接将future_return_5作为output进行模型训练是一个更好的选择。在本文接下来的讨论中，将会直接将future_return_5作为output进行模型训练。

在权重上施加正则项探究

神经网络的过拟合：在训练神经网络过程中，“过拟合”是一项尽量要避免的事。神经网络“死记”训练数据。过拟合意味着模型在训练数据的表现会很好，但对于训练以外的预测则效果很差。原因通常为模型“死记”训练数据及其噪声，从而导致模型过于复杂。本文使用的沪深300的数据量不是太多，因此防止模型过拟合就尤为重要。 训练LSTM模型时，在参数层面上有两个十分重要的参数可以控制模型的过拟合：Dropout参数和在权重上施加正则项。Dropout是指在每次输入时随机丢弃一些features，从而提高模型的鲁棒性。它的出发点是通过不停去改变网络的结构，使神经网络记住的不是训练数据本身，而是能学出一些规律性的东西。正则项则是通过在计算损失函数时增加一项L2范数，使一些权重的值趋近于0，避免模型对每个feature强行适应与拟合，从而提高鲁棒性，也有因子选择的效果；(若希望在数学层面了解正则项更多知识，参考《机器学习中防止过拟合的处理方法》。在1)的模型训练中，我们加入了Dropout参数来避免过拟合。接下来我们尝试额外在权重上施加正则项来测试模型的表现。 回测结果如图6，加入正则项之后回测1和回测2的最大回撤均有下降，说明加入正则项后确实减轻了模型的过拟合。比较加入正则项前后回测1的持仓情况，可以看到加入正则化后空仓期更长,做单次数减少(19/17)，可以理解为：加入正则项之后，模型会变得更加保守。 正则项的问题：经过试验,对于一个LSTM模型来说，正则项的参数十分重要，调参也需要长时

间尝试，不合适的参数选择会造成模型的预测值偏正分布(大部分预测值大于0)或偏负分布，从而导致预测结果不准确，而较好的正则参数会使模型泛化性非常好(图6所用参数训练出来的模型的预测值属于轻度偏正分布)。本文之后的讨论仍会基于未加权重正则项的LSTM模型。

双输入模型探究

除了传统的Sequential Model(一输入，一输出)外，本文还尝试构建了Functional Model(支持多输入，多输出)。前面提到的features处理方法丢失了一项重要的信息：价格的高低。相同的input处在3000点和6000点时的future_return_5可能有很大不同。因此，本文尝试构建了"二输入一输出"的Functional Model:标准化后的features作为input输入LSTM层,LSTM层的输出结果和一个指标-label(label=np.round(close/500))作为input输入后面的Dense层，最终输出仍为future_return_5(图7)。 回测结果如图8。由回测结果可以看出，加入指示标后的LSTM模型收益率相对下降，但是回撤更小。LSTM预测值小于0的时间段覆盖了沪深300上大多数大幅下跌的时间段,虽然也错误地将一些震荡或上涨趋势划归为下跌趋势。或许这是不可避免的，俗话说高风险高回报，风险低那么回报也不会非常高，高回报和低风险往往不可兼得。

5.结论与展望

通过探究性地应用LSTM对沪深300未来五日收益率进行预测，初步说明了LSTM Networks是可以用在股票市场上的。 由于LSTM更适用于处理个股/指数，因此，将LSTM作为择时模型与其他选股模型配合使用效果较好。利用LSTM模型对沪深300数据进行预测并将结果作为择时信号，可以显著改善stockranker选股模型在回测阶段的回撤。

展望：由于个股数据量较少，LSTM模型的可扩展程度和复杂度受到很大制约，features的选择也受到限制（若input的features太多，而data较少的话，会使一部分features不能发挥出应有的作用，也极易造成过拟合）。将来我们希望能在个股/指数的小时或分钟数据上测试LSTM的性能。另外，将探究LSTM模型能否将属于一个行业的所有股票data一起处理也是一个可选的方向。

数据说明:

基础配置

```
start_date = '2010-01-01'
split_date = '2015-01-01'
end_date = '2018-01-01'
instrument = D.instruments(start_date=start_date, end_date=end_date, market='CN_STOCK_A')
```

获取每年年报公告后的第一个交易日历

```
trading_days = D.trading_days(market='CN', start_date=start_date, end_date=end_date)
trading_days['month'] = trading_days.date.map(lambda x:x.month)
trading_days['year'] = trading_days.date.map(lambda x:x.year)
groupby_td = trading_days.groupby(['year','month']).apply(lambda x:x.head(1))
first_date_after_financial_report = list(groupby_td[groupby_td['month']==5].date) # 5月第一个交易日
first_date_after_financial_report = [i.strftime('%Y-%m-%d') for i in first_date_after_financial_report] # date转换
为str
```

特征列表

```
financial_features_fields =
['date','fs_roe_0','fs_bps_0','fs_operating_revenue_ttm_0','fs_current_assets_0','fs_non_current_assets_0',

'fs_roa_0','fs_total_profit_0','fs_free_cash_flow_0','adjust_factor_0','fs_eps_0','pe_ttm_0','close_0',

'fs_common_equity_0','fs_net_income_0','market_cap_0','fs_eps_yoy_0','beta_szzs_90_0','fs_net_profit_margin_ttm_0',

            ]
```

按年获取财务特征数据

```
def get_financial_features(date,instrument=instrument,fields=financial_features_fields):
    assert type(date) == str
    df = D.features(instrument, date, date, fields)
    return df
```

获取财务特征数据，采取缓存的形式，可以节省运行时间

```
def get_financial_features_cache():
    print('获取财务特征数据，并缓存！')
    financial_features  = pd.DataFrame()
    for dt in first_date_after_financial_report:
        df = get_financial_features(dt)
        financial_features = financial_features.append(df)
    return Outputs(financial_features=DataSource.write_df(financial_features))
```

```
m1 = M.cached.v2(run=get_financial_features_cache)
financial_features_df = m1.financial_features.read_df()
```

获取日线特征数据

```
daily_history_features_fields = ['close','amount','pb_lf']  # 标注也在这里获取
def
get_daily_history_features(start_date=start_date,end_date=end_date,instrument=instrument,fields=daily_history_feature

    df = D.history_data(instrument,start_date,end_date,fields)
    return df
```

按股票groupby 计算日线特征

```
def calcu_daily_history_features(df):
    df['mean_amount'] = pd.rolling_apply(df['amount'], 22, np.nanmean)/df['amount']
    df['month_1_mom'] = df['close']/df['close'].shift(22)
    df['month_12_mom'] = df['close']/df['close'].shift(252)
    df['volatity'] = pd.rolling_apply(df['close'], 90, np.nanstd)/df['close']
    return df
```

获取日线特征，采取缓存的形式，可以节省运行时间

```
def get_daily_features_cache():
    print('获取日线特征数据，并缓存！')
    daily_history_features =
get_daily_history_features().groupby('instrument').apply(calcu_daily_history_features)
    return Outputs(daily_features=DataSource.write_df(daily_history_features))
m2 = M.cached.v2(run=get_daily_features_cache)
daily_features_df = m2.daily_features.read_df()
```

财务特征和日线特征合并

```
result=financial_features_df.merge(daily_features_df, on=['date', 'instrument'], how='inner')
```

抽取衍生特征

资产周转率

```
result['asset_turnover'] = result['fs_operating_revenue_ttm_0']/(result['fs_non_current_assets_0'] +
result['fs_current_assets_0'])
```

总盈利/总资产

```
result['gross_profit_to_asset'] = result['fs_total_profit_0']/(result['fs_non_current_assets_0'] +
result['fs_current_assets_0'])
```

自营现金流/总资产

```
result['cash_flow_to_assets'] = result['fs_free_cash_flow_0']/(result['fs_non_current_assets_0'] +
result['fs_current_assets_0'])
```

总收入/价格

```
result['sales_yield'] = result['fs_operating_revenue_ttm_0']/result['close_0']
```

现金流/股数/股价

```
result['cash_flow_yield'] =
result['fs_free_cash_flow_0']/(result['fs_common_equity_0']/result['close'])/result['close']
```

营业收入 Sales to EV

```
result['sales_to_ev'] = result['fs_operating_revenue_ttm_0']/result['fs_common_equity_0']
```

EBITDA to EV

```
result['ebitda_to_ev'] = result['fs_net_income_0']/result['fs_common_equity_0']

def judge_positive_earnings(df):
    if df['adjust_factor_0'] > df['adjust_factor_0_forward']:
        return 1
    else:
        return 0
```

构建时序衍生特征函数（# 一年前的pe # 一年前的总收入/价格 # 复权因子哑变量）

```
def construct_derivative_features(tmp):
    tmp['pe_forward'] = tmp['pe_ttm_0'].shift(1)
    tmp['sales_yield_forward'] = tmp['sales_yield'].shift(1)
    tmp['adjust_factor_0_forward'] = tmp['adjust_factor_0'].shift(1)
    tmp['positive_earnings'] = tmp.apply(judge_positive_earnings,axis=1)
    # 标注数据构建
    tmp['label'] = tmp['pb_lf'].shift(-1)
    return tmp

features_df = result.groupby('instrument').apply(construct_derivative_features)
```

去极值和标准化

哪些特征需要进行 去极值和标准化处理

```
need_deal_with_features = ['fs_free_cash_flow_0',  'fs_eps_0',
       'fs_operating_revenue_ttm_0', 'market_cap_0', 'fs_roe_0',
       'fs_current_assets_0', 'fs_roa_0', 'pe_ttm_0',
       'fs_non_current_assets_0', 'fs_eps_yoy_0', 'fs_bps_0', 'close_0',
       'adjust_factor_0', 'fs_common_equity_0', 'fs_net_income_0',
       'fs_total_profit_0', 'amount', 'pb_lf', 'mean_amount',
       'asset_turnover', 'gross_profit_to_asset', 'cash_flow_to_assets',
       'sales_yield', 'cash_flow_yield', 'sales_to_ev', 'ebitda_to_ev',
       'pe_forward', 'sales_yield_forward', 'adjust_factor_0_forward','label']
```

去极值

```
def remove_extremum(df,features=need_deal_with_features):
    factor_list = features
```

```
    for factor in factor_list:
        df[factor][df[factor] >= np.percentile(df[factor], 95)] = np.percentile(df[factor], 95)
        df[factor][df[factor] <= np.percentile(df[factor], 5)] = np.percentile(df[factor], 5)
    return df
```

标准化

```
def standardization(df,features=need_deal_with_features):
    factor_list = features
    for factor in factor_list:
        df[factor] = (df[factor] - df[factor].mean()) / df[factor].std()
    return df

def deal_with_features(df):
    return standardization(remove_extremum(df))

features_df_after_deal_with = features_df.groupby('date').apply(deal_with_features)
```

整理因子和标注

```
key_attr = ['instrument', 'date',]
explained_features = [ 'fs_eps_0','fs_net_profit_margin_ttm_0','fs_bps_0','asset_turnover','fs_roa_0','fs_roe_0',
       'gross_profit_to_asset','cash_flow_to_assets', 'positive_earnings', 'pe_forward','sales_yield',
'sales_yield_forward',
        'cash_flow_yield', 'sales_to_ev', 'ebitda_to_ev','market_cap_0',
       'beta_szzs_90_0', 'month_1_mom', 'month_12_mom', 'volatity','mean_amount', 'fs_eps_yoy_0']
label = ['label']
final_data = features_df_after_deal_with[label+key_attr+explained_features +['pb_lf']]

al:
import xgboost as xgb # 导入包
```

样本内的数据同样 划分训练数据和测试数据

```
assert len(final_data.columns) == 26
data = final_data[final_data['date'] <= '2015-01-01'] # 样本内数据
data = data[~pd.isnull(data[label[0]])] # 删除标注为缺失值的
data = data[key_attr+label+explained_features].dropna()  # 删除 特征为缺失值的
data.index = range(len(data))
train_data = data.ix[:int(len(data)*0.8)]  # 80%的数据拿来训练
test_data = data.ix[int(len(data)*0.8):]   # 剩下的数据拿来验证
```

数据按 特征和标注处理，便于xgboost构建对象

```
X_train = train_data[explained_features]
y_train = train_data[label[0]]
X_test = test_data[explained_features]
y_test = test_data[label[0]]
```

xgboost 构建对象

```
dtrain = xgb.DMatrix(X_train.values,label=y_train.values) # 这个地方如果是X_train 其实不影响结果
dtest = xgb.DMatrix(X_test.values,label=y_test.values)
```

设置参数，参数的格式用map的形式存储

```
param = {'max_depth': 3,                  # 树的最大深度
         'eta': 0.1,                        # 一个防止过拟合的参数，默认0.3
         'n_estimators':100,                 # Number of boosted trees to fit
         'silent': 1,                     # 打印信息的繁简指标，1表示简， 0表示繁
```

```
          'objective': 'reg:linear'}  # 使用的模型，分类的数目

num_round = 100 # 迭代的次数
```

看板，每次迭代都可以在控制台打印出训练集与测试集的损失

```
watchlist = [(dtest, 'eval'), (dtrain, 'train')]
```

训练模型

```
bst = xgb.train(param, dtrain, num_round, evals=watchlist)
```

测试集上模型预测

```
preds = bst.predict(dtest)
```

preds

样本外数据

```
out_of_sample_data = final_data[final_data['date'] > '2015-01-01'] # 样本内数据
out_of_sample_data = out_of_sample_data[key_attr+explained_features+['pb_lf']]  #  取出 特征数据
assert len(out_of_sample_data.columns) == 25
```

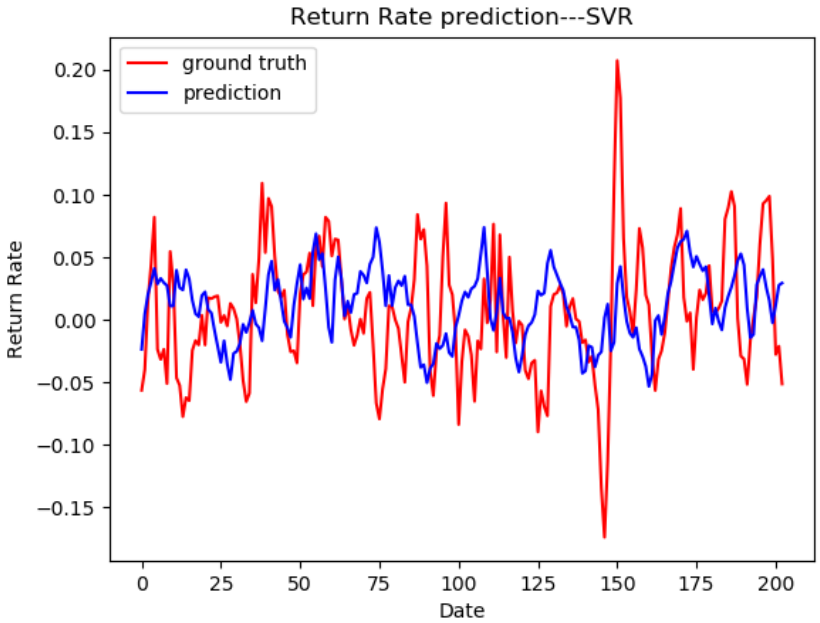
X_TEST = out_of_sample_data[explained_features] out_of_sample_dtset = xgb.DMatrix(X_TEST.values)

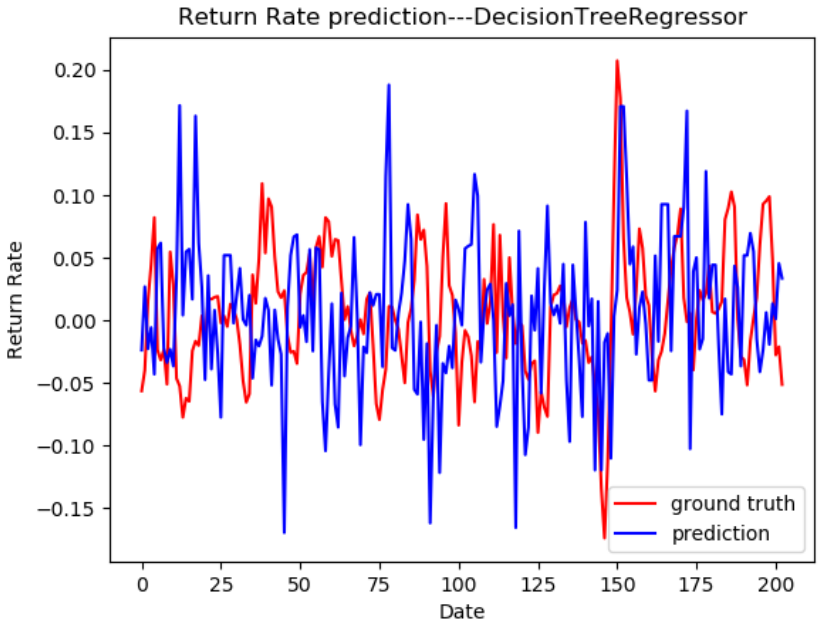
样本外预测

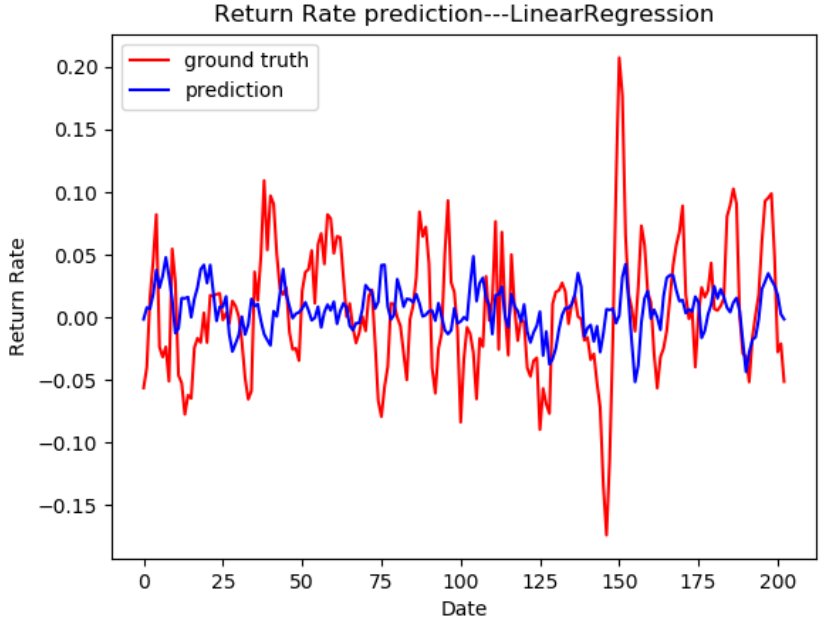
```
out_of_sample_preds = bst.predict(out_of_sample_dtset)
out_of_sample_data['predict_pb_lf'] = out_of_sample_preds
```

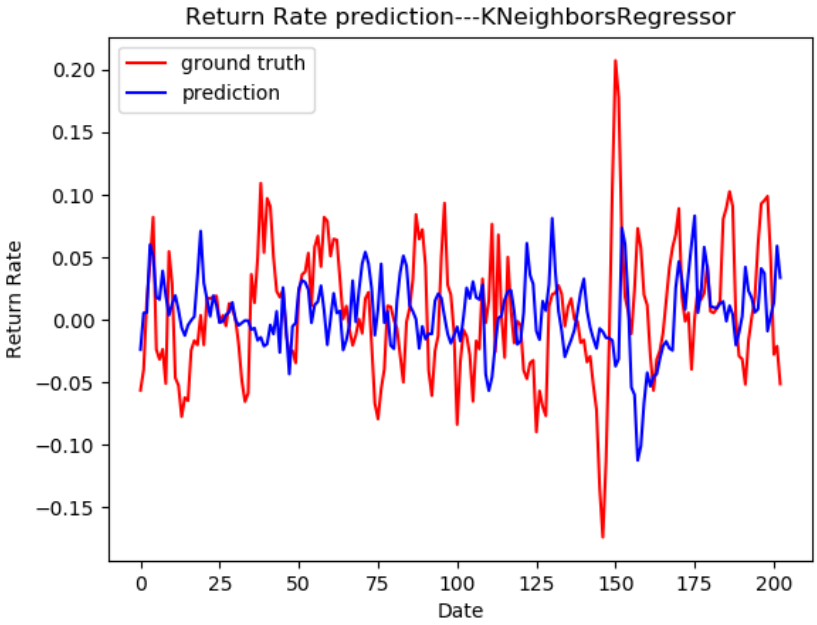
运行结果展示：

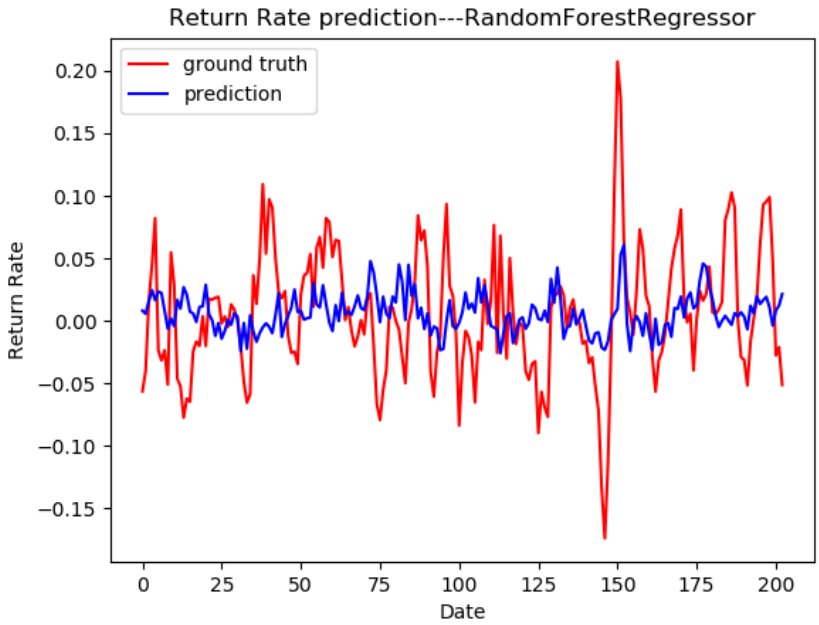
回报率预测：

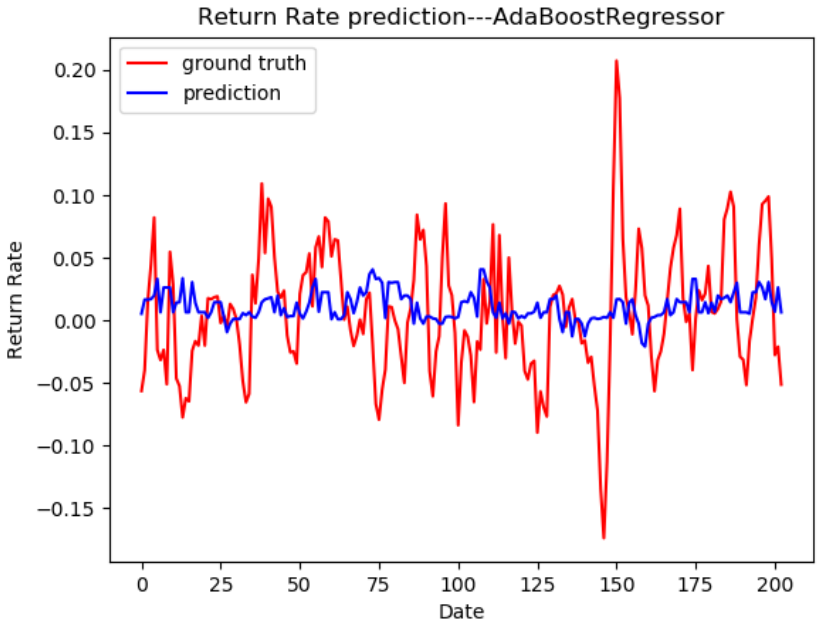



Return Rate prediction---DecisionTreeRegressor
Return Rate
Date
ground truth
prediction
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0
25
50
75
100
125
150
175
200


Return Rate prediction---LinearRegression
Return Rate
Date
ground truth
prediction
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0
25
50
75
100
125
150
175
200


Return Rate prediction---KNeighborsRegressor
ground truth
prediction
Return Rate
Date
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0
25
50
75
100
125
150
175
200


Return Rate prediction---RandomForestRegressor
ground truth
prediction
Return Rate
Date
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0
25
50
75
100
125
150
175
200


Return Rate prediction---AdaBoostRegressor
ground truth
prediction
Return Rate
Date
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0
25
50
75
100
125
150
175
200

由以上使用不同分类器进行预测的结果可以看出，本项目对股票回报率的预测达到了较好的水平。

具体运行结果如下表所示。

	EVS	MAE	MSE	R2
SVR	-0.1925581189325909	0.03581706707795301	0.0031410278791035176	-0.20414643673079214
DecisionTreeRegressor	-1.5196753053852112	0.058569527863652104	0.006572625001427888	-1.5196856825403162
LinearRegression	-0.027547608393728495	0.03470275992813882	0.0026805019394700157	-0.027598920893380008
KNeighborsRegressor	-0.3747273585328528	0.03961295237455111	0.0035866738942394935	-0.3749895752903962
RandomForestRegressor	0.002965054812178569	0.033336061179261134	0.0026008113407770175	0.002951317484161886
AdaBoostRegressor	-0.0117615935505615	0.03145987205999866	0.0026673922238331965	-0.022573172005364794

```
EVS of SVR: -0.1925581189325909
MAE of SVR: 0.03581706707795301
MSE of SVR: 0.0031410278791035176
R2 of SVR: -0.20414643673079214
EVS of DecisionTreeRegressor: -1.5196753053852112
MAE of DecisionTreeRegressor: 0.058569527863652104
MSE of DecisionTreeRegressor: 0.006572625001427888
R2 of DecisionTreeRegressor: -1.5196856825403162
EVS of LinearRegression: -0.027547608393728495
MAE of LinearRegression: 0.03470275992813882
MSE of LinearRegression: 0.0026805019394700157
R2 of LinearRegression: -0.027598920893380008
EVS of KNeighborsRegressor: -0.3747273585328528
MAE of KNeighborsRegressor: 0.03961295237455111
MSE of KNeighborsRegressor: 0.0035866738942394935
R2 of KNeighborsRegressor: -0.3749895752903962
EVS of RandomForestRegressor: 0.002965054812178569
MAE of RandomForestRegressor: 0.033336061179261134
MSE of RandomForestRegressor: 0.0026008113407770175
R2 of RandomForestRegressor: 0.002951317484161886
EVS of AdaBoostRegressor: -0.0117615935505615
MAE of AdaBoostRegressor: 0.03145987205999866
MSE of AdaBoostRegressor: 0.0026673922238331965
R2 of AdaBoostRegressor: -0.022573172005364794
```