Quantopian and Machine Learning

July 8, 2018

0.1 Quantopian and Machine Learning

Leo Liu 07/08/2018

0.1.1 Summary

In quantitative finance machine learning is used in various ways, which include prediction of future asset prices, optimizing trading strategy parameters, managing risk and detection of signals among noisy datasets.

In these exercises, simple machine learning models were used to estimate the next movement. To construct a strong classifier, I combined multiple models by simple vote. The ensemble learning is called bagging.

0.1.2 Ensemble classifier

I wrote a simple ensemble classifier **ML_classifiers**, which is lacked in current version of sklearn available in Quantopian. The **ML_classifiers** take list of classifiers as input, e.g. LogisticRegression, SVC. It fits on training data and predicts by simple vote.

```
In [3]: class ML_classifiers:
            """
                simple ensumble model
            """
            def __init__(self, models):
                """
                Args:
                    models: list of machine learning classifiers

                """
                self._models=models
            def fit(self,X,y):
                """
                fit traing data
                Args:
                    X: training features
                    y: labels
                """
                for mod in self._models:
```

```
            mod.fit(X,y)
        def predict(self,x):
            """
            predicts on new feature x from each classifiers
            Args:
                x: new feature
            """
            return [mod.predict(x)[0] for mod in self._models]
        def predict_vote(self,x):
            """
            predicts on new feature x by simple vote
            Args:
                x: new feature
            """
            cnts=Counter([mod.predict(x)[0] for mod in self._models])
            return cnts.most_common(1)[0][0]
```

0.1.3 Algorithm 1

- In this algorithm, I used previous changes to predict the next movement.
- The securities were chosen from symbols AAPL, BA, WMT, COST and MLA.
- The rolling window are ten days. The deque was used to store changes.
- A bagging of three classifiers were employed in the prediction, which are random forest, linear SVC and logistic regression.
- The predicted movement had three status: up (>0.5%), down(<-0.5%) and stay.
- The trading strategy longs stock if the stock is predicted up and short if down.

```
In [ ]: from sklearn.linear_model import LogisticRegression
        from sklearn.svm import  LinearSVC
        from sklearn.ensemble import RandomForestClassifier
        from sklearn import preprocessing
        from collections import deque
        from collections import Counter
        from quantopian.pipeline.data.builtin import USEquityPricing
        import numpy as np

        def initialize(context):

            # chosen securities
            context.securities = [symbol('AAPL'), symbol('BA'), symbol('WMT'),
                                  symbol('IBM'),symbol('COST'),symbol('MLA')]
            context.window_length = 10 # Amount of prior bars to study

            # bagging classifier of three
            context.classifiers=ML_classifiers([RandomForestClassifier(),
                                                LinearSVC(), LogisticRegression()])

            # deques are lists with a maximum length where old entries are shifted out
```

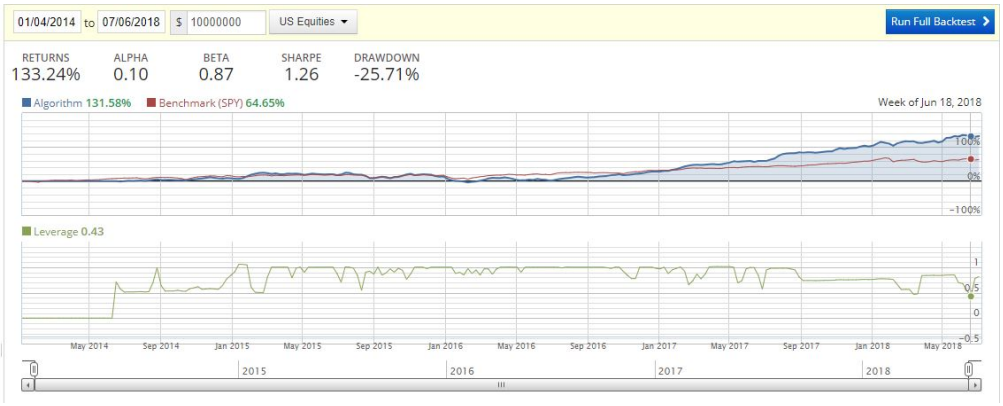
```
    # Stores recent prices with deque
    context.recent_prices = {stock: deque(maxlen=context.window_length+2)
                             for stock in context.securities}
    # Independent, or input variables
    context.X = { stock: deque(maxlen=500)  for stock in context.securities}
    # Dependent, or output variable
    context.Y = { stock: deque(maxlen=500)  for stock in context.securities}

    schedule_function(rebalance, date_rules.every_day(),
                      time_rules.market_close(minutes=5))
    schedule_function(record_vars, date_rules.every_day(),
                      time_rules.market_close())


def rebalance(context, data):
    for stock in context.securities:
        # Update the recent prices
        context.recent_prices[stock].append(data.current(stock, 'price'))
        # If there is enough recent price data
        if len(context.recent_prices[stock]) == context.window_length+2:

           # previous changes
            changes = np.diff(context.recent_prices[stock])/ \
                              context.recent_prices[stock][-1]
            context.X[stock].append(changes[:-1])
            # labels with three status [1,0.5,0] mapping to up,
            #stay and down, 0.05 is the threshold
            context.Y[stock].append(   1 if changes[-1]>0.005
                                  else 0.5 if  changes[-1]>=-0.005  else 0 )
            if len(context.Y[stock]) < 100 or np.isnan(context.X[stock]).any():
                continue
            # train model
            context.classifiers.fit(context.X[stock],context.Y[stock])
            # make prediction
            prediction=context.classifiers.predict_vote(changes[1:])
            # If prediction = 1, buy all shares affordable,if 0 sell all shares
            if prediction in [1,0]:
                # place order, multiplied by 0.5 to limit leverage
                order_target_percent(stock, prediction*0.5)


def record_vars(context, data):
    record(Leverage=context.account.leverage)
```



Algorithm 1 Result

0.1.4 Algorithm 2

- In this algorithm, I used simple moving average SMA and momentum indicators ADX, RSI to predict the next movement.
- The stocks to take is chosen by volume with percentile from 99.5 to 100.
- The algorithm predict if a stock will move up or not.
- A bagging of three classifiers were employed in the prediction, which are Adaboost and GaussianNB
- The strategy longs stock if the stock is predicted up and cashes out otherwise.

```
In [ ]: from talib import SMA, ADX, RSI
        import numpy as np
        from sklearn.naive_bayes import GaussianNB
        from sklearn.ensemble import AdaBoostClassifier as Adaboost

        def initialize(context):

            #Variables to change
            context.lookback = 1250          # number bars of data to use in  classifier
            context.omit = 251               # number of bars to omit. To get rid of nan's.
            context.timeframe = 20           #How often are we trading?  Monthly? 20.
            context.time_to_run = 15         #How many minutes before market close
            context.gain_cutoff = 0.01       #Threshold to long
            context.target_levervage = 1.0   #target leverage
            #bagging classifier of two
            context.ML=ML_classifiers( [Adaboost(),  GaussianNB()])
            #Stocks to trade
            set_universe(universe.DollarVolumeUniverse(floor_percentile=99.5,
                                                       ceiling_percentile=100))

            schedule_function(rebalance, date_rules.month_start(),
             time_rules.market_close(minutes=(context.time_to_run+2)), half_days=True)
```

```
def handle_data(context, data):
    record(leverage=context.account.leverage)

#Getting the data, formatting the features and labels.
def rebalance(context, data):

    #Pricing data
    prices = history(context.lookback, '1d', 'price')
    highs = history(context.lookback, '1d', 'high')
    lows = history(context.lookback, '1d', 'low')


    #Clear the lists so that some points are not repeated
    hold_list = []           # holding list of stocks
    feature_test={}          # features to predict, indexed by stock
    X,y=[],[]                # list holds features and label
    for stock in data:
        #Indicators simple moving average SMA and momentum indicators ADX, RSI
        feature_list=[]
        feature_list+=list(map(lambda  period : SMA(prices[stock],
                           timeperiod=period)/prices[stock],
                              [5,10,20,50,100,200,250]))

        feature_list+=list(map(lambda  period : ADX(highs[stock],
                           lows[stock],prices[stock],
                           timeperiod=period)/prices[stock],[7,14,25]))

        feature_list+=list(map(lambda  period : RSI(prices[stock],
                           timeperiod=period)/prices[stock], [7,14,25]))

        #Labels 1 means move up
        labels=(prices[stock].pct_change(
            periods=context.timeframe)>context.gain_cutoff)\
                        .astype('int32').shift(-context.timeframe)

        #Only go through the none nan values
        for i in range(context.omit, (context.lookback-context.timeframe)):
            #collect features by date
            features=[ item[i]  for item in feature_list]
             #Check any nan value in the features
            if not np.isnan(features).any():
                X.append(features)
                y.append(labels[i])


        # gather feature to predict
        tmp=[ item[-1]  for item in feature_list]
```

Algorithm 2 Result

```
    if not np.isnan(tmp).any():
        feature_test[stock]=tmp


# train model
context.ML.fit(X,y)
for stock in data:
    if stock in feature_test :
        # predict by sum of predictions
        predict=sum(context.ML.predict(feature_test[stock]))
        # if predicted move up, add stock to holding list
        if predict>1.5:
            hold_list.append(stock)

# calcualte weight according to holding list
weight= float(context.target_levervage / len(hold_list))\
                if hold_list else 0.0
# Make orders
for stock in data:
    if stock in hold_list:
        order_target_percent(stock, weight)
    else:
        order_target_percent(stock, 0.0)
```