Machine Learning

Lecture 5.5

We are using the same optimal features and optimal hyperparameters we found on the first training period. Maybe this is not a good idea.

Instead we will compute the hyperparametrs and optimal features on a training set and then use these features on the next test set. This may not be the optimal features for the next training set but the idea (or hope) is that the optimal sets set do not change too much from one quarter to the next. Thus we compute the optimal feature on each training set. This takes quite a long time but some of them (with various hyperparameters) can be downloaded from the class files

We try one of them:

```
with open(r'optimal_hyperparameters.pkl','rb') as f:
    optimal_hyperparameters = pickle.load(f)
```

```
optimal_hyperparameters
```

```
[{'n_estimators': 30,
  'max_features': 'log2',
  'min_samples_leaf': 1600,
  'max_depth': 15},
 {'n_estimators': 10,
  'max_features': 'sqrt',
  'min_samples_leaf': 400,
  'max_depth': 20},
 {'n_estimators': 10,
  'max_features': 'log2',
  'min_samples_leaf': 800,
  'max_depth': 25},
 {'n_estimators': 30,
  'max_features': 'sqrt',
  'min_samples_leaf': 400,
  'max_depth': 15},
 {'n_estimators': 20,
  'max_features': 'sqrt'
```

As you can see this is an array of dicts of hyperparameters

We can then train RandomForest classifiers on each of the training sets with these parameters

```
classifiers = []
```

```
for hyp_par in optimal_hyperparameters:
    rf_clf = RandomForestClassifier(**hyp_par)
    classifiers.append(rf_clf)
```

```
for i in range(len(start_dates)-1):
    classifiers[i].fit(training_data[i],training_labels[i])
```

```
with open(r'classifiers.pkl','wb') as f:
    pickle.dump(classifiers,f)
```

and then find the features with feature importance > 0

```python
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)
```

```python
significant_features = []
```

```python
for i in range(len(start_dates)-1):
    fi = randomforest_feat_importances(classifiers[i],training_data[i])
    features = fi[(fi['feat_imp'] > 0.00)]['cols'].values
    significant_features.append(features)
```

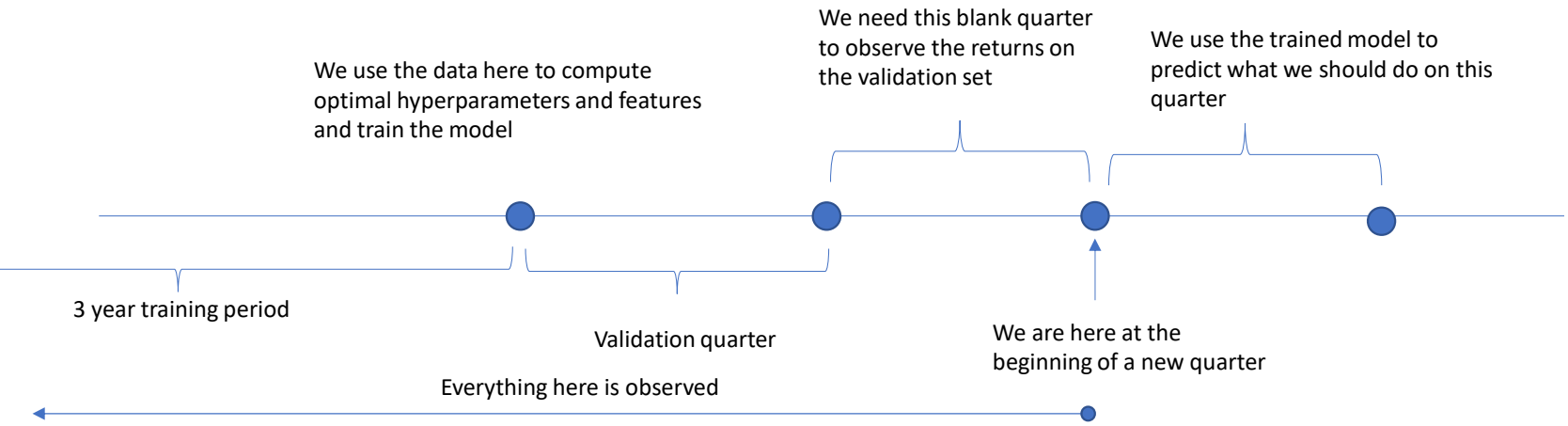
```python
significant_features
```

```
[array(['rectq', 'apq', 'equity_invcapq', 'curr_ratioq', 'oeps12',
        'ibcomq', 'oepsxq', 'epsfiq', 'fcf_csfhdq', 'opepsq', 'market_cap',
        'lctq', 'dvpspq', 'pe_op_basicq', 'cash_ratioq', 'debt_atq',
        'cf_yield', 'epsfi12', 'atq', 'oepsxy', 'pe_op_dilq', 'rect_actq',
        'lagbe4', 'ltq', 'yearly_sales', 'dvy', 'csh12q', 'actq', 'revty',
        'prccq', 'invtq', 'quick_ratioq', 'cheq', 'roeq', 'xinty', 'at4',
        'lt_debtq', 'book_value_yield', 'cfo-per-share', 'capxq', 'cshprq',
        'ibcq', 'niq', 'lagseq4', 'dlttq', 'lagppent_alt4', 'mibq',
        'dltisy', 'cfmq', 'txditcq', 'ibcomy'], dtype=object),
 array(['cfmq', 'epspi12', 'oeps12', 'oepf12', 'saleq', 'epsfi12',
        'curr_ratioq', 'pe_op_dilq', 'ptpmq', 'csh12q', 'market_cap',
        'dvpspq', 'cfo-per-share', 'capeiq', 'fcf_yield', 'pe_incq',
        'fcf_csfhdq', 'req', 'rectq', 'yearly_sales', 'oancfy',
        'pay_turnq', 'lagicapt4', 'oancfy_q', 'prccq', 'lagppent_alt4',
        'pretret_noaq', 'rd_saleq', 'sale_nwcq', 'cshfdq', 'be4', 'opepsq',
        'lctq', 'book_value_yield', 'cogsq', 'pe_op_basicq', 'beq',
        'debt_assetsq', 'curr_debtq', 'xrdq', 'apq', 'roeq', 'cshprq',
        'profit_lctq', 'at_turnq', 'cf_yield', 'dpraq', 'ibcq', 'xsgay'
```

We can then use shap to find the Shapley values that contribute to the validation set profits. This takes a very long time (appr. 8 hours).

Of course the reason it takes so long is because we are back testing and so has to compute the features for lots of quarters.

If we imagine we are running the strategy and we are at the beginning of a quarter



At the start of any new quarter we only have to find the optimal hyperparameters and –features for one training/validation set

```
with open(r'shap_features.pkl','rb') as f:
    shap_features = pickle.load(f)
```

```
shap_features[10]
```

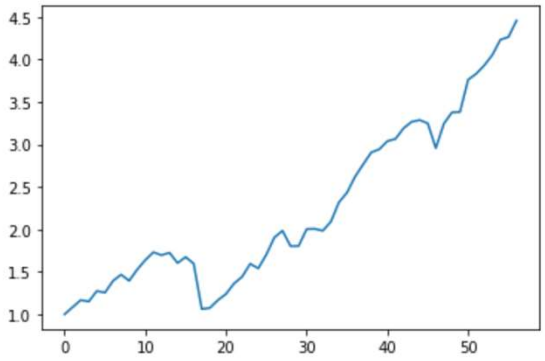
```
Index(['beq', 'curr_debtq', 'oancfy_q', 'oancfy', 'intcov_ratioq',
       'debt_invcapq', 'seqq', 'dvpspq', 'epsfxq', 'dvq', 'fcf_csfhdq',
       'book_value_yield', 'efftaxq', 'dpcq', 'actq', 'psq', 'at4', 'oibdpy'],
      dtype='object')
```

Now we can run our backtest just like before, except that at each we put in the classifier with the optimal hyperparameters for that quarter and the optimal features

```
x = [1]
ret = []

for i in range(len(start_dates)-1):

        classifiers[i].fit(opt_training_data[i],training_labels[i])

        preds = classifiers[i].predict(opt_test_data[i])
        profit_i = (preds*test_frames[i]['next_period_return']).sum()
        ret.append(profit_i)
        num_names = len(opt_test_data[i])
        x.append(x[i] + (x[i]/num_names)*profit_i)
```

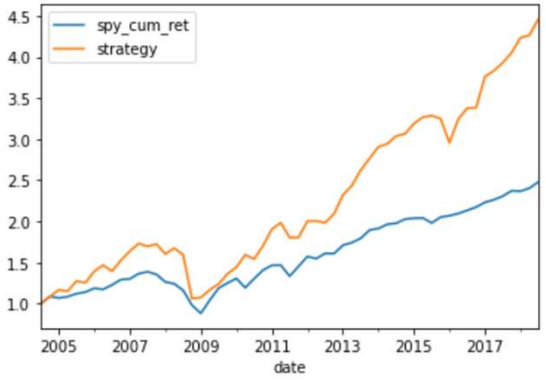
```

```

```
plt.plot(x);
```

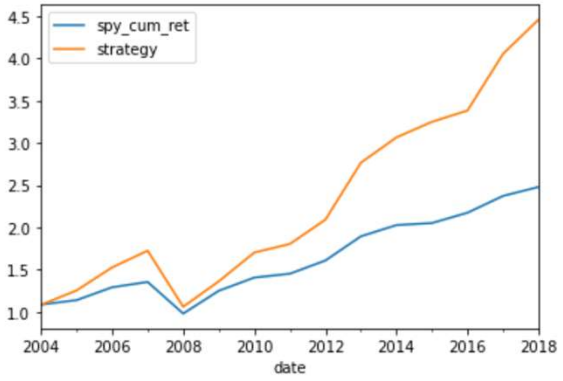


```
SPY = pd.read_pickle(r'C:\Users\niels\OneDrive\Machine Lea
SPY = SPY.loc['2004-09-01':'2018-09-30']
SPY = SPY.resample('Q').ffill()
SPY['spy_cum_ret'] = (SPY['spy_cum_ret'] - SPY['spy_cum_re
SPY['strategy'] = x
```

```
SPY.plot();
```




```
SPY = SPY.resample('Y').ffill()
SPY.plot();
```



We get a better Sharpe Ratio but the beta is still way too high

```
strategy_mean_ret = (SPY['strategy'] - 1).diff().mean()
strategy_std = (SPY['strategy'] - 1).diff().std()
strategy_sr = strategy_mean_ret/strategy_std
print('Strategy Sharpe Ratio: ',strategy_sr)
```

```
Strategy Sharpe Ratio:  0.7687828128384174
```

```
x[-1]
```

```
5]: 4.456992195589677
```

```
SPY['spy_cum_ret'][-1]
```

```
6]: 2.4779340000000003
```

```
strategy_ret = (SPY['strategy'] - 1).diff().values[1:]
spy_ret = (SPY['spy_cum_ret'] - 1).diff().values[1:]
```

```
beta = (np.cov(spy_ret,strategy_ret)/np.var(spy_ret))[1,0]
```

```
beta
```

```
8]: 1.5246470397535217
```