HW2 Q1

February 4, 2024

In this notebook we explore the performance of thee different classifiers: (DecisionTree, BaggingClassifier, and RandomForestClassifiers) on financial data, and compare their performance after finding the optimal hyperparameters (using Optuna) and the optimal feature sets (using Shap).

The objective for optimisations will be profit (defined traditionally), and we backl test the performance of the models over the period 2010 - 2018, and finally compute the Sharpe Ratios, Information Ratios and alpha for the three classifier strategies and the buy-and-hold strategy for SPY.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
import sklearn
from sklearn.preprocessing import StandardScaler

pd.set_option('use_inf_as_na', True)
from collections import Counter

from tqdm import tqdm # to measure progress of for loops
```

```
/var/folders/sp/wlr6xm297918vx6kjh2z1dk00000gn/T/ipykernel_2415/524871555.py:12:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  pd.set_option('use_inf_as_na', True)
```

[2]:
```
# load dataset
raw_data = pd.read_pickle('dataset.pkl')
data = raw_data.drop([x  for x in raw_data.columns if 'fqtr' in x],axis=1)
# restrict companies to market cap > 1 billion
data = raw_data[raw_data['market_cap'] > 1000.0]
data = data.copy()
# fill in missing values
data.fillna(0.0,inplace=True)
```

0.0.1 Insert a column in dataset based on stock performance

Inserting a column in the dataset where entries are:
- 1 if the stock return is more than 5% higher than the SPY return
- 0 if it is between -10% and +5% relative to the SPY return
- -1 if it is less then -10% relative to the SPY return

[3]:
```python
# function to return appropriate values based on performance as detailed above
def f(x):
    if x > 0.05:
        return 1
    elif x < -0.1:
        return -1
    else:
        return 0
```

[4]:
```python
# add the new column
data['rel_performance'] = data['pred_rel_return'].apply(f)
# make the date the index
data.reset_index(inplace=True)
data.set_index('date',inplace=True)
```

[5]:
```python
data[['pred_rel_return', 'rel_performance']][100:110]
```

[5]:
```
            pred_rel_return  rel_performance
date
2000-04-03         0.261336                1
2000-04-05        -0.337466               -1
2000-04-06         0.272372                1
2000-04-07        -0.645219               -1
2000-04-07         0.192161                1
2000-04-11         0.153474                1
2000-04-11         0.317203                1
2000-04-11        -0.051039                0
2000-04-11         0.028730                0
2000-04-11         0.095712                1
```

[6]:
```python
print(data.index)
```

```
DatetimeIndex(['2000-02-09', '2000-02-09', '2000-02-10', '2000-02-11',
               '2000-02-15', '2000-02-16', '2000-02-16', '2000-02-16',
               '2000-02-16', '2000-02-16',
               …
               '2018-12-21', '2018-12-21', '2018-12-21', '2018-12-21',
               '2018-12-21', '2018-12-21', '2018-12-21', '2018-12-21',
               '2018-12-21', '2018-12-24'],
              dtype='datetime64[ns]', name='date', length=111468, freq=None)
```

0.0.2 Split the data into training, validation, and test

- Data for training period is from 2007 to 2009 (inclusive, i.e. 3 years)
- Data for validation period is 1 quarter after end of training period
- Data for test period is immediately proceeding the training period

[7]:
```
df_1 = data.loc['2007-01-01':'2010-01-01']
df_valid = data.loc['2010-04-01':'2010-07-01']
df_test = data.loc['2010-07-01':'2010-10-01']
```

[8]:
```
train_1 = df_1.reset_index().drop(['ticker','date',
                                    'next_period_return',
                                    'spy_next_period_return',
                                    'rel_performance','pred_rel_return',
                                   'return', 'cum_ret', 'spy_cum_ret'],axis=1)

valid = df_valid.reset_index().drop(['ticker','date',
                                     'next_period_return',
                                    'spy_next_period_return',
                                    'rel_performance','pred_rel_return',
                                   'return', 'cum_ret', 'spy_cum_ret'],axis=1)

test = df_test.reset_index().drop(['ticker','date',
                                    'next_period_return',
                                    'spy_next_period_return',
                                    'rel_performance','pred_rel_return',
                                   'return', 'cum_ret', 'spy_cum_ret'],axis=1)
```

[9]:
```
# Obtain the y values for each data split
train_1_stock_returns = df_1['next_period_return']
valid_stock_returns = df_valid['next_period_return']
test_stock_returns = df_test['next_period_return']

y_1 = df_1['rel_performance']
y_valid = df_valid['rel_performance']
y_test = df_test['rel_performance']

y_1 = y_1.values
y_valid = y_valid.values
y_test = y_test.values
```

0.0.3 Import Optuna to find the optimal hyperparameters for the classifiers

[10]:
```
import optuna
from optuna.trial import Trial
# optuna.logging.set_verbosity(optuna.logging.FATAL)
import warnings
warnings.filterwarnings("ignore")
```

0.1 Defining the Optuna objective function for our 3 classifiers:

- DecisionTree classifier
- Bagging classifier
- RandomForest classifier

Note that in each case we are optimizing for the profit rather than the accuracy

[11]:
```
def objective_tree(trial:
↪Trial,train=None,labels=None,val=None,val_labels=None,val_rets=None):

    t_min_samples_leaf = trial.suggest_int('min_samples_leaf',100,1200,step=200)
    t_max_depth = trial.suggest_int('max_depth',5,25,step=5)

    tree_clf = DecisionTreeClassifier(min_samples_leaf =␣
↪t_min_samples_leaf,max_depth=t_max_depth,random_state=123)
    tree_clf.fit(train,labels)

    preds = tree_clf.predict(val)
    profit = (preds * val_rets).sum()

    return profit
```

[12]:
```
def objective_bagging(trial:
↪Trial,train=None,labels=None,val=None,val_labels=None,val_rets=None):

    t_min_samples_leaf = trial.suggest_int('min_samples_leaf',100,1200,step=200)
    t_max_depth = trial.suggest_int('max_depth',5,25,step=5)
    t_n_estimators = trial.suggest_int('n_estimators',5,50,step=5)

    t_clf = DecisionTreeClassifier(min_samples_leaf =␣
↪t_min_samples_leaf,max_depth=t_max_depth,random_state=123)
    bg_clf =␣
↪BaggingClassifier(t_clf,n_estimators=t_n_estimators,random_state=123,n_jobs=1)
    bg_clf.fit(train,labels)

    preds = bg_clf.predict(val)
    profit = (preds * val_rets).sum()

    return profit
```

[13]:
```
def objective_rf(trial:
↪Trial,train=None,labels=None,val=None,val_labels=None,val_rets=None):

    rf_n_estimators = trial.suggest_int('n_estimators', 10,40,step=5)
    rf_max_features = trial.suggest_categorical('max_features',['sqrt','log2'])
    rf_min_samples_leaf = trial.
↪suggest_int('min_samples_leaf',800,2400,step=800)
```

```
    rf_max_depth = trial.suggest_int('max_depth',4,15)

    rf_clf = RandomForestClassifier(n_estimators=rf_n_estimators,
                                    max_depth=rf_max_depth,
                                    min_samples_leaf=rf_min_samples_leaf,
                                    ␣
↪max_features=rf_max_features,random_state=123)
    rf_clf.fit(train,labels)

    preds = rf_clf.predict(val)
    profit = (preds * val_rets).sum()

    return profit
```

[14]:
```
study_tree = optuna.create_study(direction="maximize")
study_bagging = optuna.create_study(direction="maximize")
study_rf = optuna.create_study(direction="maximize")
```

```
[I 2024-02-04 20:38:13,061] A new study created in memory with name: no-
name-9c8e0ef4-6fad-4938-b3df-43bc4decf7d1
[I 2024-02-04 20:38:13,061] A new study created in memory with name: no-
name-32573266-7473-4929-8fd1-9ea6b5732dfe
[I 2024-02-04 20:38:13,061] A new study created in memory with name: no-name-
dabe04c2-f647-4709-824c-d3b1f871bb0e
```

0.1.1 Run optimizations to find optimal parameters

[15]:
```
from functools import partial
```

[16]:
```
%%time
study_tree.
 ↪optimize(partial(objective_tree,train=train_1,labels=y_1,val=valid,val_labels=y_valid,val_r
 ↪n_trials=200,n_jobs=-1)
```

```
[I 2024-02-04 20:38:14,225] Trial 8 finished with value: -42.11555900000003 and
parameters: {'min_samples_leaf': 1100, 'max_depth': 15}. Best is trial 8 with
value: -42.11555900000003.
[I 2024-02-04 20:38:14,260] Trial 6 finished with value: -20.87378900000001 and
parameters: {'min_samples_leaf': 700, 'max_depth': 5}. Best is trial 6 with
value: -20.87378900000001.
[I 2024-02-04 20:38:14,283] Trial 3 finished with value: -33.88995700000003 and
parameters: {'min_samples_leaf': 900, 'max_depth': 15}. Best is trial 6 with
value: -20.87378900000001.
[I 2024-02-04 20:38:14,289] Trial 1 finished with value: -33.88995700000003 and
parameters: {'min_samples_leaf': 900, 'max_depth': 25}. Best is trial 6 with
value: -20.87378900000001.
[I 2024-02-04 20:38:14,361] Trial 2 finished with value: -33.88995700000003 and
parameters: {'min_samples_leaf': 900, 'max_depth': 15}. Best is trial 6 with
```

```
value: -20.46544500000001.
[I 2024-02-04 20:38:38,875] Trial 196 finished with value: -20.46544500000001
and parameters: {'min_samples_leaf': 700, 'max_depth': 25}. Best is trial 4 with
value: -20.46544500000001.
[I 2024-02-04 20:38:38,933] Trial 198 finished with value: -20.46544500000001
and parameters: {'min_samples_leaf': 700, 'max_depth': 20}. Best is trial 4 with
value: -20.46544500000001.
[I 2024-02-04 20:38:39,249] Trial 199 finished with value: -20.46544500000001
and parameters: {'min_samples_leaf': 700, 'max_depth': 20}. Best is trial 4 with
value: -20.46544500000001.
```

```
CPU times: user 3min 48s, sys: 2.72 s, total: 3min 51s
Wall time: 26.2 s
```

[17]:

```
%%time
study_bagging.
 ↪optimize(partial(objective_bagging,train=train_1,labels=y_1,val=valid,val_labels=y_valid,va
 ↪n_trials=200,n_jobs=-1)
```

```
[I 2024-02-04 20:38:43,087] Trial 9 finished with value: -33.29038000000004 and
parameters: {'min_samples_leaf': 300, 'max_depth': 5, 'n_estimators': 5}. Best
is trial 9 with value: -33.29038000000004.
[I 2024-02-04 20:38:44,820] Trial 0 finished with value: -37.487501000000044 and
parameters: {'min_samples_leaf': 900, 'max_depth': 10, 'n_estimators': 10}. Best
is trial 9 with value: -33.29038000000004.
[I 2024-02-04 20:38:48,573] Trial 3 finished with value: -34.22570200000003 and
parameters: {'min_samples_leaf': 700, 'max_depth': 5, 'n_estimators': 15}. Best
is trial 9 with value: -33.29038000000004.
[I 2024-02-04 20:38:50,427] Trial 1 finished with value: -38.103020000000036 and
parameters: {'min_samples_leaf': 900, 'max_depth': 5, 'n_estimators': 20}. Best
is trial 9 with value: -33.29038000000004.
[I 2024-02-04 20:38:51,806] Trial 6 finished with value: -34.67733300000003 and
parameters: {'min_samples_leaf': 700, 'max_depth': 15, 'n_estimators': 20}. Best
is trial 9 with value: -33.29038000000004.
[I 2024-02-04 20:38:56,828] Trial 4 finished with value: -39.066652000000026 and
parameters: {'min_samples_leaf': 1100, 'max_depth': 20, 'n_estimators': 35}.
Best is trial 9 with value: -33.29038000000004.
[I 2024-02-04 20:38:58,185] Trial 8 finished with value: -35.50093700000003 and
parameters: {'min_samples_leaf': 700, 'max_depth': 20, 'n_estimators': 30}. Best
is trial 9 with value: -33.29038000000004.
[I 2024-02-04 20:39:02,119] Trial 12 finished with value: -38.14073200000004 and
parameters: {'min_samples_leaf': 900, 'max_depth': 15, 'n_estimators': 25}. Best
is trial 9 with value: -33.29038000000004.
[I 2024-02-04 20:39:03,578] Trial 5 finished with value: -38.74424600000003 and
parameters: {'min_samples_leaf': 500, 'max_depth': 5, 'n_estimators': 35}. Best
is trial 9 with value: -33.29038000000004.
[I 2024-02-04 20:39:08,115] Trial 10 finished with value: -28.781955000000053
and parameters: {'min_samples_leaf': 100, 'max_depth': 10, 'n_estimators': 20}.
Best is trial 10 with value: -28.781955000000053.
```

```
[I 2024-02-04 20:42:52,010] Trial 159 finished with value: -30.705330000000068
and parameters: {'min_samples_leaf': 100, 'max_depth': 10, 'n_estimators': 35}.
Best is trial 21 with value: -26.708862000000046.
[I 2024-02-04 20:42:52,226] Trial 190 finished with value: -26.708862000000046
and parameters: {'min_samples_leaf': 100, 'max_depth': 10, 'n_estimators': 5}.
Best is trial 21 with value: -26.708862000000046.
[I 2024-02-04 20:42:54,294] Trial 191 finished with value: -26.708862000000046
and parameters: {'min_samples_leaf': 100, 'max_depth': 10, 'n_estimators': 5}.
Best is trial 21 with value: -26.708862000000046.
[I 2024-02-04 20:42:54,494] Trial 161 finished with value: -30.705330000000068
and parameters: {'min_samples_leaf': 100, 'max_depth': 10, 'n_estimators': 35}.
Best is trial 21 with value: -26.708862000000046.
[I 2024-02-04 20:42:55,467] Trial 193 finished with value: -26.708862000000046
and parameters: {'min_samples_leaf': 100, 'max_depth': 10, 'n_estimators': 5}.
Best is trial 21 with value: -26.708862000000046.
[I 2024-02-04 20:42:55,532] Trial 192 finished with value: -26.708862000000046
and parameters: {'min_samples_leaf': 100, 'max_depth': 10, 'n_estimators': 5}.
Best is trial 21 with value: -26.708862000000046.
[I 2024-02-04 20:42:56,268] Trial 194 finished with value: -26.708862000000046
and parameters: {'min_samples_leaf': 100, 'max_depth': 10, 'n_estimators': 5}.
Best is trial 21 with value: -26.708862000000046.
[I 2024-02-04 20:42:57,417] Trial 195 finished with value: -26.708862000000046
and parameters: {'min_samples_leaf': 100, 'max_depth': 10, 'n_estimators': 5}.
Best is trial 21 with value: -26.708862000000046.
[I 2024-02-04 20:42:57,446] Trial 188 finished with value: -29.53389800000005
and parameters: {'min_samples_leaf': 100, 'max_depth': 10, 'n_estimators': 10}.
Best is trial 21 with value: -26.708862000000046.
[I 2024-02-04 20:42:57,718] Trial 189 finished with value: -29.53389800000005
and parameters: {'min_samples_leaf': 100, 'max_depth': 10, 'n_estimators': 10}.
Best is trial 21 with value: -26.708862000000046.
[I 2024-02-04 20:42:57,868] Trial 196 finished with value: -26.708862000000046
and parameters: {'min_samples_leaf': 100, 'max_depth': 10, 'n_estimators': 5}.
Best is trial 21 with value: -26.708862000000046.
[I 2024-02-04 20:43:02,906] Trial 197 finished with value: -29.53389800000005
and parameters: {'min_samples_leaf': 100, 'max_depth': 10, 'n_estimators': 10}.
Best is trial 21 with value: -26.708862000000046.
[I 2024-02-04 20:43:04,754] Trial 198 finished with value: -29.53389800000005
and parameters: {'min_samples_leaf': 100, 'max_depth': 10, 'n_estimators': 10}.
Best is trial 21 with value: -26.708862000000046.
[I 2024-02-04 20:43:05,056] Trial 199 finished with value: -29.53389800000005
and parameters: {'min_samples_leaf': 100, 'max_depth': 10, 'n_estimators': 10}.
Best is trial 21 with value: -26.708862000000046.

CPU times: user 38min 43s, sys: 15 s, total: 38min 58s
Wall time: 4min 25s
```

[18]:

```
%%time
```

```
study_rf.
↪optimize(partial(objective_rf,train=train_1,labels=y_1,val=valid,val_labels=y_valid,val_ret
↪n_trials=200,n_jobs=-1)
```

```
[I 2024-02-04 20:43:05,336] Trial 4 finished with value: -40.74410800000003 and
parameters: {'n_estimators': 15, 'max_features': 'log2', 'min_samples_leaf':
800, 'max_depth': 11}. Best is trial 4 with value: -40.74410800000003.
[I 2024-02-04 20:43:05,346] Trial 6 finished with value: -39.25296700000003 and
parameters: {'n_estimators': 10, 'max_features': 'sqrt', 'min_samples_leaf':
1600, 'max_depth': 6}. Best is trial 6 with value: -39.25296700000003.
[I 2024-02-04 20:43:05,413] Trial 8 finished with value: -40.00959700000004 and
parameters: {'n_estimators': 10, 'max_features': 'sqrt', 'min_samples_leaf':
800, 'max_depth': 15}. Best is trial 6 with value: -39.25296700000003.
[I 2024-02-04 20:43:05,451] Trial 9 finished with value: -37.859784000000026 and
parameters: {'n_estimators': 20, 'max_features': 'sqrt', 'min_samples_leaf':
2400, 'max_depth': 15}. Best is trial 9 with value: -37.859784000000026.
[I 2024-02-04 20:43:05,464] Trial 7 finished with value: -38.53366700000004 and
parameters: {'n_estimators': 30, 'max_features': 'log2', 'min_samples_leaf':
2400, 'max_depth': 11}. Best is trial 9 with value: -37.859784000000026.
[I 2024-02-04 20:43:05,491] Trial 3 finished with value: -40.77534300000005 and
parameters: {'n_estimators': 25, 'max_features': 'log2', 'min_samples_leaf':
800, 'max_depth': 14}. Best is trial 9 with value: -37.859784000000026.
[I 2024-02-04 20:43:05,586] Trial 1 finished with value: -40.466121000000044 and
parameters: {'n_estimators': 35, 'max_features': 'log2', 'min_samples_leaf':
1600, 'max_depth': 9}. Best is trial 9 with value: -37.859784000000026.
[I 2024-02-04 20:43:05,638] Trial 13 finished with value: -40.13517700000003 and
parameters: {'n_estimators': 10, 'max_features': 'log2', 'min_samples_leaf':
800, 'max_depth': 13}. Best is trial 9 with value: -37.859784000000026.
[I 2024-02-04 20:43:05,676] Trial 0 finished with value: -35.91543100000004 and
parameters: {'n_estimators': 25, 'max_features': 'sqrt', 'min_samples_leaf':
1600, 'max_depth': 8}. Best is trial 0 with value: -35.91543100000004.
[I 2024-02-04 20:43:05,729] Trial 5 finished with value: -41.546579000000044 and
parameters: {'n_estimators': 40, 'max_features': 'log2', 'min_samples_leaf':
800, 'max_depth': 12}. Best is trial 0 with value: -35.91543100000004.
[I 2024-02-04 20:43:05,739] Trial 2 finished with value: -41.546579000000044 and
parameters: {'n_estimators': 40, 'max_features': 'log2', 'min_samples_leaf':
800, 'max_depth': 10}. Best is trial 0 with value: -35.91543100000004.
[I 2024-02-04 20:43:05,774] Trial 12 finished with value: -40.00959700000004 and
parameters: {'n_estimators': 10, 'max_features': 'sqrt', 'min_samples_leaf':
800, 'max_depth': 10}. Best is trial 0 with value: -35.91543100000004.
[I 2024-02-04 20:43:05,951] Trial 10 finished with value: -37.08585400000004 and
parameters: {'n_estimators': 20, 'max_features': 'sqrt', 'min_samples_leaf':
800, 'max_depth': 7}. Best is trial 0 with value: -35.91543100000004.
[I 2024-02-04 20:43:05,966] Trial 14 finished with value: -40.466121000000044
and parameters: {'n_estimators': 35, 'max_features': 'log2', 'min_samples_leaf':
1600, 'max_depth': 14}. Best is trial 0 with value: -35.91543100000004.
[I 2024-02-04 20:43:06,059] Trial 11 finished with value: -40.38669600000003 and
parameters: {'n_estimators': 40, 'max_features': 'sqrt', 'min_samples_leaf':
```

```
1600, 'max_depth': 7}. Best is trial 15 with value: -35.43491800000005.
[I 2024-02-04 20:43:16,591] Trial 190 finished with value: -35.548178000000036
and parameters: {'n_estimators': 20, 'max_features': 'sqrt', 'min_samples_leaf':
1600, 'max_depth': 7}. Best is trial 15 with value: -35.43491800000005.
[I 2024-02-04 20:43:16,660] Trial 193 finished with value: -35.548178000000036
and parameters: {'n_estimators': 20, 'max_features': 'sqrt', 'min_samples_leaf':
1600, 'max_depth': 7}. Best is trial 15 with value: -35.43491800000005.
[I 2024-02-04 20:43:16,852] Trial 195 finished with value: -35.548178000000036
and parameters: {'n_estimators': 20, 'max_features': 'sqrt', 'min_samples_leaf':
1600, 'max_depth': 9}. Best is trial 15 with value: -35.43491800000005.
[I 2024-02-04 20:43:16,879] Trial 197 finished with value: -35.548178000000036
and parameters: {'n_estimators': 20, 'max_features': 'sqrt', 'min_samples_leaf':
1600, 'max_depth': 9}. Best is trial 15 with value: -35.43491800000005.
[I 2024-02-04 20:43:16,888] Trial 198 finished with value: -35.548178000000036
and parameters: {'n_estimators': 20, 'max_features': 'sqrt', 'min_samples_leaf':
1600, 'max_depth': 9}. Best is trial 15 with value: -35.43491800000005.
[I 2024-02-04 20:43:16,907] Trial 199 finished with value: -35.548178000000036
and parameters: {'n_estimators': 20, 'max_features': 'sqrt', 'min_samples_leaf':
1600, 'max_depth': 9}. Best is trial 15 with value: -35.43491800000005.
[I 2024-02-04 20:43:16,924] Trial 192 finished with value: -37.329935000000035
and parameters: {'n_estimators': 40, 'max_features': 'sqrt', 'min_samples_leaf':
1600, 'max_depth': 7}. Best is trial 15 with value: -35.43491800000005.
[I 2024-02-04 20:43:17,140] Trial 196 finished with value: -37.329935000000035
and parameters: {'n_estimators': 40, 'max_features': 'sqrt', 'min_samples_leaf':
1600, 'max_depth': 12}. Best is trial 15 with value: -35.43491800000005.
[I 2024-02-04 20:43:17,141] Trial 194 finished with value: -37.329935000000035
and parameters: {'n_estimators': 40, 'max_features': 'sqrt', 'min_samples_leaf':
1600, 'max_depth': 12}. Best is trial 15 with value: -35.43491800000005.
```

```
CPU times: user 1min 14s, sys: 5 s, total: 1min 19s
Wall time: 12.1 s
```

0.1.2 Instantiate the classifiers with the best parameters

[19]:
```
print("Best parameters for decision tree: ", study_tree.best_params)
print("Best parameters for bagging: ", study_bagging.best_params)
print("Best parameters for random forest: ", study_rf.best_params)
```

```
Best parameters for decision tree:  {'min_samples_leaf': 700, 'max_depth': 25}
Best parameters for bagging:  {'min_samples_leaf': 100, 'max_depth': 10,
'n_estimators': 5}
Best parameters for random forest:  {'n_estimators': 30, 'max_features': 'sqrt',
'min_samples_leaf': 800, 'max_depth': 15}
```

[20]:
```
t_clf = DecisionTreeClassifier(**study_tree.best_params,random_state=123)
```

[21]:
```
tree_cfl = DecisionTreeClassifier(**{'min_samples_leaf': study_bagging.
 ↪best_params['min_samples_leaf'],
```

```
                                     'max_depth': study_bagging.
 ↪best_params['max_depth']})
bg_clf = BaggingClassifier(tree_cfl,n_estimators=study_bagging.
 ↪best_params['n_estimators'],random_state=123 ,n_jobs=-1)
```

[22]:
```
rf_clf = RandomForestClassifier(**study_rf.best_params)
```

0.1.3 Train each of the classifiers

[23]:
```
t_clf.fit(train_1,y_1)
```

[23]:
```
DecisionTreeClassifier(max_depth=25, min_samples_leaf=700, random_state=123)
```

[24]:
```
bg_clf.fit(train_1,y_1)
```

[24]:
```
BaggingClassifier(estimator=DecisionTreeClassifier(max_depth=10,
                                                   min_samples_leaf=100),
                  n_estimators=5, n_jobs=-1, random_state=123)
```

[25]:
```
rf_clf.fit(train_1,y_1)
```

[25]:
```
RandomForestClassifier(max_depth=15, min_samples_leaf=800, n_estimators=30)
```

0.1.4 Back test the performance of the models over the period 2010 - 2018

[26]:
```
start_dates = [pd.to_datetime('2010-01-01') + pd.DateOffset(months = 3*i) for i␣
 ↪in range(21)]
end_dates = [d + pd.DateOffset(months = 36) for d in start_dates]
# So the first period is [2010 Jan 1 - 2013 Jan 1], and the last period is␣
 ↪[2015 Jan 1 - 2018 Jan 1]
```

[27]:
```
training_frames = [data.loc[d:d+pd.DateOffset(months = 36)] for d in␣
 ↪start_dates]
valid_frames = [data.loc[d + pd.DateOffset(months=3):d+pd.DateOffset(months =␣
 ↪6)] for d in end_dates]
test_frames = [data.loc[d + pd.DateOffset(months=6):d+pd.DateOffset(months =␣
 ↪9)] for d in end_dates]

training_labels = [d['rel_performance'].values for d in training_frames]

training_stock_returns = [d['next_period_return'].values for d in␣
 ↪training_frames]
test_stock_returns = [d['next_period_return'] for d in test_frames]
```

[28]:
```
training_data = [df.reset_index().drop(['ticker','date',
                                 'next_period_return',
                                 'spy_next_period_return',
```

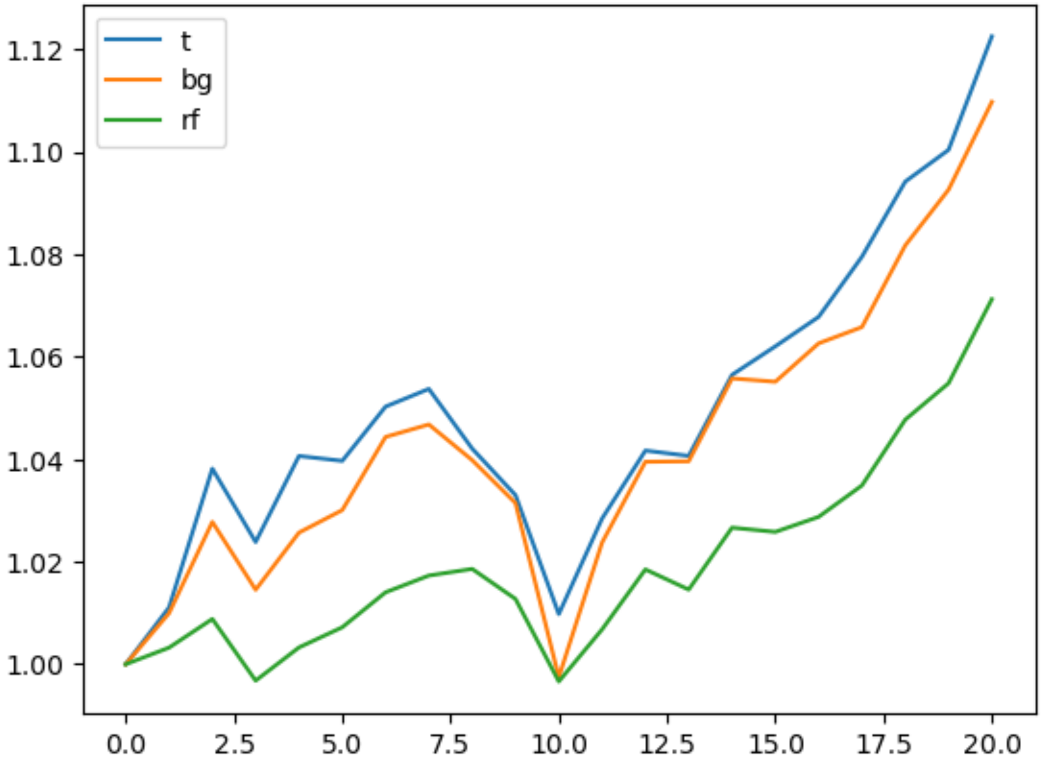
```
                                        'rel_performance', 'pred_rel_return',
                                       'return', 'cum_ret', 'spy_cum_ret'],axis=1)␣
 ↪for df in training_frames]
valid_data = [df.reset_index().drop(['ticker','date',
                                        'next_period_return',
                                        'spy_next_period_return',
                                        'rel_performance','pred_rel_return',
                                       'return', 'cum_ret', 'spy_cum_ret'],axis=1)␣
 ↪for df in valid_frames]
test_data = [df.reset_index().drop(['ticker','date',
                                        'next_period_return',
                                        'spy_next_period_return',
                                        'rel_performance','pred_rel_return',
                                       'return', 'cum_ret', 'spy_cum_ret'],axis=1)␣
 ↪for df in test_frames]
```

[29]:
```
# dictionary to store performance and returns
xs = {'t':[1], 'bg':[1], 'rf':[1]}
rets = {'t':[], 'bg':[], 'rf':[]}
models = {'t':t_clf, 'bg':bg_clf, 'rf':rf_clf}

for i in tqdm(range(len(start_dates)-1)):
    for key, model in models.items():
        model.fit(training_data[i],training_labels[i])

        preds = model.predict(test_data[i])
        profit_i = (preds*test_stock_returns[i]).sum()

        rets[key].append(profit_i)
        num_names = len(test_data[i])
        xs[key].append(xs[key][i] + (xs[key][i]/num_names)*profit_i)
```

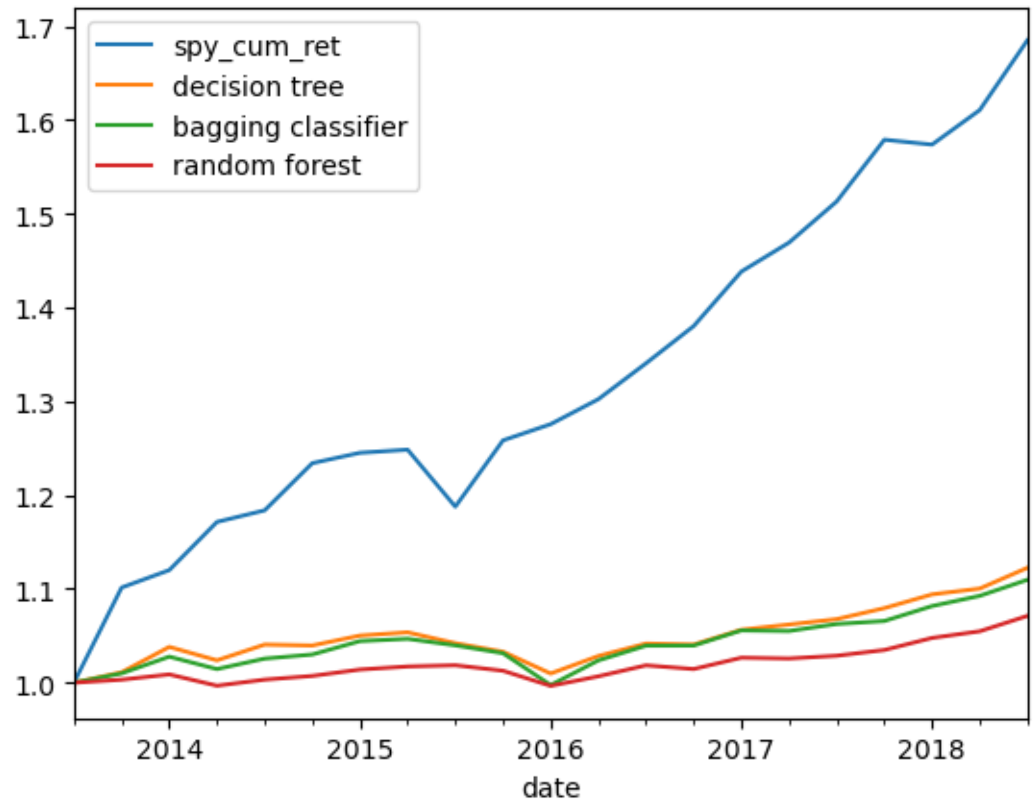
```
100%|                         | 20/20 [01:28<00:00,  4.41s/it]
```

[30]:
```
for key, x_list in xs.items():
    plt.plot(x_list, label = key);
plt.legend()
```

[30]:
```
<matplotlib.legend.Legend at 0x2c5f0f490>
```



[31]:
```
# Compare to buy and hold of SPY
SPY = pd.read_pickle(r'SPY_cum_ret.pkl')
SPY = SPY.loc['2013-07-01':'2018-09-30']
SPY = SPY.resample('Q').ffill()
SPY['spy_cum_ret'] = (SPY['spy_cum_ret'] - SPY['spy_cum_ret'][0]+1)
SPY['decision tree'] = xs['t']
SPY['bagging classifier'] = xs['bg']
SPY['random forest'] = xs['rf']
SPY.plot();
```



0.1.5 Now we can compute the Shapley values for these models and see how the performance changes

0.1.6 Finding the features with non zero Shapley values

[32]:
```
# Retrain the models using the original training set (i.e. before backtesting)
t_clf.fit(train_1,y_1)
bg_clf.fit(train_1,y_1)
rf_clf.fit(train_1,y_1)
```

[32]:
```
RandomForestClassifier(max_depth=15, min_samples_leaf=800, n_estimators=30)
```

[33]:
```
# Obtain the feature importances
def tree_feat_importance(m, df):
    return pd.DataFrame({'cols':df.columns, 'feat_imp':m.feature_importances_}
                       ).sort_values('feat_imp', ascending=False)

def bagging_feat_importance(m, df):
    feature_importances = []
    for est in m.estimators_:
        fi = est.feature_importances_
```

```
        feature_importances.append(fi)
    feature_importances = np.array(feature_importances)

    return pd.DataFrame({'cols':train_1.columns, 'feat_imp':np.
 ↪mean(feature_importances,axis=0)}
                       ).sort_values('feat_imp', ascending=False)

def randomforest_feat_importances(m, df):

    return pd.DataFrame({'cols':df.columns, 'feat_imp': m.feature_importances_}
                       ).sort_values('feat_imp', ascending=False)

def plot_fi(fi): return fi.plot('cols', 'feat_imp', 'barh', figsize=(12,7),␣
 ↪legend=False)
```

[34]:
```
t_fi = tree_feat_importance(t_clf,train_1)
bg_fi = bagging_feat_importance(bg_clf,train_1)
rf_fi = randomforest_feat_importances(rf_clf,train_1)
```

[35]:
```
# Only use features that have positive feature importance
t_features = t_fi[(t_fi['feat_imp'] > 0.00)]
bg_features = bg_fi[(bg_fi['feat_imp'] > 0.00)]
rf_features = rf_fi[(rf_fi['feat_imp'] > 0.00)]
```

[36]:
```
train_t = train_1[t_features['cols'].values]
valid_t = valid[t_features['cols'].values]
valid_t['returns'] = valid_stock_returns.values

train_bg = train_1[bg_features['cols'].values]
valid_bg = valid[bg_features['cols'].values]
valid_bg['returns'] = valid_stock_returns.values

train_rf = train_1[rf_features['cols'].values]
valid_rf = valid[rf_features['cols'].values]
valid_rf['returns'] = valid_stock_returns.values
```

[37]:
```
print(f"Number of features used for decision tree classifier reduced from:␣
 ↪{train_1.shape[1]} to {train_t.shape[1]}")
print(f"Number of features used for bagging classifier reduced from      :␣
 ↪{train_1.shape[1]} to {train_bg.shape[1]}")
print(f"Number of features used for random forest classifier reduced from:␣
 ↪{train_1.shape[1]} to {train_rf.shape[1]}")
```

```
Number of features used for decision tree classifier reduced from: 725 to 16
Number of features used for bagging classifier reduced from      : 725 to 133
Number of features used for random forest classifier reduced from: 725 to 121
```

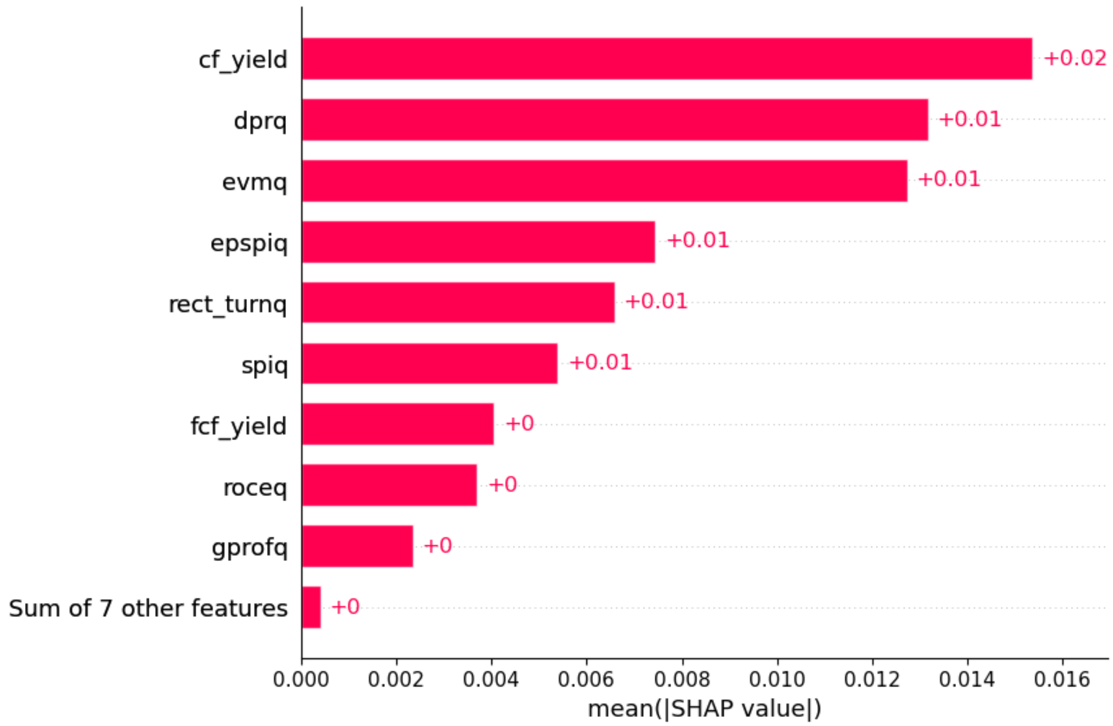
[38]:
```python
# Retrain the classifiers using the new feature set
t_clf.fit(train_t,y_1)
bg_clf.fit(train_bg,y_1)
rf_clf.fit(train_rf,y_1)
```

[38]: RandomForestClassifier(max_depth=15, min_samples_leaf=800, n_estimators=30)

[39]:
```python
import shap

def model_t(features):
    tree_features = features[features.columns[:-1].values]
    pred = t_clf.predict(tree_features)
    ret = pred * features[features.columns[-1]]
    return ret

def model_bg(features):
    bagging_features = features[features.columns[:-1].values]
    pred = bg_clf.predict(bagging_features)
    ret = pred * features[features.columns[-1]]
    return ret

def model_rf(features):
    rf_features = features[features.columns[:-1].values]
    pred = rf_clf.predict(rf_features)
    ret = pred * features[features.columns[-1]]
    return ret
```

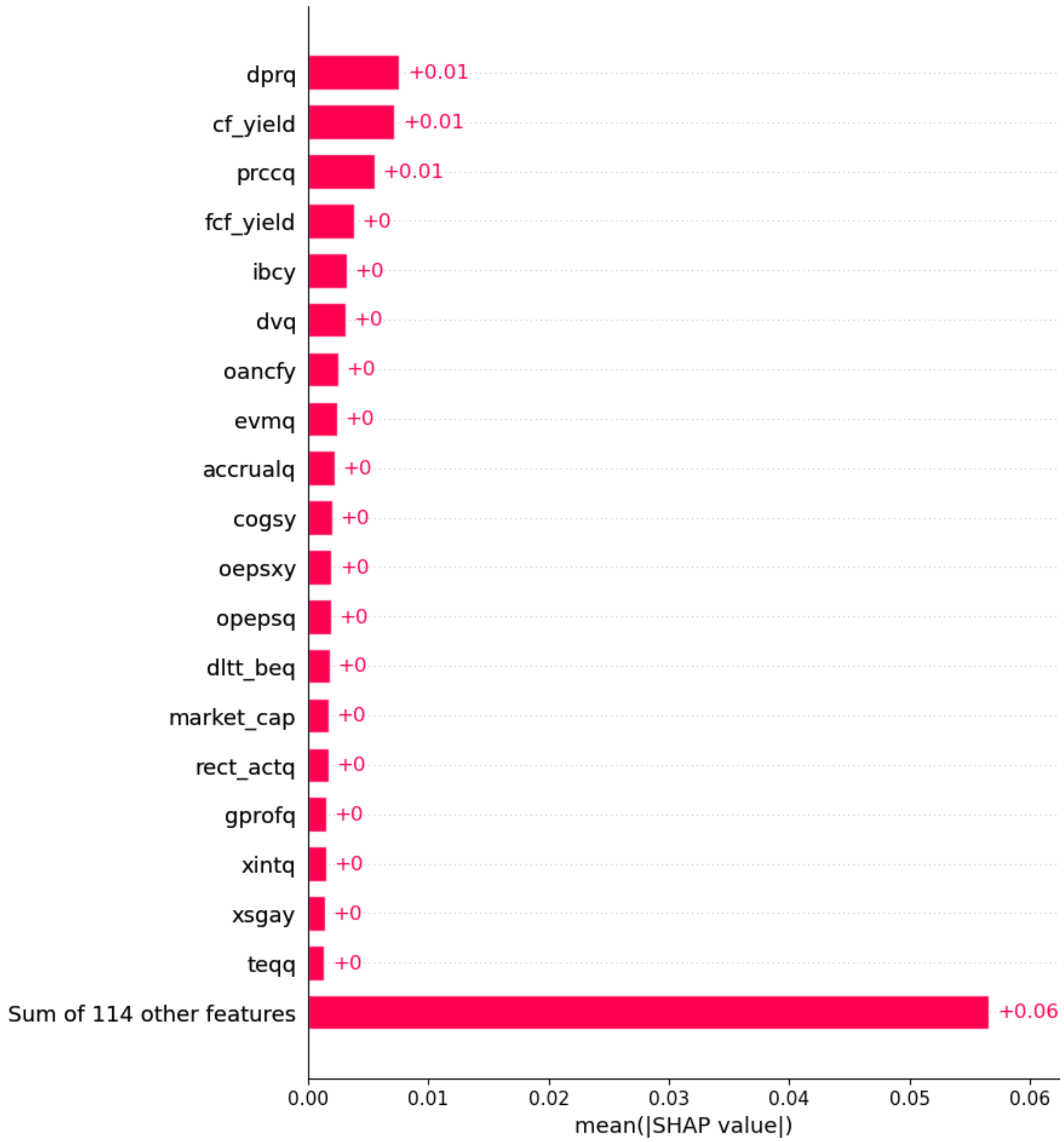
[40]:
```python
# Shapley for tree classifier
model_t(valid_t)
t_explainer = shap.explainers.Permutation(model_t,valid_t)
t_shap_values = t_explainer(valid_t,max_evals=2000)
shap.plots.bar(t_shap_values[:,:-1],max_display=10)
```

PermutationExplainer explainer: 1442it [00:54, 21.85it/s]



[41]:
```
# Shapley for bagging classifier
model_bg(valid_bg)
bg_explainer = shap.explainers.Permutation(model_bg,valid_bg)
bg_shap_values = bg_explainer(valid_bg,max_evals=2000)
shap.plots.bar(bg_shap_values[:,:-1],max_display=20)
```

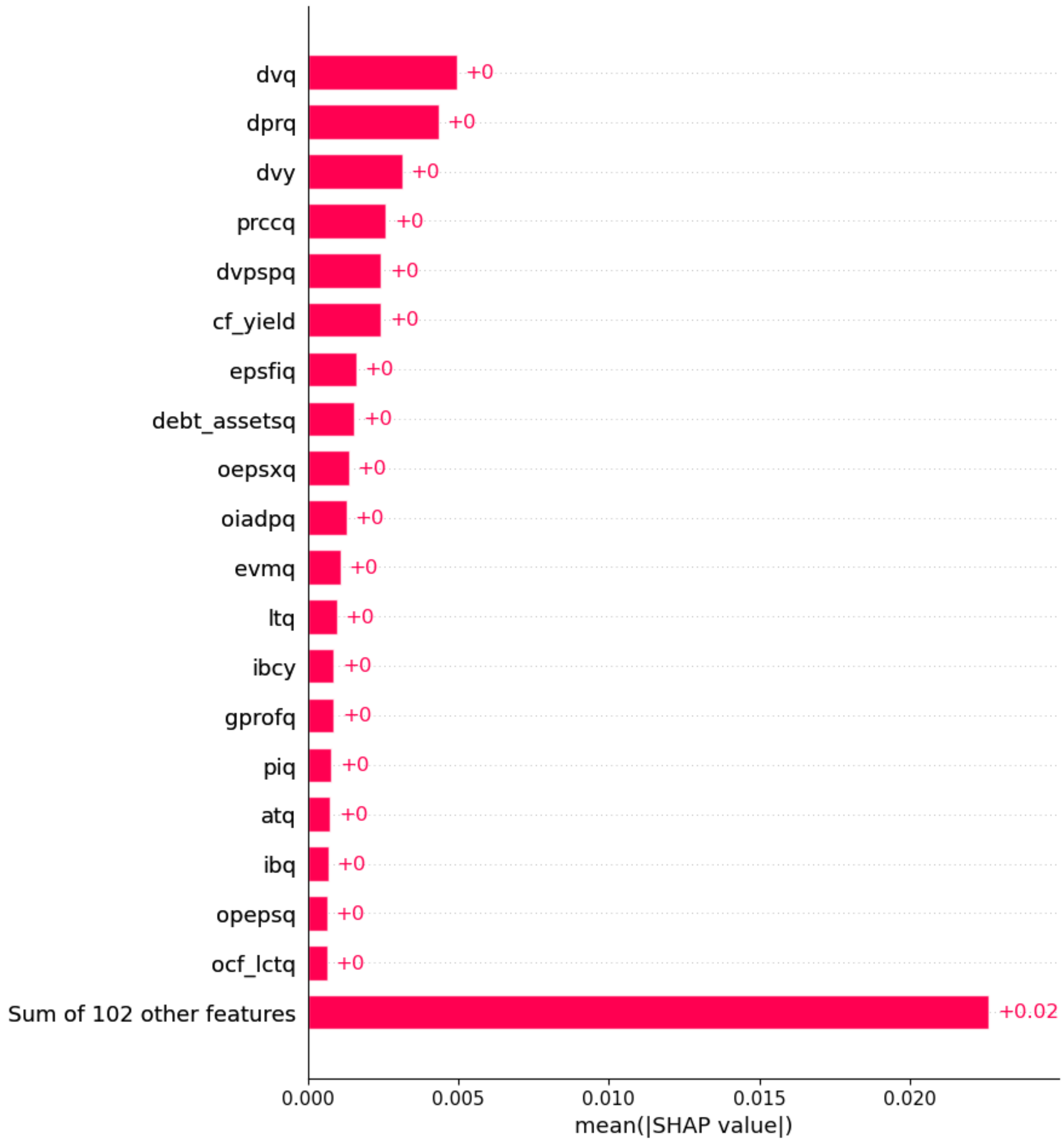
```
PermutationExplainer explainer: 1442it [35:30,  1.48s/it]
```



[42]:

```
# Shapley for random forest classifier
model_rf(valid_rf)
rf_explainer = shap.explainers.Permutation(model_rf,valid_rf)
rf_shap_values = rf_explainer(valid_rf,max_evals=2000)
shap.plots.bar(rf_shap_values[:,:-1],max_display=20)
```

```
PermutationExplainer explainer: 1442it [05:45,  4.06it/s]
```



0.1.7 Retrain the models with the features that have non-zero Shapley values

[43]:
```
t_cols = t_features['cols'].values
t_shap_cols = t_cols[np.abs(t_shap_values[:,:-1].values).mean(axis=0)>0.000]

bg_cols = bg_features['cols'].values
bg_shap_cols = bg_cols[np.abs(bg_shap_values[:,:-1].values).mean(axis=0)>0.000]

rf_cols = rf_features['cols'].values
rf_shap_cols = rf_cols[np.abs(rf_shap_values[:,:-1].values).mean(axis=0)>0.000]
```

[44]:
```python
# Retrain the classifiers using the new feature set
t_clf.fit(train_t[t_shap_cols],y_1)
bg_clf.fit(train_bg[bg_shap_cols],y_1)
rf_clf.fit(train_rf[rf_shap_cols],y_1)
```

[44]:
```
RandomForestClassifier(max_depth=15, min_samples_leaf=800, n_estimators=30)
```

0.1.8 Back test over the period 2010 - 2018

[45]:
```python
scalers = [StandardScaler() for _ in range(len(training_data))]

def get_opt_data(shap_cols):
    opt_training_data = [pd.DataFrame(scalers[i].
 ↪fit_transform(training_frames[i][shap_cols].values),columns=shap_cols) for i␣
 ↪in range(len(training_data))]
    opt_valid_data = [pd.DataFrame(scalers[i].
 ↪transform(valid_frames[i][shap_cols].values),columns=shap_cols) for i in␣
 ↪range(len(valid_data))]
    opt_test_data = [pd.DataFrame(scalers[i].
 ↪transform(test_frames[i][shap_cols].values),columns=shap_cols) for i in␣
 ↪range(len(test_data))]
    return opt_training_data, opt_valid_data, opt_test_data

t_opt_training_data, t_opt_valid_data, t_opt_test_data =␣
 ↪get_opt_data(t_shap_cols)
bg_opt_training_data, bg_opt_valid_data, bg_opt_test_data =␣
 ↪get_opt_data(bg_shap_cols)
rf_opt_training_data, rf_opt_valid_data, rf_opt_test_data =␣
 ↪get_opt_data(rf_shap_cols)
```

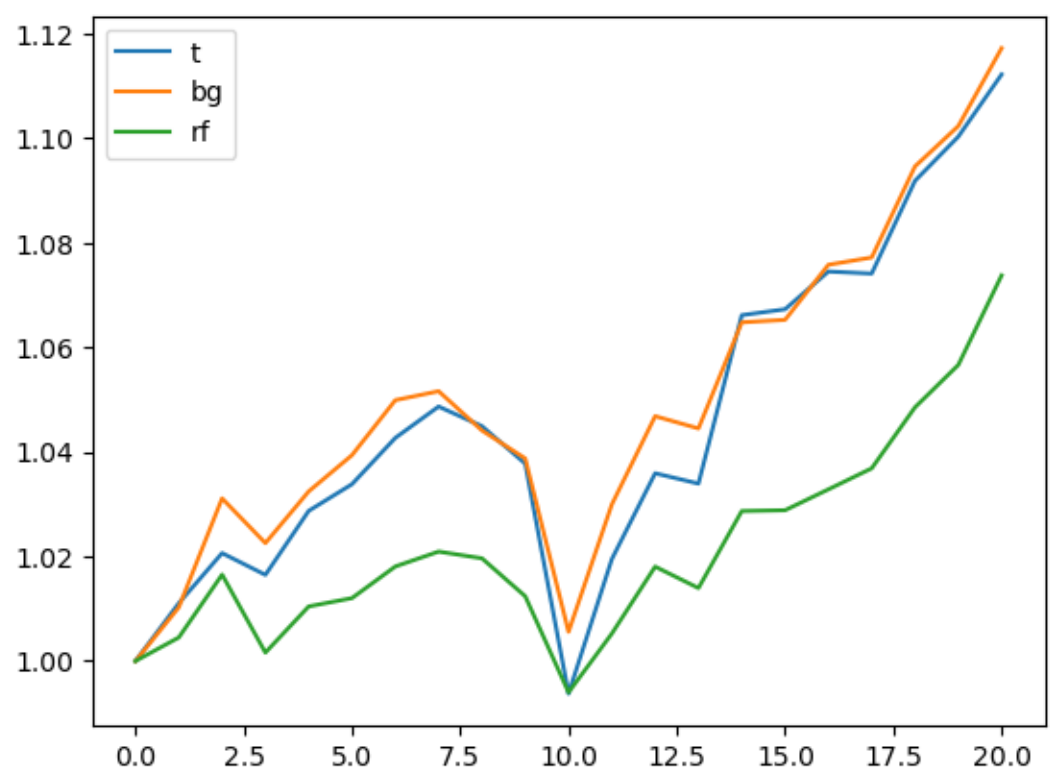
[46]:
```python
# dictionary to store performance and returns
opt_xs = {'t':[1], 'bg':[1], 'rf':[1]}
opt_rets = {'t':[], 'bg':[], 'rf':[]}
models = {'t':t_clf, 'bg':bg_clf, 'rf':rf_clf}

# dictionary of training and test data for each classifier model
opt_training_data = {'t':t_opt_training_data, 'bg':bg_opt_training_data, 'rf':
 ↪rf_opt_training_data}
opt_test_data = {'t':t_opt_test_data, 'bg':bg_opt_test_data, 'rf':
 ↪rf_opt_test_data}

for i in tqdm(range(len(start_dates)-1)):
    for key, model in models.items():
        model.fit(opt_training_data[key][i],training_labels[i])

        preds = model.predict(opt_test_data[key][i])
        profit_i = (preds*test_stock_returns[i]).sum()
```

```
        opt_rets[key].append(profit_i)
        num_names = len(opt_test_data[key][i])
        opt_xs[key].append(opt_xs[key][i] + (opt_xs[key][i]/num_names)*profit_i)
```

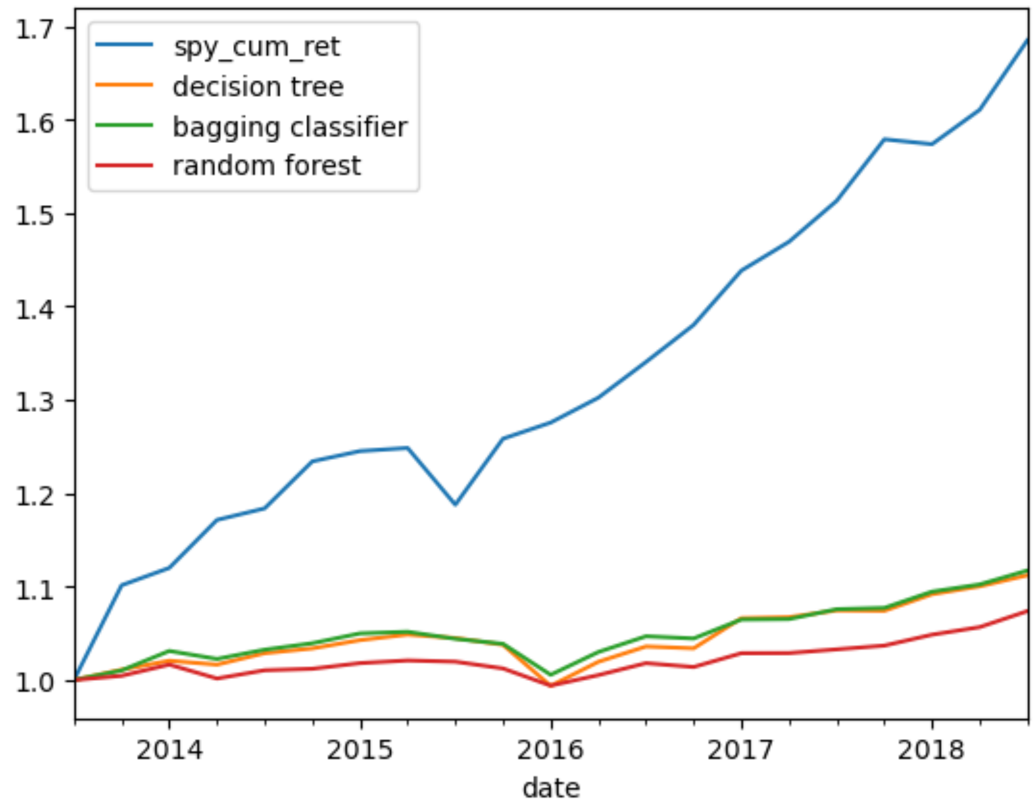
```
100%|                        | 20/20 [00:52<00:00,  2.65s/it]
```

[47]:
```
for key, x_list in opt_xs.items():
    plt.plot(x_list, label = key);
plt.legend()
```

[47]:
```
<matplotlib.legend.Legend at 0x2f5b61550>
```



[48]:
```
# Compare to buy and hold of SPY
SPY = pd.read_pickle(r'SPY_cum_ret.pkl')
SPY = SPY.loc['2013-07-01':'2018-09-30']
SPY = SPY.resample('Q').ffill()
SPY['spy_cum_ret'] = (SPY['spy_cum_ret'] - SPY['spy_cum_ret'][0]+1)
SPY['decision tree'] = opt_xs['t']
SPY['bagging classifier'] = opt_xs['bg']
SPY['random forest'] = opt_xs['rf']
SPY.plot();
```



0.1.9 Compute the Sharpe Ratio, Information Ratio, and alpha for the strategies and for the buy-and-hold strategy for SPY

[49]:
```
strategies = {'decision tree' :      "Decision Tree Strategy     ",
              'bagging classifier': "Bagging Classifier Strategy",
              'random forest' :      "Random Forest Strategy     ",
              'spy_cum_ret':         "SPY Buy-and-hold Strategy  "}
```

[50]:
```
# Sharpe Ratio
for key, strat in strategies.items():
    strategy_mean_ret = (SPY[key] - 1).diff().mean()
    strategy_std = (SPY[key] - 1).diff().std()
    strategy_sr = strategy_mean_ret/strategy_std
    print(strat, ' Sharpe Ratio:', strategy_sr)
```

```
Decision Tree Strategy      Sharpe Ratio: 0.36664541127858863
Bagging Classifier Strategy Sharpe Ratio: 0.44052597772678764
Random Forest Strategy      Sharpe Ratio: 0.39177764778019614
SPY Buy-and-hold Strategy   Sharpe Ratio: 0.9869583355280026
```

[51]:
```python
# Information Ratio
spy_ret = (SPY['spy_cum_ret'] - 1).diff().values[1:]

for key, strat in list(strategies.items())[:-1]:
    strategy_ret = (SPY[key] - 1).diff().values[1:]
    beta = (np.cov(spy_ret,strategy_ret)/np.var(spy_ret))[1,0]
    residual_ret = strategy_ret - beta * spy_ret
    IR = np.mean(residual_ret)/np.std(residual_ret)
    print(strat, ' Information Ratio:', IR)
```

```
Decision Tree Strategy       Information Ratio: 0.24407902834503367
Bagging Classifier Strategy  Information Ratio: 0.2944040595301684
Random Forest Strategy       Information Ratio: 0.3438453383357505
```

[52]:
```python
# Alpha
for key, strat in list(strategies.items())[:-1]:
    strategy_ret = (SPY[key] - 1).diff().values[1:]
    beta = (np.cov(spy_ret,strategy_ret)/np.var(spy_ret))[1,0]
    residual_ret = strategy_ret - beta * spy_ret
    alpha = np.mean(residual_ret)
    print(strat, ' alpha:', alpha)
```

```
Decision Tree Strategy       alpha: 0.003612123226636429
Bagging Classifier Strategy  alpha: 0.003775197633395072
Random Forest Strategy       alpha: 0.003151106466031975
```

HW2 Q2

February 4, 2024

In this notebook, we do the same classifier explorations and strategy comparisons as the previous notebook. However in this case we optimize for the features and the hyperparameters using a new objective function which uses a new definition of profit.

Here the definition of profit will use the predict_proba method from the classifiers on the validation set, from which we can obtain the conviction for each prediction:

$$\text{conviction}_s = \text{Prob}_s(+1)\text{–}\text{Prob}_s(-1)$$

Then

$$\text{weights} = \frac{\text{conviction}_s}{\sum_{s'} |conviction_{s'}|}$$

So

$$\sum_s \text{weights} = 1$$

The profit is then

$$\sum_s \text{weights} * \text{'next\_period\_return'}$$

```
[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
import sklearn
from sklearn.preprocessing import StandardScaler

pd.set_option('use_inf_as_na', True)
from collections import Counter

from tqdm import tqdm # to measure progress of for loops
```

```
/var/folders/sp/wlr6xm297918vx6kjh2z1dk00000gn/T/ipykernel_1879/524871555.py:12:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  pd.set_option('use_inf_as_na', True)
```

[2]:
```
# load dataset
raw_data = pd.read_pickle('dataset.pkl')
data = raw_data.drop([x  for x in raw_data.columns if 'fqtr' in x],axis=1)
# restrict companies to market cap > 1 billion
data = raw_data[raw_data['market_cap'] > 1000.0]
data = data.copy()
# fill in missing values
data.fillna(0.0,inplace=True)
```

0.0.1 Insert a column in dataset based on stock performance

Inserting a column in the dataset where entries are:
- 1 if the stock return is more than 5% higher than the SPY return
- 0 if it is between -10% and +5% relative to the SPY return
- -1 if it is less then -10% relative to the SPY return

[3]:
```
# function to return appropriate values based on performance as detailed above
def f(x):
    if x > 0.05:
        return 1
    elif x < -0.1:
        return -1
    else:
        return 0
```

[4]:
```
# add the new column
data['rel_performance'] = data['pred_rel_return'].apply(f)
# make the date the index
data.reset_index(inplace=True)
data.set_index('date',inplace=True)
```

[5]:
```
data[['pred_rel_return', 'rel_performance']][100:110]
```

[5]:

date	pred_rel_return	rel_performance
2000-04-03	0.261336	1
2000-04-05	-0.337466	-1
2000-04-06	0.272372	1
2000-04-07	-0.645219	-1
2000-04-07	0.192161	1
2000-04-11	0.153474	1
2000-04-11	0.317203	1
2000-04-11	-0.051039	0
2000-04-11	0.028730	0
2000-04-11	0.095712	1

0.0.2 Split the data into training, validation, and test

- Data for training period is from 2007 to 2009 (inclusive, i.e. 3 years)
- Data for validation period is 1 quarter after end of training period
- Data for test period is immediately proceeding the training period

[6]:
```
df_1 = data.loc['2007-01-01':'2010-01-01']
df_valid = data.loc['2010-04-01':'2010-07-01']
df_test = data.loc['2010-07-01':'2010-10-01']
```

[7]:
```
train_1 = df_1.reset_index().drop(['ticker','date',
                                   'next_period_return',
                                   'spy_next_period_return',
                                   'rel_performance','pred_rel_return',
                                  'return', 'cum_ret', 'spy_cum_ret'],axis=1)

valid = df_valid.reset_index().drop(['ticker','date',
                                    'next_period_return',
                                   'spy_next_period_return',
                                   'rel_performance','pred_rel_return',
                                  'return', 'cum_ret', 'spy_cum_ret'],axis=1)

test = df_test.reset_index().drop(['ticker','date',
                                   'next_period_return',
                                   'spy_next_period_return',
                                   'rel_performance','pred_rel_return',
                                  'return', 'cum_ret', 'spy_cum_ret'],axis=1)
```

[8]:
```
# Obtain the y values for each data split
train_1_stock_returns = df_1['next_period_return']
valid_stock_returns = df_valid['next_period_return']
test_stock_returns = df_test['next_period_return']

y_1 = df_1['rel_performance']
y_valid = df_valid['rel_performance']
y_test = df_test['rel_performance']

y_1 = y_1.values
y_valid = y_valid.values
y_test = y_test.values
```

0.0.3 Import Optuna to find the optimal hyperparameters for the classifiers

[9]:
```
import optuna
from optuna.trial import Trial
# optuna.logging.set_verbosity(optuna.logging.FATAL)
import warnings
warnings.filterwarnings("ignore")
```

0.1 Defining the Optuna objective function for our 3 classifiers:

- DecisionTree classifier
- Bagging classifier
- RandomForest classifier

0.1.1 Note now that we are optimizng for the new definitoin of the profit using the 'conviction' of the model.

[10]:
```
def objective_tree(trial:
↪Trial,train=None,labels=None,val=None,val_labels=None,val_rets=None):

    t_min_samples_leaf = trial.suggest_int('min_samples_leaf',100,1200,step=200)
    t_max_depth = trial.suggest_int('max_depth',5,25,step=5)

    t_clf = DecisionTreeClassifier(min_samples_leaf =␣
↪t_min_samples_leaf,max_depth=t_max_depth,random_state=123)
    t_clf.fit(train,labels)

    # Predicting probabilities
    probs = t_clf.predict_proba(val)

    # Conviction for each stock
    conviction = probs[:, 2] - probs[:, 0]  # Prob_s(+1) - Prob_s(-1)
    weights = conviction / np.sum(np.abs(conviction))

    # Calculating profit
    profit = np.sum(weights * val_rets)

    return profit
```

[11]:
```
def objective_bagging(trial:
↪Trial,train=None,labels=None,val=None,val_labels=None,val_rets=None):

    t_min_samples_leaf = trial.suggest_int('min_samples_leaf',100,1200,step=200)
    t_max_depth = trial.suggest_int('max_depth',5,25,step=5)
    t_n_estimators = trial.suggest_int('n_estimators',5,50,step=5)

    tree_clf = DecisionTreeClassifier(min_samples_leaf =␣
↪t_min_samples_leaf,max_depth=t_max_depth,random_state=123)
    bg_clf =␣
↪BaggingClassifier(tree_clf,n_estimators=t_n_estimators,random_state=123,n_jobs=1)
    bg_clf.fit(train,labels)

    # Predicting probabilities
    probs = bg_clf.predict_proba(val)

    # Conviction for each stock
```

```
    conviction = probs[:, 2] - probs[:, 0]  # Prob_s(+1) - Prob_s(-1)
    weights = conviction / np.sum(np.abs(conviction))

    # Calculating profit
    profit = np.sum(weights * val_rets)

    return profit
```

[12]:
```
def objective_rf(trial:
 ↪Trial,train=None,labels=None,val=None,val_labels=None,val_rets=None):

    rf_n_estimators = trial.suggest_int('n_estimators', 10,40,step=5)
    rf_max_features = trial.suggest_categorical('max_features',['sqrt','log2'])
    rf_min_samples_leaf = trial.
 ↪suggest_int('min_samples_leaf',800,2400,step=800)
    rf_max_depth = trial.suggest_int('max_depth',4,15)

    rf_clf = RandomForestClassifier(n_estimators=rf_n_estimators,
                                    max_depth=rf_max_depth,
                                    min_samples_leaf=rf_min_samples_leaf,
                                   ␣
 ↪max_features=rf_max_features,random_state=123)
    rf_clf.fit(train,labels)

    # Predicting probabilities
    probs = rf_clf.predict_proba(val)

    # Conviction for each stock
    conviction = probs[:, 2] - probs[:, 0]  # Prob_s(+1) - Prob_s(-1)
    weights = conviction / np.sum(np.abs(conviction))

    # Calculating profit
    profit = np.sum(weights * val_rets)

    return profit
```

[13]:
```
study_tree = optuna.create_study(direction="maximize")
study_bagging = optuna.create_study(direction="maximize")
study_rf = optuna.create_study(direction="maximize")
```

```
[I 2024-02-04 19:49:52,271] A new study created in memory with name: no-
name-99061f9f-bc79-4399-8d59-724c220fdb93
[I 2024-02-04 19:49:52,272] A new study created in memory with name: no-
name-2d115b25-6ec6-42d9-bca8-59743d956cf0
[I 2024-02-04 19:49:52,272] A new study created in memory with name: no-
name-43cb0cbd-7ad8-4643-b14c-cc6f7228ffe3
```

0.1.2 Run optimizations to find optimal parameters

[14]:
```python
from functools import partial
```

[15]:
```python
%%time
study_tree.
  ↪optimize(partial(objective_tree,train=train_1,labels=y_1,val=valid,val_labels=y_valid,val_r
  ↪n_trials=200,n_jobs=-1)
```

```
[I 2024-02-04 19:49:53,297] Trial 9 finished with value: -0.03507104894488454
and parameters: {'min_samples_leaf': 1100, 'max_depth': 25}. Best is trial 9
with value: -0.03507104894488454.
[I 2024-02-04 19:49:53,496] Trial 8 finished with value: -0.03507104894488454
and parameters: {'min_samples_leaf': 1100, 'max_depth': 15}. Best is trial 9
with value: -0.03507104894488454.
[I 2024-02-04 19:49:53,603] Trial 0 finished with value: -0.03170653629246879
and parameters: {'min_samples_leaf': 700, 'max_depth': 15}. Best is trial 0 with
value: -0.03170653629246879.
[I 2024-02-04 19:49:53,609] Trial 6 finished with value: -0.03507104894488454
and parameters: {'min_samples_leaf': 1100, 'max_depth': 10}. Best is trial 0
with value: -0.03170653629246879.
[I 2024-02-04 19:49:53,676] Trial 1 finished with value: -0.03618642324470992
and parameters: {'min_samples_leaf': 900, 'max_depth': 25}. Best is trial 0 with
value: -0.03170653629246879.
[I 2024-02-04 19:49:53,679] Trial 2 finished with value: -0.030271772656712496
and parameters: {'min_samples_leaf': 500, 'max_depth': 15}. Best is trial 2 with
value: -0.030271772656712496.
[I 2024-02-04 19:49:53,759] Trial 5 finished with value: -0.034943347794546935
and parameters: {'min_samples_leaf': 300, 'max_depth': 5}. Best is trial 2 with
value: -0.030271772656712496.
[I 2024-02-04 19:49:53,818] Trial 4 finished with value: -0.030271772656712496
and parameters: {'min_samples_leaf': 500, 'max_depth': 20}. Best is trial 2 with
value: -0.030271772656712496.
[I 2024-02-04 19:49:53,930] Trial 7 finished with value: -0.033228938079633744
and parameters: {'min_samples_leaf': 300, 'max_depth': 10}. Best is trial 2 with
value: -0.030271772656712496.
[I 2024-02-04 19:49:54,488] Trial 3 finished with value: -0.02944046269221485
and parameters: {'min_samples_leaf': 100, 'max_depth': 10}. Best is trial 3 with
value: -0.02944046269221485.
[I 2024-02-04 19:49:54,612] Trial 12 finished with value: -0.03507104894488454
and parameters: {'min_samples_leaf': 1100, 'max_depth': 25}. Best is trial 3
with value: -0.02944046269221485.
[I 2024-02-04 19:49:54,742] Trial 11 finished with value: -0.030271772656712496
and parameters: {'min_samples_leaf': 500, 'max_depth': 15}. Best is trial 3 with
value: -0.02944046269221485.
[I 2024-02-04 19:49:54,780] Trial 13 finished with value: -0.03507104894488454
and parameters: {'min_samples_leaf': 1100, 'max_depth': 5}. Best is trial 3 with
value: -0.02944046269221485.
```

```
[I 2024-02-04 19:50:30,248] Trial 188 finished with value: -0.029139837603797823
and parameters: {'min_samples_leaf': 100, 'max_depth': 15}. Best is trial 10
with value: -0.029139837603797823.
[I 2024-02-04 19:50:30,303] Trial 187 finished with value: -0.029139837603797823
and parameters: {'min_samples_leaf': 100, 'max_depth': 20}. Best is trial 10
with value: -0.029139837603797823.
[I 2024-02-04 19:50:30,433] Trial 191 finished with value: -0.029139837603797823
and parameters: {'min_samples_leaf': 100, 'max_depth': 25}. Best is trial 10
with value: -0.029139837603797823.
[I 2024-02-04 19:50:30,598] Trial 192 finished with value: -0.029139837603797823
and parameters: {'min_samples_leaf': 100, 'max_depth': 25}. Best is trial 10
with value: -0.029139837603797823.
[I 2024-02-04 19:50:31,405] Trial 193 finished with value: -0.029139837603797823
and parameters: {'min_samples_leaf': 100, 'max_depth': 25}. Best is trial 10
with value: -0.029139837603797823.
[I 2024-02-04 19:50:31,701] Trial 194 finished with value: -0.029139837603797823
and parameters: {'min_samples_leaf': 100, 'max_depth': 25}. Best is trial 10
with value: -0.029139837603797823.
[I 2024-02-04 19:50:31,909] Trial 195 finished with value: -0.029139837603797823
and parameters: {'min_samples_leaf': 100, 'max_depth': 25}. Best is trial 10
with value: -0.029139837603797823.
[I 2024-02-04 19:50:31,971] Trial 197 finished with value: -0.029139837603797823
and parameters: {'min_samples_leaf': 100, 'max_depth': 25}. Best is trial 10
with value: -0.029139837603797823.
[I 2024-02-04 19:50:31,979] Trial 198 finished with value: -0.029139837603797823
and parameters: {'min_samples_leaf': 100, 'max_depth': 25}. Best is trial 10
with value: -0.029139837603797823.
[I 2024-02-04 19:50:32,035] Trial 196 finished with value: -0.029139837603797823
and parameters: {'min_samples_leaf': 100, 'max_depth': 25}. Best is trial 10
with value: -0.029139837603797823.
[I 2024-02-04 19:50:32,036] Trial 199 finished with value: -0.029139837603797823
and parameters: {'min_samples_leaf': 100, 'max_depth': 25}. Best is trial 10
with value: -0.029139837603797823.

CPU times: user 5min 49s, sys: 2.93 s, total: 5min 52s
Wall time: 39.8 s
```

[16]:

```
%%time
study_bagging.
 ↪optimize(partial(objective_bagging,train=train_1,labels=y_1,val=valid,val_labels=y_valid,va
 ↪n_trials=200,n_jobs=-1)
```

```
[I 2024-02-04 19:50:37,819] Trial 8 finished with value: -0.03513729520240121
and parameters: {'min_samples_leaf': 900, 'max_depth': 10, 'n_estimators': 10}.
Best is trial 8 with value: -0.03513729520240121.
[I 2024-02-04 19:50:42,062] Trial 2 finished with value: -0.03435772493864116
and parameters: {'min_samples_leaf': 700, 'max_depth': 20, 'n_estimators': 15}.
Best is trial 2 with value: -0.03435772493864116.
[I 2024-02-04 19:50:49,965] Trial 11 finished with value: -0.034797389358008066
```

```
and parameters: {'min_samples_leaf': 100, 'max_depth': 15, 'n_estimators': 10}.
Best is trial 17 with value: -0.03265847612796275.
[I 2024-02-04 19:56:59,347] Trial 194 finished with value: -0.03265847612796275
and parameters: {'min_samples_leaf': 100, 'max_depth': 15, 'n_estimators': 10}.
Best is trial 17 with value: -0.03265847612796275.
[I 2024-02-04 19:57:00,296] Trial 196 finished with value: -0.03265847612796275
and parameters: {'min_samples_leaf': 100, 'max_depth': 15, 'n_estimators': 10}.
Best is trial 17 with value: -0.03265847612796275.
[I 2024-02-04 19:57:00,407] Trial 195 finished with value: -0.03265847612796275
and parameters: {'min_samples_leaf': 100, 'max_depth': 15, 'n_estimators': 10}.
Best is trial 17 with value: -0.03265847612796275.
[I 2024-02-04 19:57:01,518] Trial 198 finished with value: -0.03265847612796275
and parameters: {'min_samples_leaf': 100, 'max_depth': 15, 'n_estimators': 10}.
Best is trial 17 with value: -0.03265847612796275.
[I 2024-02-04 19:57:01,682] Trial 199 finished with value: -0.03265847612796275
and parameters: {'min_samples_leaf': 100, 'max_depth': 15, 'n_estimators': 10}.
Best is trial 17 with value: -0.03265847612796275.

CPU times: user 59min 37s, sys: 19.6 s, total: 59min 57s
Wall time: 6min 29s
```

[17]:
```
%%time
study_rf.
 ↪optimize(partial(objective_rf,train=train_1,labels=y_1,val=valid,val_labels=y_valid,val_ret
 ↪n_trials=200,n_jobs=-1)
```

```
[I 2024-02-04 19:57:01,897] Trial 0 finished with value: -0.033294890464413515
and parameters: {'n_estimators': 10, 'max_features': 'log2', 'min_samples_leaf':
800, 'max_depth': 6}. Best is trial 0 with value: -0.033294890464413515.
[I 2024-02-04 19:57:01,983] Trial 7 finished with value: -0.03312109365778519
and parameters: {'n_estimators': 25, 'max_features': 'log2', 'min_samples_leaf':
2400, 'max_depth': 9}. Best is trial 7 with value: -0.03312109365778519.
[I 2024-02-04 19:57:02,077] Trial 6 finished with value: -0.033087587056024326
and parameters: {'n_estimators': 35, 'max_features': 'log2', 'min_samples_leaf':
2400, 'max_depth': 13}. Best is trial 6 with value: -0.033087587056024326.
[I 2024-02-04 19:57:02,089] Trial 10 finished with value: -0.03305251857272103
and parameters: {'n_estimators': 15, 'max_features': 'log2', 'min_samples_leaf':
1600, 'max_depth': 6}. Best is trial 10 with value: -0.03305251857272103.
[I 2024-02-04 19:57:02,220] Trial 1 finished with value: -0.03352655896226559
and parameters: {'n_estimators': 20, 'max_features': 'sqrt', 'min_samples_leaf':
1600, 'max_depth': 8}. Best is trial 10 with value: -0.03305251857272103.
[I 2024-02-04 19:57:02,221] Trial 2 finished with value: -0.03352655896226559
and parameters: {'n_estimators': 20, 'max_features': 'sqrt', 'min_samples_leaf':
1600, 'max_depth': 15}. Best is trial 10 with value: -0.03305251857272103.
[I 2024-02-04 19:57:02,348] Trial 11 finished with value: -0.033087587056024326
and parameters: {'n_estimators': 35, 'max_features': 'log2', 'min_samples_leaf':
2400, 'max_depth': 14}. Best is trial 10 with value: -0.03305251857272103.
[I 2024-02-04 19:57:02,406] Trial 13 finished with value: -0.032997758066637024
and parameters: {'n_estimators': 30, 'max_features': 'log2', 'min_samples_leaf':
```

800, 'max_depth': 14}. Best is trial 68 with value: -0.03292388157772522.

CPU times: user 52.6 s, sys: 6.33 s, total: 58.9 s
Wall time: 11.4 s

0.1.3 Instantiate the classifiers with the best parameters

[18]:
```
print("Best parameters for decision tree: ", study_tree.best_params)
print("Best parameters for bagging: ", study_bagging.best_params)
print("Best parameters for random forest: ", study_rf.best_params)
```

```
Best parameters for decision tree:  {'min_samples_leaf': 100, 'max_depth': 25}
Best parameters for bagging:  {'min_samples_leaf': 100, 'max_depth': 25,
'n_estimators': 10}
Best parameters for random forest:  {'n_estimators': 20, 'max_features': 'log2',
'min_samples_leaf': 800, 'max_depth': 14}
```

[19]:
```
t_clf = DecisionTreeClassifier(**study_tree.best_params,random_state=123)
```

[20]:
```
tree_cfl = DecisionTreeClassifier(**{'min_samples_leaf': study_bagging.
 ↪best_params['min_samples_leaf'],
                                     'max_depth': study_bagging.
 ↪best_params['max_depth']})
bg_clf = BaggingClassifier(tree_cfl,n_estimators=study_bagging.
 ↪best_params['n_estimators'],random_state=123 ,n_jobs=-1)
```

[21]:
```
rf_clf = RandomForestClassifier(**study_rf.best_params)
```

0.1.4 Train each of the classifiers

[22]:
```
t_clf.fit(train_1,y_1)
```

[22]:
```
DecisionTreeClassifier(max_depth=25, min_samples_leaf=100, random_state=123)
```

[23]:
```
bg_clf.fit(train_1,y_1)
```

[23]:
```
BaggingClassifier(estimator=DecisionTreeClassifier(max_depth=25,
                                                   min_samples_leaf=100),
                  n_jobs=-1, random_state=123)
```

[24]:
```
rf_clf.fit(train_1,y_1)
```

[24]:
```
RandomForestClassifier(max_depth=14, max_features='log2', min_samples_leaf=800,
                       n_estimators=20)
```

0.1.5 Finding the features with non zero Shapley values

[25]:
```
# Retrain the models using the original training set (i.e. before backtesting)
t_clf.fit(train_1,y_1)
bg_clf.fit(train_1,y_1)
rf_clf.fit(train_1,y_1)
```

[25]:
```
RandomForestClassifier(max_depth=14, max_features='log2', min_samples_leaf=800,
                       n_estimators=20)
```

[26]:
```
# Obtain the feature importances
def tree_feat_importance(m, df):
    return pd.DataFrame({'cols':df.columns, 'feat_imp':m.feature_importances_}
                       ).sort_values('feat_imp', ascending=False)

def bagging_feat_importance(m, df):
    feature_importances = []
    for est in m.estimators_:
        fi = est.feature_importances_
        feature_importances.append(fi)
    feature_importances = np.array(feature_importances)

    return pd.DataFrame({'cols':train_1.columns, 'feat_imp':np.
 ↪mean(feature_importances,axis=0)}
                       ).sort_values('feat_imp', ascending=False)

def randomforest_feat_importances(m, df):

    return pd.DataFrame({'cols':df.columns, 'feat_imp': m.feature_importances_}
                       ).sort_values('feat_imp', ascending=False)

def plot_fi(fi): return fi.plot('cols', 'feat_imp', 'barh', figsize=(12,7),␣
 ↪legend=False)
```

[27]:
```
t_fi = tree_feat_importance(t_clf,train_1)
bg_fi = bagging_feat_importance(bg_clf,train_1)
rf_fi = randomforest_feat_importances(rf_clf,train_1)
```

[28]:
```
# Only use features that have positive feature importance
t_features = t_fi[(t_fi['feat_imp'] > 0.00)]
bg_features = bg_fi[(bg_fi['feat_imp'] > 0.00)]
rf_features = rf_fi[(rf_fi['feat_imp'] > 0.00)]
```

[29]:
```
train_t = train_1[t_features['cols'].values]
valid_t = valid[t_features['cols'].values]
valid_t['returns'] = valid_stock_returns.values

train_bg = train_1[bg_features['cols'].values]
```

```
valid_bg = valid[bg_features['cols'].values]
valid_bg['returns'] = valid_stock_returns.values

train_rf = train_1[rf_features['cols'].values]
valid_rf = valid[rf_features['cols'].values]
valid_rf['returns'] = valid_stock_returns.values
```

[30]:
```
print(f"Number of features used for decision tree classifier reduced from:␣
 ↪{train_1.shape[1]} to {train_t.shape[1]}")
print(f"Number of features used for bagging classifier reduced from      :␣
 ↪{train_1.shape[1]} to {train_bg.shape[1]}")
print(f"Number of features used for random forest classifier reduced from:␣
 ↪{train_1.shape[1]} to {train_rf.shape[1]}")
```

```
Number of features used for decision tree classifier reduced from: 725 to 84
Number of features used for bagging classifier reduced from      : 725 to 169
Number of features used for random forest classifier reduced from: 725 to 87
```

[31]:
```
# Retrain the classifiers using the new feature set
t_clf.fit(train_t,y_1)
bg_clf.fit(train_bg,y_1)
rf_clf.fit(train_rf,y_1)
```

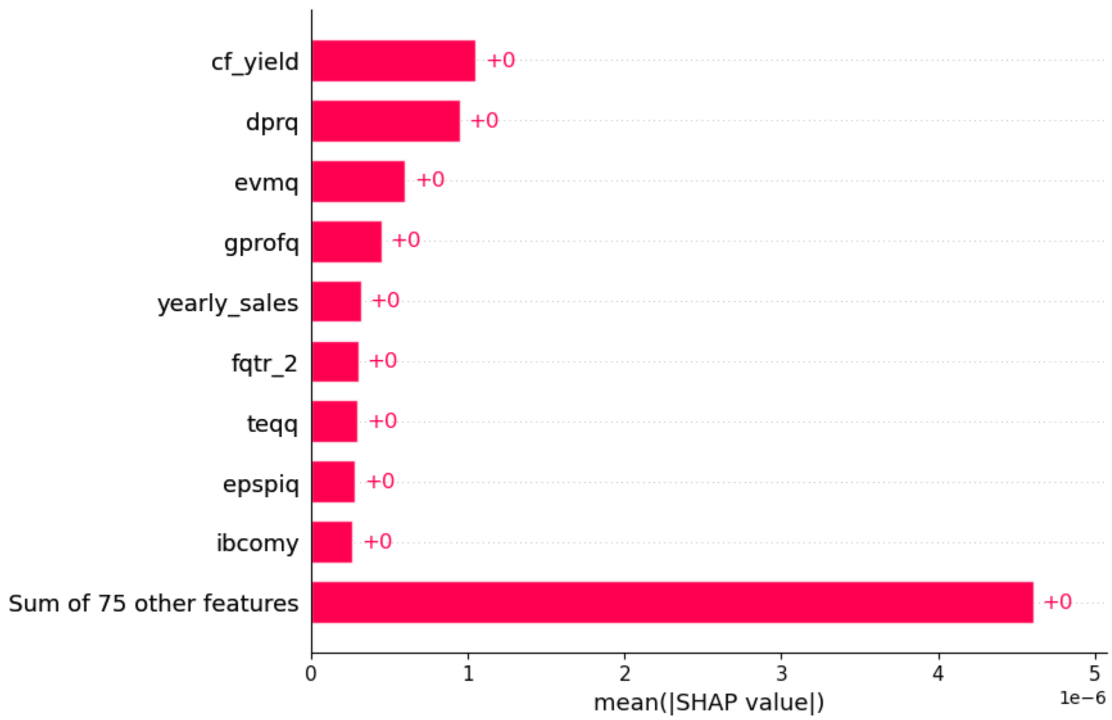
[31]:
```
RandomForestClassifier(max_depth=14, max_features='log2', min_samples_leaf=800,
                       n_estimators=20)
```

[32]:
```
import shap

def model_t(features):
    tree_features = features[features.columns[:-1].values]
    # Use predict_proba and the conviction definition of profit
    probs = t_clf.predict_proba(tree_features)
    conviction = probs[:, 2] - probs[:, 0]  # Prob_s(+1) - Prob_s(-1)
    weights = conviction / np.sum(np.abs(conviction))
    ret = weights * features[features.columns[-1]]
    return ret

def model_bg(features):
    bagging_features = features[features.columns[:-1].values]
    # Use predict_proba and the conviction definition of profit
    probs = bg_clf.predict_proba(bagging_features)
    conviction = probs[:, 2] - probs[:, 0]  # Prob_s(+1) - Prob_s(-1)
    weights = conviction / np.sum(np.abs(conviction))
    ret = weights * features[features.columns[-1]]
    return ret

def model_rf(features):
    rf_features = features[features.columns[:-1].values]
```

```
    probs = rf_clf.predict_proba(rf_features)
    conviction = probs[:, 2] - probs[:, 0]  # Prob_s(+1) - Prob_s(-1)
    weights = conviction / np.sum(np.abs(conviction))
    ret = weights * features[features.columns[-1]]
    return ret
```

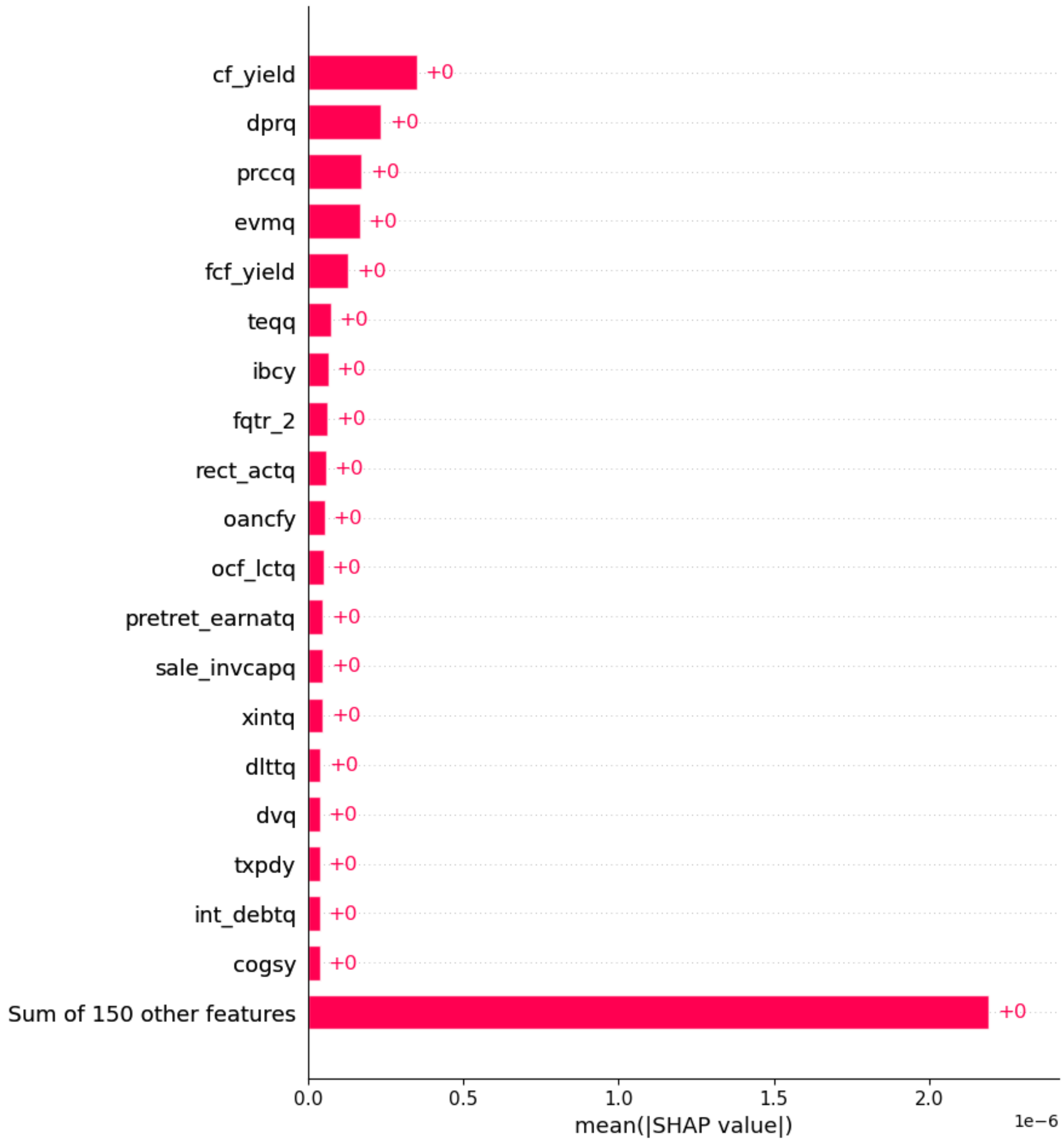
[33]:
```
# Shapley for tree classifier
model_t(valid_t)
t_explainer = shap.explainers.Permutation(model_t,valid_t)
t_shap_values = t_explainer(valid_t,max_evals=2000)
shap.plots.bar(t_shap_values[:,:-1],max_display=10)
```

```
PermutationExplainer explainer: 1442it [01:42, 12.76it/s]
```



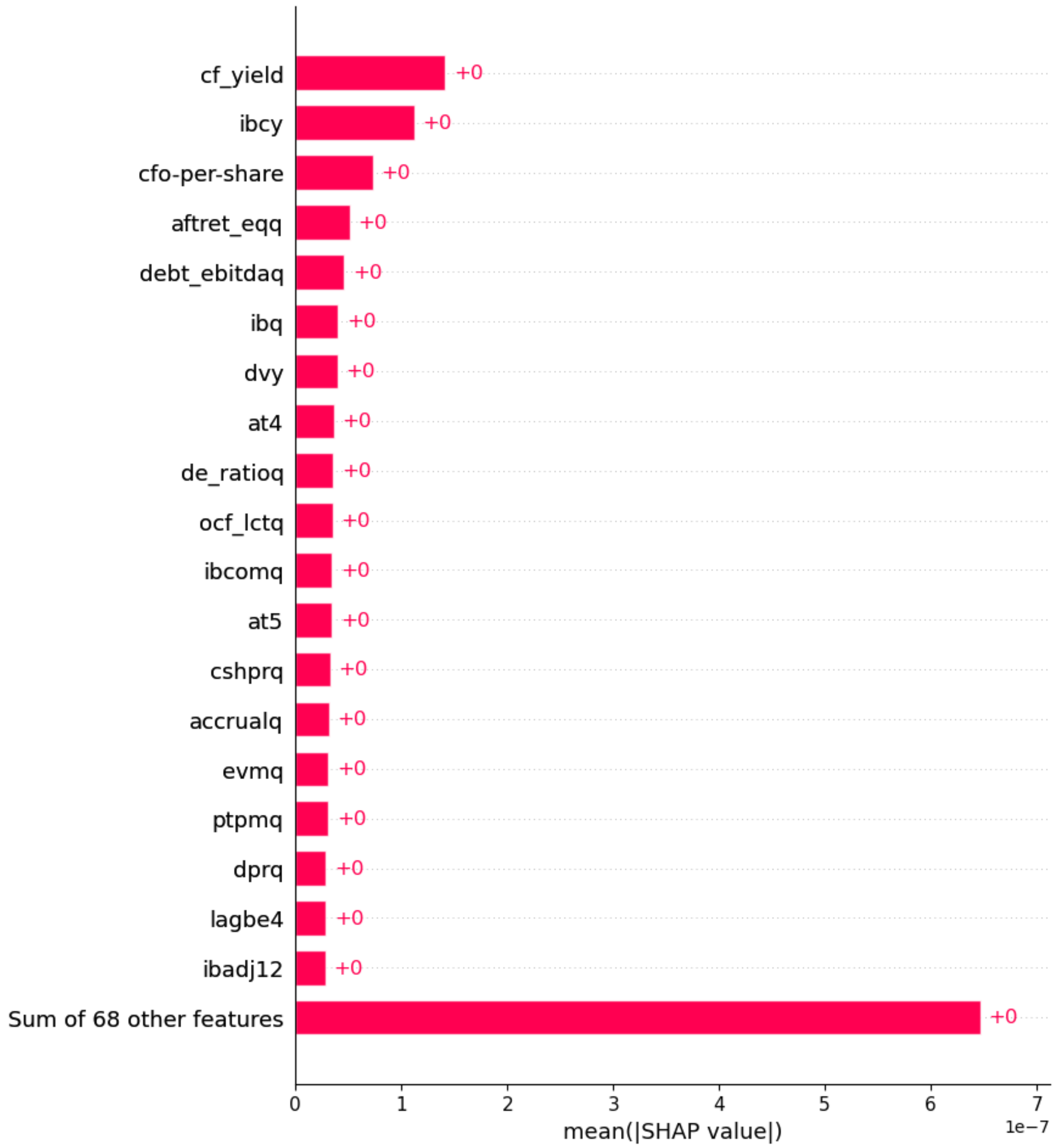
[34]:
```
# Shapley for bagging classifier
model_bg(valid_bg)
bg_explainer = shap.explainers.Permutation(model_bg,valid_bg)
bg_shap_values = bg_explainer(valid_bg,max_evals=2000)
shap.plots.bar(bg_shap_values[:,:-1],max_display=20)
```

```
PermutationExplainer explainer: 1442it [31:31,  1.32s/it]
```



[35]:
```
# Shapley for random forest classifier
model_rf(valid_rf)
rf_explainer = shap.explainers.Permutation(model_rf,valid_rf)
rf_shap_values = rf_explainer(valid_rf,max_evals=2000)
shap.plots.bar(rf_shap_values[:,:-1],max_display=20)
```

```
PermutationExplainer explainer: 1442it [03:47,  6.06it/s]
```



0.1.6 Retrain the models with the features that have non-zero Shapley values

[36]:

```
t_cols = t_features['cols'].values
t_shap_cols = t_cols[np.abs(t_shap_values[:,:-1].values).mean(axis=0)>0.000]

bg_cols = bg_features['cols'].values
bg_shap_cols = bg_cols[np.abs(bg_shap_values[:,:-1].values).mean(axis=0)>0.000]

rf_cols = rf_features['cols'].values
rf_shap_cols = rf_cols[np.abs(rf_shap_values[:,:-1].values).mean(axis=0)>0.000]
```

[37]:
```
t_clf.fit(train_t[t_shap_cols],y_1)
bg_clf.fit(train_bg[bg_shap_cols],y_1)
rf_clf.fit(train_rf[rf_shap_cols],y_1)
```

[37]:
```
RandomForestClassifier(max_depth=14, max_features='log2', min_samples_leaf=800,
                       n_estimators=20)
```

0.1.7 Back test over the period 2010 - 2018

[38]:
```
start_dates = [pd.to_datetime('2010-01-01') + pd.DateOffset(months = 3*i) for i␣
 ↪in range(21)]
end_dates = [d + pd.DateOffset(months = 36) for d in start_dates]
# So the first period is [2010 Jan 1 - 2013 Jan 1], and the last period is␣
 ↪[2015 Jan 1 - 2018 Jan 1]
```

[39]:
```
training_frames = [data.loc[d:d+pd.DateOffset(months = 36)] for d in␣
 ↪start_dates]
valid_frames = [data.loc[d + pd.DateOffset(months=3):d+pd.DateOffset(months =␣
 ↪6)] for d in end_dates]
test_frames = [data.loc[d + pd.DateOffset(months=6):d+pd.DateOffset(months =␣
 ↪9)] for d in end_dates]

training_labels = [d['rel_performance'].values for d in training_frames]

training_stock_returns = [d['next_period_return'].values for d in␣
 ↪training_frames]
test_stock_returns = [d['next_period_return'] for d in test_frames]
```

[40]:
```
training_data = [df.reset_index().drop(['ticker','date',
                                   'next_period_return',
                                   'spy_next_period_return',
                                   'rel_performance', 'pred_rel_return',
                                  'return', 'cum_ret', 'spy_cum_ret'],axis=1)␣
 ↪for df in training_frames]
valid_data = [df.reset_index().drop(['ticker','date',
                                   'next_period_return',
                                   'spy_next_period_return',
                                   'rel_performance','pred_rel_return',
                                  'return', 'cum_ret', 'spy_cum_ret'],axis=1)␣
 ↪for df in valid_frames]
test_data = [df.reset_index().drop(['ticker','date',
                                   'next_period_return',
                                   'spy_next_period_return',
                                   'rel_performance','pred_rel_return',
                                  'return', 'cum_ret', 'spy_cum_ret'],axis=1)␣
 ↪for df in test_frames]
```

[41]:
```
scalers = [StandardScaler() for _ in range(len(training_data))]

def get_opt_data(shap_cols):
    opt_training_data = [pd.DataFrame(scalers[i].
 ↪fit_transform(training_frames[i][shap_cols].values),columns=shap_cols) for i␣
 ↪in range(len(training_data))]
    opt_valid_data = [pd.DataFrame(scalers[i].
 ↪transform(valid_frames[i][shap_cols].values),columns=shap_cols) for i in␣
 ↪range(len(valid_data))]
    opt_test_data = [pd.DataFrame(scalers[i].
 ↪transform(test_frames[i][shap_cols].values),columns=shap_cols) for i in␣
 ↪range(len(test_data))]
    return opt_training_data, opt_valid_data, opt_test_data

t_opt_training_data, t_opt_valid_data, t_opt_test_data =␣
 ↪get_opt_data(t_shap_cols)
bg_opt_training_data, bg_opt_valid_data, bg_opt_test_data =␣
 ↪get_opt_data(bg_shap_cols)
rf_opt_training_data, rf_opt_valid_data, rf_opt_test_data =␣
 ↪get_opt_data(rf_shap_cols)
```

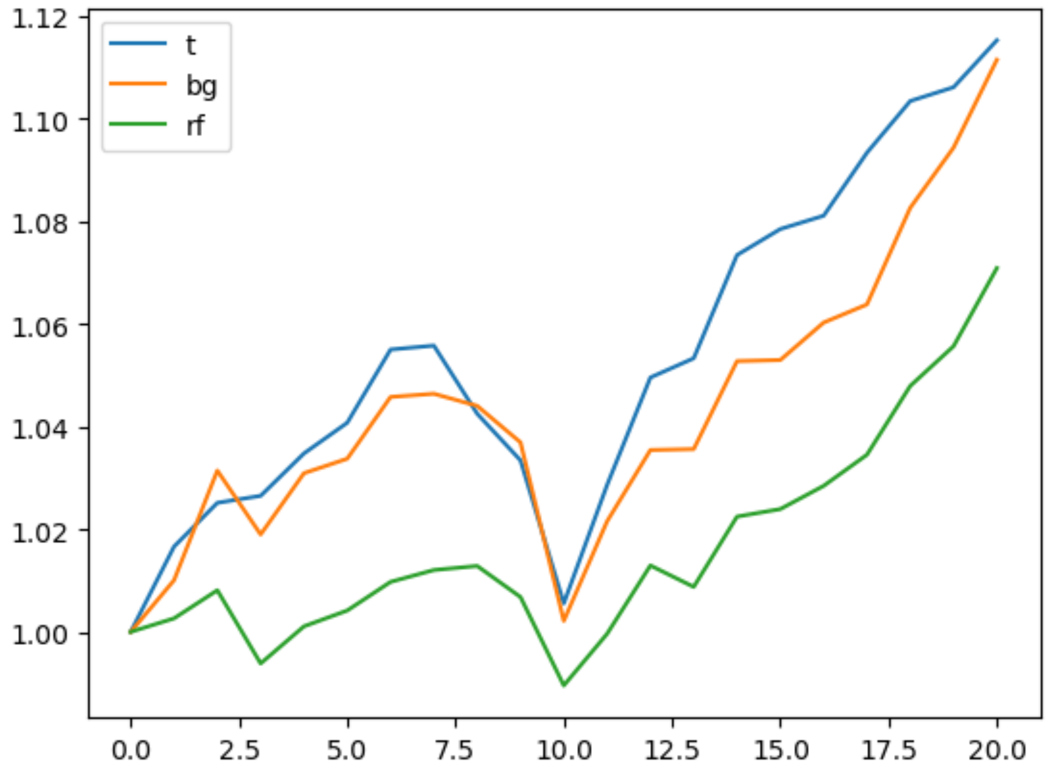
[42]:
```
# dictionary to store performance and returns
opt_xs = {'t':[1], 'bg':[1], 'rf':[1]}
opt_rets = {'t':[], 'bg':[], 'rf':[]}
models = {'t':t_clf, 'bg':bg_clf, 'rf':rf_clf}

# dictionary of training and test data for each classifier model
opt_training_data = {'t':t_opt_training_data, 'bg':bg_opt_training_data, 'rf':
 ↪rf_opt_training_data}
opt_test_data = {'t':t_opt_test_data, 'bg':bg_opt_test_data, 'rf':
 ↪rf_opt_test_data}

for i in tqdm(range(len(start_dates)-1)):
    for key, model in models.items():
        model.fit(opt_training_data[key][i],training_labels[i])

        preds = model.predict(opt_test_data[key][i])
        profit_i = (preds*test_stock_returns[i]).sum()

        opt_rets[key].append(profit_i)
        num_names = len(opt_test_data[key][i])
        opt_xs[key].append(opt_xs[key][i] + (opt_xs[key][i]/num_names)*profit_i)
```

```
100%|                          | 20/20 [01:12<00:00,  3.62s/it]
```

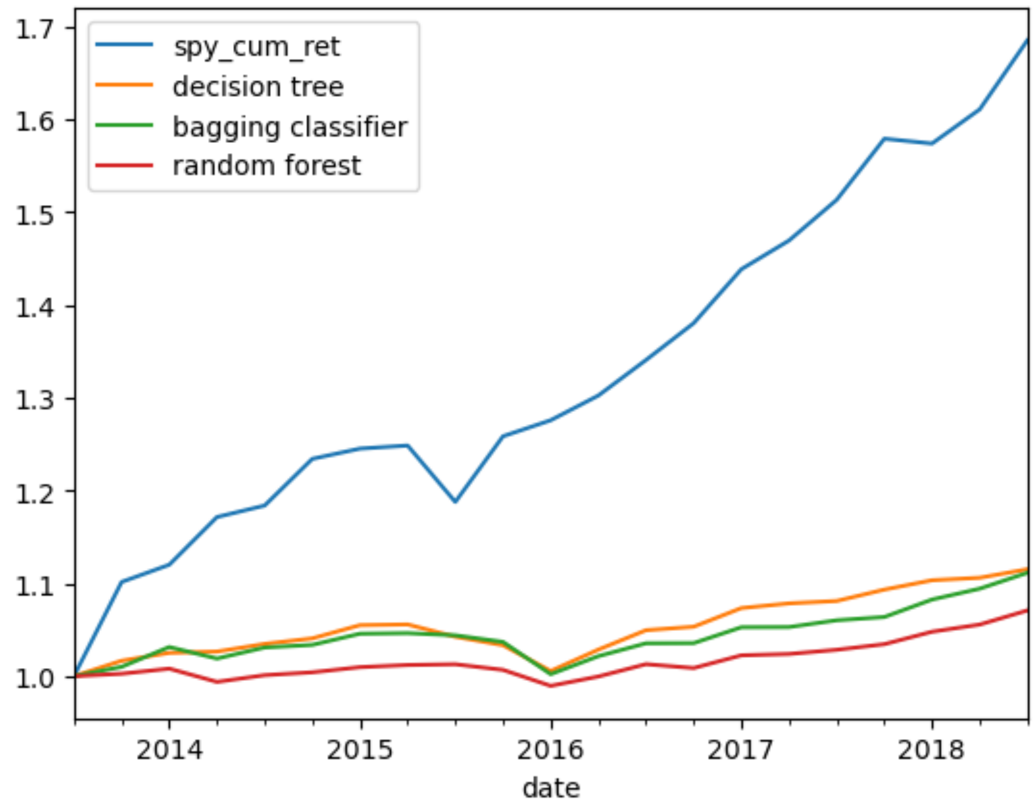
[54]:
```
for key, x_list in opt_xs.items():
    plt.plot(x_list, label = key);
```

```
plt.legend()
```

[54]: <matplotlib.legend.Legend at 0x2e55a7e10>



[55]:
```
# Compare to buy and hold of SPY
SPY = pd.read_pickle(r'SPY_cum_ret.pkl')
SPY = SPY.loc['2013-07-01':'2018-09-30']
SPY = SPY.resample('Q').ffill()
SPY['spy_cum_ret'] = (SPY['spy_cum_ret'] - SPY['spy_cum_ret'][0]+1)
SPY['decision tree'] = opt_xs['t']
SPY['bagging classifier'] = opt_xs['bg']
SPY['random forest'] = opt_xs['rf']
SPY.plot();
```



0.1.8 Compute the Sharpe Ratio, Information Ratio, and alpha for the strategies and for the buy-and-hold strategy for SPY

[56]:
```
strategies = {'decision tree' :      "Decision Tree Strategy     ",
              'bagging classifier': "Bagging Classifier Strategy",
              'random forest' :      "Random Forest Strategy     ",
              'spy_cum_ret':         "SPY Buy-and-hold Strategy  "}
```

[57]:
```
# Sharpe Ratio
for key, strat in strategies.items():
    strategy_mean_ret = (SPY[key] - 1).diff().mean()
    strategy_std = (SPY[key] - 1).diff().std()
    strategy_sr = strategy_mean_ret/strategy_std
    print(strat, ' Sharpe Ratio:', strategy_sr)
```

```
Decision Tree Strategy       Sharpe Ratio: 0.47523455895031175
Bagging Classifier Strategy  Sharpe Ratio: 0.4187276875359691
Random Forest Strategy       Sharpe Ratio: 0.4076851619586382
SPY Buy-and-hold Strategy    Sharpe Ratio: 0.9869583355280026
```

[58]:
```python
# Information Ratio
spy_ret = (SPY['spy_cum_ret'] - 1).diff().values[1:]

for key, strat in list(strategies.items())[:-1]:
    strategy_ret = (SPY[key] - 1).diff().values[1:]
    beta = (np.cov(spy_ret,strategy_ret)/np.var(spy_ret))[1,0]
    residual_ret = strategy_ret - beta * spy_ret
    IR = np.mean(residual_ret)/np.std(residual_ret)
    print(strat, ' Information Ratio:', IR)
```

```
Decision Tree Strategy       Information Ratio: 0.12851206603115614
Bagging Classifier Strategy  Information Ratio: 0.35241553371504525
Random Forest Strategy       Information Ratio: 0.3910570649408755
```

[59]:
```python
# Alpha
for key, strat in list(strategies.items())[:-1]:
    strategy_ret = (SPY[key] - 1).diff().values[1:]
    beta = (np.cov(spy_ret,strategy_ret)/np.var(spy_ret))[1,0]
    residual_ret = strategy_ret - beta * spy_ret
    alpha = np.mean(residual_ret)
    print(strat, ' alpha:', alpha)
```

```
Decision Tree Strategy       alpha: 0.0014250818852215128
Bagging Classifier Strategy  alpha: 0.004555065833868771
Random Forest Strategy       alpha: 0.0033088661970511026
```