Return predictability with Machine Learning

Table of Contents

Introduction

In this live script we will see how to apply several Machine learning (ML) techniques in the context of out-of-sample prediction. We will look at a baseline case that predicts excess returns on the U.S. stock market (equity risk premia) using the set of predictors from Goyal and Welch (2008). As such, this is directy linked to the first lectures on return predictability. The aim of this script is to illustrate the application of ML techniques as well as how to trace out marginal effects/associations that enable interpretation of the models. We will see a very broad set of techniques, some that tune hyperparameters, and some that don't. The objective is not to showcase how well the techniques work. Rather, it is aimed at illustraing how to apply most of the techniques as well as their associated analyses related to marginal effects and hyperparameter tuning. Note that the script takes a while to run (multiple hours), but you can run parts of the code yourself.

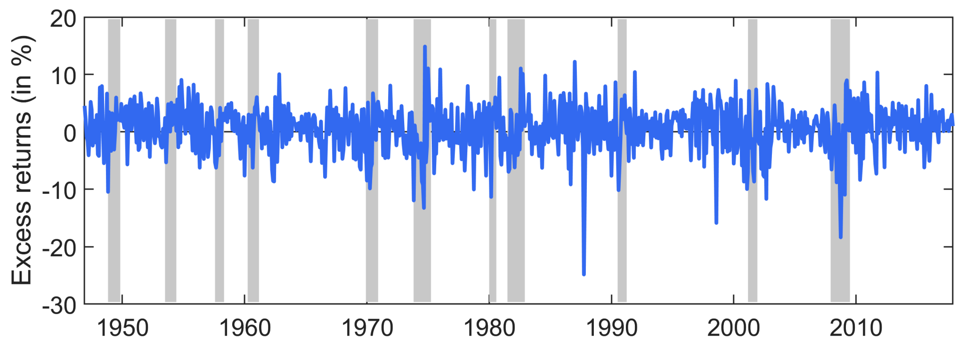
Data

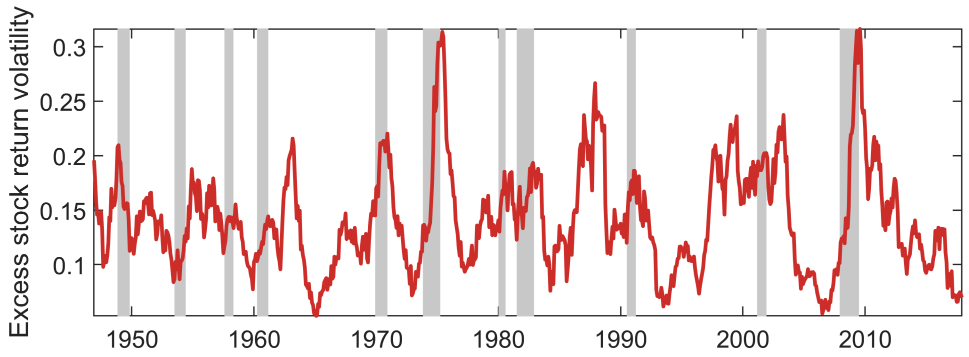
We use the predictor data from Goyal and Welch (2008) that was applied in the first lectures on return predictability. Instead of focusing on the dividend-price ratio only, we will not construct a full set of predictors, using monthly data. This amounts to 14 predictors. While this is nowhere near the amount used in Gu, Kelly, and Xiu (2020) (GKU) it is useful for the purpose of illustration here.

```
% Housekeeping
clear;
clc;

% Reading in data from raw excel file: 1946 - 2018
gwData      = xlsread('PredictorData2018','Monthly','a901:r1777');
recDates    = xlsread('USREC','FRED Graph','b1105:b1980');

% Constructing log excess returns
riskFree    = gwData(1:end-1,11);
returnSP500 = gwData(2:end,17);
retExcess   = log(1+returnSP500) - log(1+riskFree);
divGrowth   = diff(log(gwData(:,3)));

% 1. Constructing the log dividend-price ratio
sp500       = gwData(2:end,2);
d12         = gwData(2:end,3);
```

```
dpRatio     = log(d12./sp500);

% 2. Constructing the log dividend yield
sp500lagged = gwData(1:end-1,2);
dYield      = log(d12./sp500lagged);

% 3. Constructing the log earnings-price ratio
e12         = gwData(2:end,4);
epRatio     = log(e12./sp500);

% 4. Constructing the log dividend-payout ratio
deRatio     = log(d12./e12);

% 5. Constructing annualized stock excess return volatility
rxVol       = NaN(size(returnSP500,1)-11,1);
for iObs = 1:size(rxVol,1)
    % Computing volatility using the Mele (2007) estimator
    rxVol(iObs,1) = mean(abs(returnSP500(iObs:iObs+11,1)-riskFree(iObs:iObs+11,1)));
end
rxVol       = [NaN(11,1);sqrt(pi/2).*sqrt(12).*rxVol];

% 6. Constructing the book-to-market ratio
bmRatio     = gwData(2:end,5);

% 7. Constructing net equity issuance
ntis        = gwData(2:end,10);

% 8. Constructing the annualized Treasury bill rate
tblRate     = gwData(2:end,6);

% 9. Constructing the annualized long-term yield
ltYield     = gwData(2:end,9);

% 10. Constructing the annualized long-term return
ltReturn    = gwData(2:end,13);

% 11. Constructing the annualized term spread
tSpread     = ltYield - tblRate;

% 12. Constructing the annualized default yield spread
aaaYield    = gwData(2:end,7);
baaYield    = gwData(2:end,8);
dfYield     = baaYield - aaaYield;

% 13. Constructing the default return spread
corpReturn  = gwData(2:end,14);
dfReturn    = corpReturn - ltReturn;

% 14. Constructing lagged inflation (to account for data release)
inflRate    = gwData(1:end-1,12);

% Collecting all 14 predictors in a matrix
gwPredictors = [
    dpRatio ...
    dYield ...
    epRatio ...
    deRatio ...
    rxVol ...
    bmRatio ...
    ntis ...
    tblRate ...
    ltYield ...
    ltReturn ...
    tSpread ...
    dfYield ...
    dfReturn ...
    inflRate ...
];

% Cut since some are not defined due to the 12 month lag
gwPredictors  = gwPredictors(1+11:(end-12),:);
retExcess     = retExcess(1+11:(end-12),:);

predList = {
    'Dividend-price ratio'
    'Dividend yield'
    'Earnings-price ratio'
    'Dividend-earnings ratio'
    'Excess stock return volatility'
    'Book-to-market ratio'
    'Net equity issuance'
    'T-bill rate'
    'Long-term yield'
    'Long-term return'
    'Term spread'
    'Default yield spread'
    'Default return spread'
    'Lagged inflation'
};

% Setting number of time series observations
nObs        = size(retExcess,1);

% Setting datenum index
datenumIdx  = datenum(num2str(gwData(12+1:(end-12),1)),'yyyymm');

% Setting 0 to NaN for prettier plotting
recDates(recDates==0) = NaN;
recDates              = recDates(1+11:(end-12));

% Plotting returns against any of the predictors
whichPredictor    = 5;
figure;
subplot(2,1,1);
hold on
b1 = bar(datenumIdx,recDates.*19.5);
b2 = bar(datenumIdx,-recDates.*29.5);
b1.EdgeColor = [0.8 0.8 0.8];
```

```
b1.FaceColor = [0.8 0.8 0.8];
b2.EdgeColor = [0.8 0.8 0.8];
b2.FaceColor = [0.8 0.8 0.8];
b1.ShowBaseLine = 'Off';
b2.ShowBaseLine = 'Off';
p1 = plot(datenumIdx,retExcess.*100);
p1.Color = colorBrewer(1);
p1.LineWidth = 1.4;
box on
datetick('x','yyyy');
axis([-inf inf -30 20]);
ylabel('Excess returns (in %)');
subplot(2,1,2);
hold on
b1 = bar(datenumIdx,recDates.*min(gwPredictors(:,whichPredictor)));
b2 = bar(datenumIdx,recDates.*max(gwPredictors(:,whichPredictor)));
p1 = plot(datenumIdx,gwPredictors(:,whichPredictor));
b1.EdgeColor = [0.8 0.8 0.8];
b1.FaceColor = [0.8 0.8 0.8];
b2.EdgeColor = [0.8 0.8 0.8];
b2.FaceColor = [0.8 0.8 0.8];
b1.ShowBaseLine = 'Off';
b2.ShowBaseLine = 'Off';
p1.Color = colorBrewer(2);
p1.LineWidth = 1.4;
box on
datetick('x','yyyy');
axis([-inf inf min(gwPredictors(:,whichPredictor)) max(gwPredictors(:,whichPredictor))]);
ylabel(predList(whichPredictor));
```





Out-of-sample predictability (one month ahead)

Next, we consider an out-of-sample exercise. We consider an (initiall) estimation window of $R_1 = 180$, a fixed length (yet rolling) validation period of $R_2 - R_1 = 60$, and leave the rest of observations as true testing period. We use expanding window for the estimation routine.

```
% Setting parameters of the environment
initEstimationSample  = 180;
validationSample      = 60;
nFrcst                = nObs - initEstimationSample - validationSample;
```

Penalized linear regression

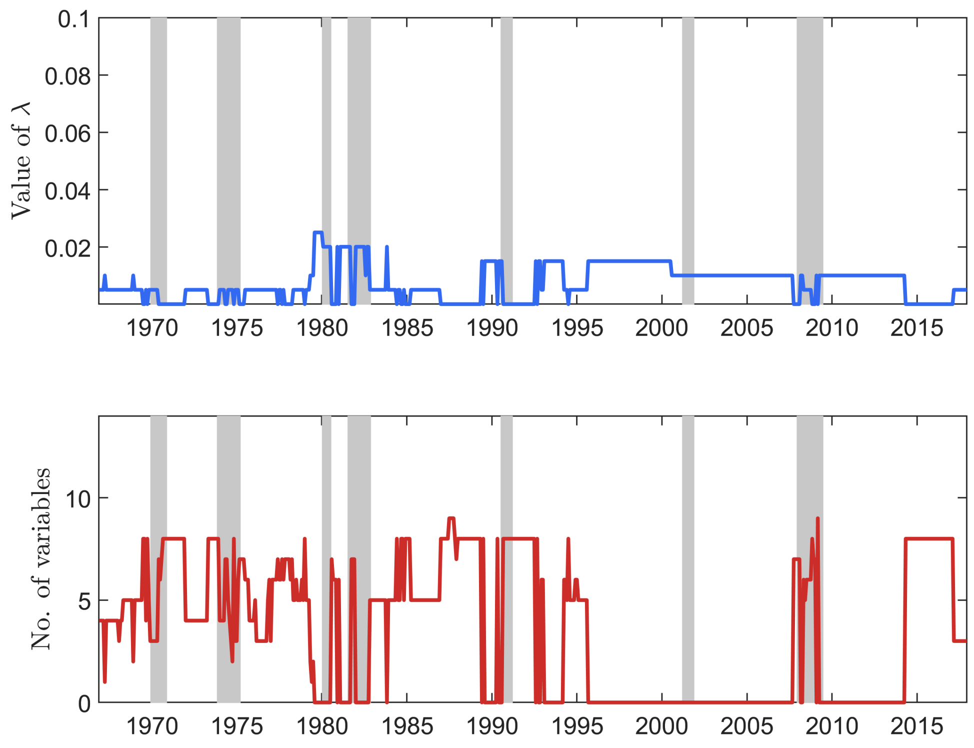
Implementation of the elastic net through Elastic Net (ENet) can be done in matlab using the function *lasso()*. It has two tuning parameters, λ and α.

```
% Preallocations prior to loop
actual_ENet      = NaN(nFrcst,1);
bench_ENet       = NaN(nFrcst,1);
retOOS_ENet      = NaN(nFrcst,1);
hyperParam_ENet  = NaN(nFrcst,1);
noParam_ENet     = NaN(nFrcst,1);
indxSave_ENet    = NaN(nFrcst,size(gwPredictors,2));

% Specify grid og hyperparameters
% Here we fix alpha = 0.5, which yields the elastic net, as done in GKU.
% One could use the grid window used in the GKU paper, which can be found in their Table A.5 ir
alphaGrid   = 0.5;
nAlpha      = length(alphaGrid);
lambdaGrid  = 10^(-4):0.005:10^(-1); % if predictors are not standardized, this might need to c
nLambda     = length(lambdaGrid);

for iFrcst = 1:nFrcst

    % Pre-allocation for validation
    retOOS  = NaN(validationSample,nLambda);
    actual  = NaN(validationSample,nLambda);
    bench   = NaN(validationSample,nLambda);

    % While we are in the validation sample, search for optimal hyperparameters
    for iVal = 1:validationSample
        for l=1:nLambda

                % Estimate model over estimation period and generate forecasts
                % for validation window. Note that the lasso function
                % automatically includes and estimates an intercept
                x                 = gwPredictors(1:initEstimationSample+iVal+iFrcst-2-1,:);
                y                 = retExcess(2:initEstimationSample+iVal+iFrcst-1-1,1);

                [coef,fitInfo]    = lasso(x,y,'Lambda',lambdaGrid(l),'Alpha',alphaGrid);
                coef              = [fitInfo.Intercept; coef];

                % Construct forecasts
                retOOS(iVal,l)    = [1 gwPredictors(initEstimationSample+iVal+iFrcst-1-1,:)]*coe

                % Actual realized returns
                actual(iVal,:)    = retExcess(initEstimationSample+iVal+iFrcst-1,1);

                % Historical average benchmark (because we consider the entire
                % stock market - recall the discussion in class or in GKU)
                bench(iVal,l)     = mean(retExcess(1:initEstimationSample+iVal+iFrcst-1-1,1));

        end
    end

    % Find the optimal hyperparamater by looking at the R2_OS of each spec
    R2os      = 100*(1 - mean((retOOS-actual).^2)./mean((bench-actual).^2));

    % Pick the value that has the maxium of R2_OS (if multiple, pick the smallest one)
    indx      = min(find(R2os == max(R2os)));
    lambdaOpt = lambdaGrid(indx);

    % Save for inspectation
    hyperParam_ENet(iFrcst,1) = lambdaOpt;

    % Conduct forecasting for testing period using chosen hyperparameter
    % Note that here, sometimes the models is still trained on only the
    % training period, but since we have already contaminated the
    % validation period by using it for selecting hyperparameters, we might
    % as well use it for esttimation now
    x                 = gwPredictors(1:initEstimationSample+iVal+iFrcst-2,:);
    y                 = retExcess(2:initEstimationSample+iVal+iFrcst-1,1);

    [coef,fitInfo]    = lasso(x,y,'Lambda',lambdaOpt,'Alpha',alphaGrid);
    coef              = [fitInfo.Intercept; coef];

    % Construct forecasts
    retOOS_ENet(iFrcst,1)   = [1 gwPredictors(initEstimationSample+iVal+iFrcst-1,:)]*coef;

    % Actual realized returns
    actual_ENet(iFrcst,1)   = retExcess(initEstimationSample+iVal+iFrcst,1);

    % Historical average benchmark (because we consider the entire
    % stock market - recall the discussion in class or in GKU)
    bench_ENet(iFrcst,1)    = mean(retExcess(1:initEstimationSample+iVal+iFrcst-1,1));

    % Save for inspection
    indx_param                = coef(2:end) > 0;
    noParam_ENet(iFrcst,1)    = sum(indx_param);
    indxSave_ENet(iFrcst,:)   = indx_param;

end
```

Plot the variation in the hyperparameter and in the no. of variables it implies to be selected to get a sense of the variation over time.

```
% Plotting hyperparameter and no. of implied variables
figure;
subplot(2,1,1);
hold on
b1 = bar(datenumIdx(1+(initEstimationSample+validationSample):end,1),recDates(1+(initEstimation
b2 = bar(datenumIdx(1+(initEstimationSample+validationSample):end,1),-recDates(1+(initEstimatio
b1.EdgeColor = [0.8 0.8 0.8];
b1.FaceColor = [0.8 0.8 0.8];
b2.EdgeColor = [0.8 0.8 0.8];
b2.FaceColor = [0.8 0.8 0.8];
b1.ShowBaseLine = 'Off';
b2.ShowBaseLine = 'Off';
p1 = plot(datenumIdx(1+(initEstimationSample+validationSample):end,1),hyperParam_ENet);
p1.Color = colorBrewer(1);
p1.LineWidth = 1.4;
box on
datetick('x','yyyy');
axis([-inf inf 10^(-4) 10^(-1)]);
ylabel('Value of $\lambda$','Interpreter','latex');
subplot(2,1,2);
hold on
b1 = bar(datenumIdx(1+(initEstimationSample+validationSample):end,1),recDates(1+(initEstimation
b2 = bar(datenumIdx(1+(initEstimationSample+validationSample):end,1),-recDates(1+(initEstimatio
b1.EdgeColor = [0.8 0.8 0.8];
b1.FaceColor = [0.8 0.8 0.8];
b2.EdgeColor = [0.8 0.8 0.8];
b2.FaceColor = [0.8 0.8 0.8];
b1.ShowBaseLine = 'Off';
```

```
b2.ShowBaseLine = 'Off';
p1 = plot(datenumIdx(1+(initEstimationSample+validationSample):end,1),noParam_ENet);
b1.EdgeColor = [0.8 0.8 0.8];
b1.FaceColor = [0.8 0.8 0.8];
b1.ShowBaseLine = 'Off';
p1.Color = colorBrewer(2);
p1.LineWidth = 1.4;
box on
datetick('x','yyyy');
axis([-inf inf 0 14]);
ylabel('No. of variables','Interpreter','latex');
```



Inspect which variable is chosen most often (computing inclusion frequency). One could also inspect the time series of those inclusions to assess when certain variables are chosen.

```
% Computing inclusion frequency
incFreq     = mean(indxSave_ENet)
```

```
incFreq = 1×14
    0.5253    0.2137    0.2219    0.3132    0.4095    0.1533    0.0065    0.0440 ⋯
```

Statistical evaluation

We consider the out-of-sample R^2 suggested in Campbell and Thompson (2008)

$$\mathrm{R}^2_{OS} = 1 - \frac{\mathrm{MSFE}_x}{\mathrm{MSFE}_{HA}} = 1 - \frac{\sum_{i=R_2+1}^{T} (r_i - \hat{r}_i)^2}{\sum_{i=R_2+1}^{T} (r_i - \bar{r}_i)^2}$$

and test for significance using the Diebold and Mariano (1995) test

```
% Computing out-of-sample R2
R2oos    = 100*(1 - mean((retOOS_ENet-actual_ENet).^2)./mean((bench_ENet-actual_ENet).^2))
```

```
R2oos = -3.3367
```

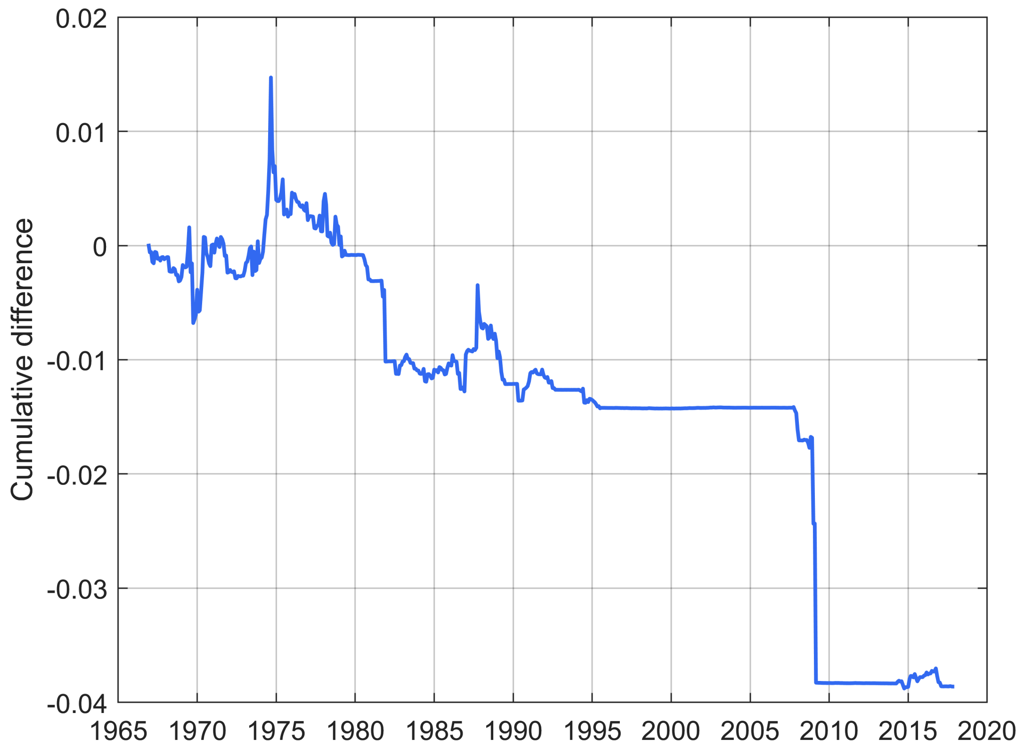
```


% Conducting Diebold-Mariano test
ft       = (bench_ENet-actual_ENet).^2 - (retOOS_ENet-actual_ENet).^2;
dmTest   = nwRegress(ft,ones(size(ft,1),1),0,3);
dmPval   = 1-normcdf(dmTest.tbv,0,1)
```

```
dmPval = 0.9311
```

Finally, we can compute and plot the Goyal and Welch (2008) graphical device using the cumulative differences in squared forecast errors (CDSFE)

$$\mathrm{CDSFE}_t = \sum_{i=R_2+1}^{t} (r_i - \bar{r}_i)^2 - \sum_{i=R_2+1}^{t} (r_i - \hat{r}_i)^2$$

```
% Computing CDSFE
cdsfe = cumsum( (bench_ENet-actual_ENet).^2 ) - cumsum( (retOOS_ENet-actual_ENet).^2 );

% Plotting CDSFE
figure;
p1 = plot(datenumIdx(1+(initEstimationSample+validationSample):end,1),cdsfe);
p1(1).Color = colorBrewer(1);
p1(1).LineWidth = 1.4;
datetick('x','yyyy');
ylabel('Cumulative difference');
box on
grid on
```



Random forrests

We no longer conduct hyperparameter tuning for expositional reasons. It should, however, be straightforward to adjust using the code applied to the ENet above. As a result, we consider an (initiall) estimation window of $R_2 = 240$ and leave the rest of observations as true testing period. We use expanding window for the estimation routine. Note that since we have a finite number of trees in the bagging step of the random forests, you will with probability zero obtain the exact same numbers as I do in this script when re-running it due to randomization.

```
% Setting parameters of the environment
initEstimationSample  = 240;
nFrcst                = nObs - initEstimationSample;

% Preallocations prior to loop
actual_RF      = NaN(nFrcst,1);
bench_RF       = NaN(nFrcst,1);
retOOS_RF      = NaN(nFrcst,1);

% Set hypterparameters
noTrees        = 300;
%... noting that the no. of features in each split takes p/3 for regression
% trees, here p being the no. of predictors, and that we allow the trees to
% be deep by not restricting its depth.

for iFrcst = 1:nFrcst
    x                  = gwPredictors(1:initEstimationSample+iFrcst-2,:);
    y                  = retExcess(2:initEstimationSample+iFrcst-1,1);
    fitInfo          = TreeBagger(noTrees,x,y,'Method','regression');

    % Construct forecasts
    retOOS_RF(iFrcst,1)   = predict(fitInfo,gwPredictors(initEstimationSample+iFrcst-1,:));

    % Actual realized returns
    actual_RF(iFrcst,1)   = retExcess(initEstimationSample+iFrcst,1);

    % Historical average benchmark (because we consider the entire
    % stock market - recall the discussion in class or in GKU)
    bench_RF(iFrcst,1)    = mean(retExcess(1:initEstimationSample+iFrcst-1,1));
end
```

... and then some statistical evaluation of the forecasts.

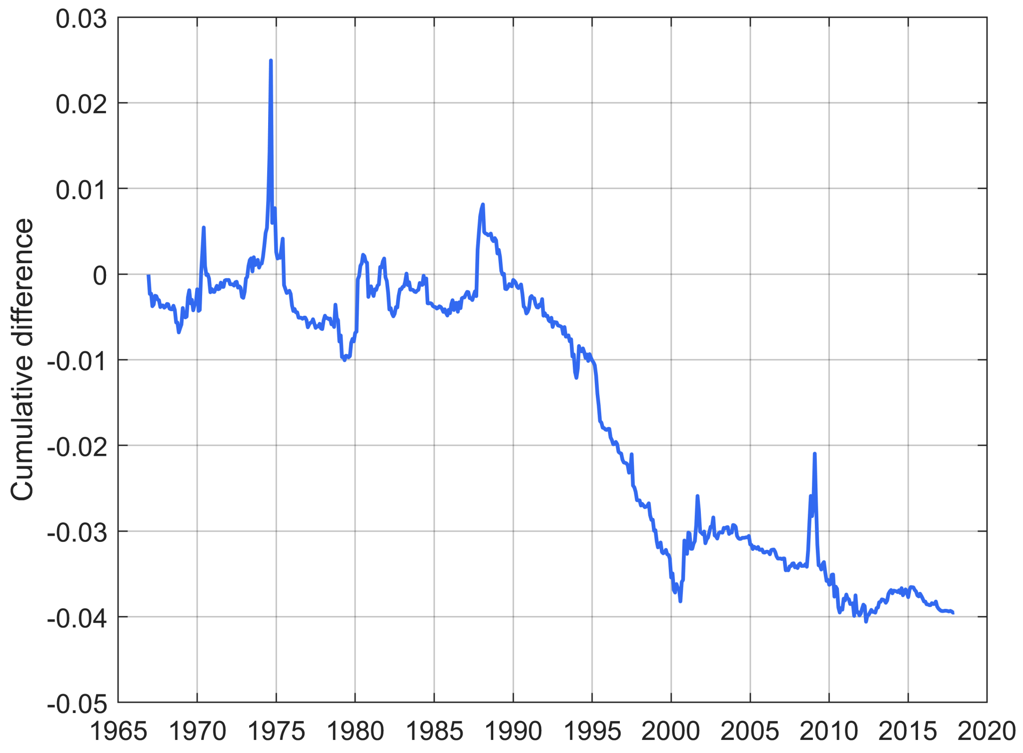
```
% Computing out-of-sample R2
R2oos   = 100*(1 - mean((retOOS_RF-actual_RF).^2)./mean((bench_RF-actual_RF).^2))
```

```
R2oos = -3.4136
```

```

% Conducting Diebold-Mariano test
ft      = (bench_RF-actual_RF).^2 - (retOOS_RF-actual_RF).^2;
dmTest  = nwRegress(ft,ones(size(ft,1),1),0,3);
dmPval  = 1-normcdf(dmTest.tbv,0,1)
```

```
dmPval = 0.8941
```

```

% Computing CDSFE
cdsfe = cumsum( (bench_RF-actual_RF).^2 ) - cumsum( (retOOS_RF-actual_RF).^2 );

% Plotting CDSFE
figure;
p1 = plot(datenumIdx(1+(initEstimationSample):end,1),cdsfe);
p1(1).Color = colorBrewer(1);

p1(1).LineWidth = 1.4;
datetick('x','yyyy');
ylabel('Cumulative difference');
box on
grid on
```



Variable importance and marginal effects

To trace out variable importance and marginal effects, we can use the techniques by GKU. To obtain variable importance, we compute for each variable the impact of the out-of-sample R^2 by leaving its information out when forming the prediction. That is, setting all its values equal to zero when forming the prediction, yet keeping all other estimates fixed.

```
marginalR2           = NaN(size(gwPredictors,2),1);

for m=1:size(gwPredictors,2)

    % Set variable m equal to zero for all t
    marginal_gwPredictors   = gwPredictors;
    marginal_gwPredictors(:,m) = zeros(nObs,1);

    for iFrcst = 1:nFrcst

        x                   = gwPredictors(1:initEstimationSample+iFrcst-2,:);
        y                   = retExcess(2:initEstimationSample+iFrcst-1,1);

        fitInfo             = TreeBagger(noTrees,x,y,'Method','regression');

        % Construct forecasts
        retOOS_RF(iFrcst,1)    = predict(fitInfo,marginal_gwPredictors(initEstimationSample+iFrc

        % Actual realized returns
        actual_RF(iFrcst,1)    = retExcess(initEstimationSample+iFrcst,1);
        % Historical average benchmark (because we consider the entire
        % stock market - recall the discussion in class or in GKU)
        bench_RF(iFrcst,1)    = mean(retExcess(1:initEstimationSample+iFrcst-1,1));
    end

    % Computing out-of-sample R2
    R2oos_temp   = 100*(1 - mean((retOOS_RF-actual_RF).^2)./mean((bench_RF-actual_RF).^2));

    % Save new R2_OS
    marginalR2(m,1)  = R2oos_temp;

end
```

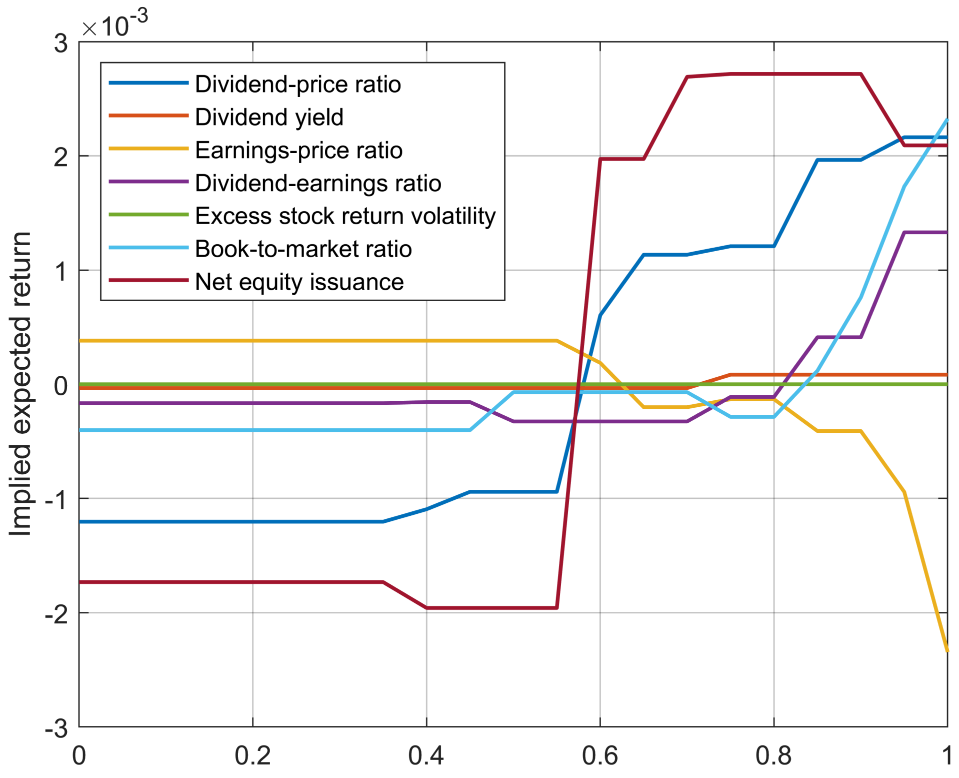
To trace out marginal effects, we estimate a predictive version of the RF over all time points. Next, we set all variables except the *j*'th one equal to their median values and, subsequently, vary the *j*'th variable across its entire support and save the fitted expected returns.

```
noSteps                 = 20; % no of partitions of support of x (in excess of max)
marginalExpRet          = NaN(noSteps+1,size(gwPredictors,2));

% Estimate predictive RF over full sample
x                   = gwPredictors(1:end-1,:);
y                   = retExcess(2:end,1);
fitInfo             = TreeBagger(noTrees,x,y,'Method','regression');

for m=1:size(gwPredictors,2)

    marginal_gwPredictors  = median(gwPredictors);
    stepX     = 0:1/noSteps:1;
    temp      = prctile(gwPredictors(:,m),stepX);

    for iStep=1:length(stepX)

        % Set value of x
        marginal_gwPredictors(m)    = temp(iStep);

        % Predict/fit expected returns using entire support of x
        expRet  = predict(fitInfo,marginal_gwPredictors);

        % Save
        marginalExpRet(iStep,m) = expRet;

    end

end
```

We can analyse the marginal effects over the domain of the *j*'th variale by plotting the implied expected returns against the partition of its domain.

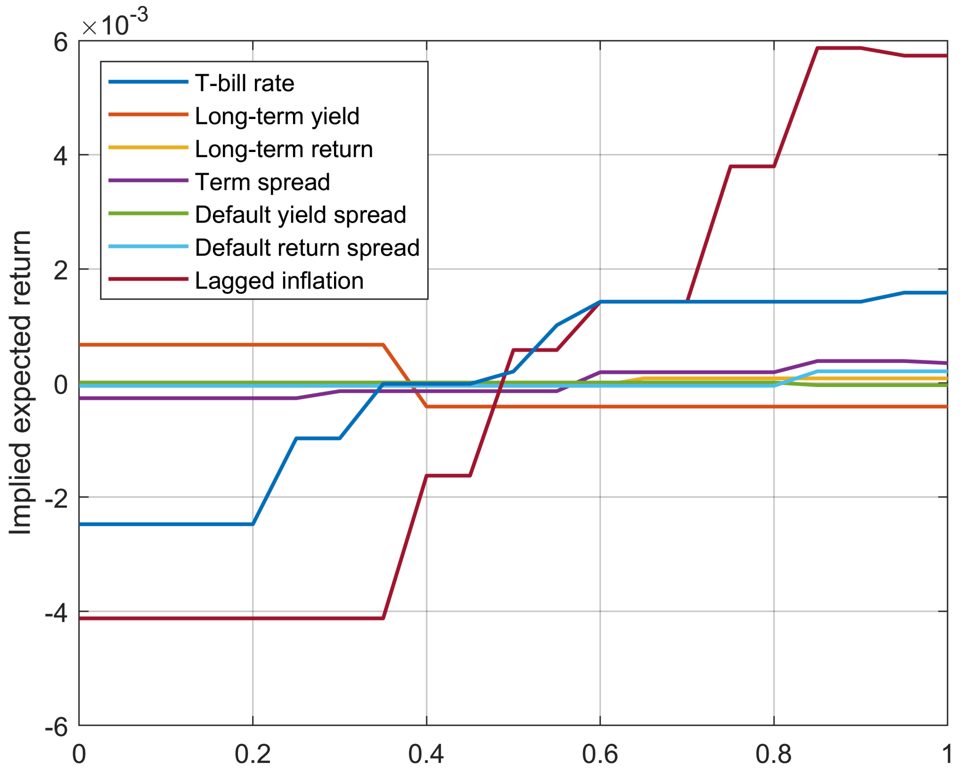
```
% Too see the impact of changing, we can normalize the effects to being
% mean zero
marginalExpRetNorm = marginalExpRet - mean(marginalExpRet);
```

```
% Let's plot all, puting the first 7 in the first graph and the rest in a
% second graph
whichVariable = 1;
figure;
p1 = plot(stepX,marginalExpRetNorm(:,whichVariable));
% p1(1).Color = colorBrewer(1);
p1(1).LineWidth = 1.4;
ylabel('Implied expected return');
% title(predList(whichVariable))
box on
grid on
hold on
for whichVariable=2:7
    p1(whichVariable) = plot(stepX,marginalExpRetNorm(:,whichVariable));
    p1(whichVariable).LineWidth = 1.4;
end
hold off
legend(predList(1:7),'Location','northwest')
```



```
whichVariable = 8;
figure;
p1 = plot(stepX,marginalExpRetNorm(:,whichVariable));
% p1(1).Color = colorBrewer(1);
p1(1).LineWidth = 1.4;
ylabel('Implied expected return');
% title(predList(whichVariable))
```

```
box on
grid on
hold on
for whichVariable=8:14
    p1(whichVariable) = plot(stepX,marginalExpRetNorm(:,whichVariable));
    p1(whichVariable).LineWidth = 1.4;
end
hold off
legend(predList(8:end),'Location','northwest')
```



```
% The effect on annualized monthly returns from moving from the median onto
% the 80th percentile, for instance, is then computed as (note that this
% code is dependent on your choice of noSteps.
for whichVariable=1:size(gwPredictors,2)
    expRetEffect(whichVariable) = (marginalExpRetNorm(17,whichVariable) - marginalExpRetNorm(11
end
tableEffcts = table(predList,round(expRetEffect',4))
```

```
tableEffcts = 14×2 table
```

	predList	Var2
1	'Dividend-...	2.5799
2	'Dividend ...	0.1412
3	'Earnings-...	-0.6152
4	'Dividend-...	0.2576

	predList	Var2
5	'Excess st...	0
6	'Book-to-m...	-0.2593
7	'Net equit...	5.6102
8	'T-bill rate'	0
9	'Long-term...	0.1577
10	'Long-term...	0.3983
11	'Term spread'	0
12	'Default y...	0
13	'Default r...	3.8608
14	'Lagged in...	1.4672

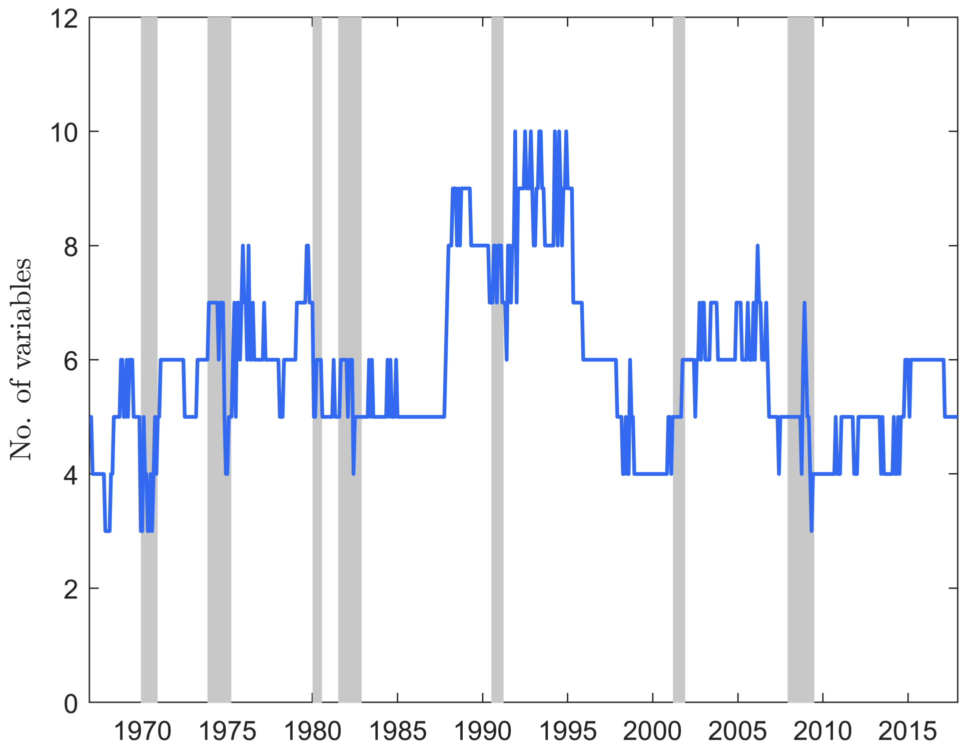
Generalized linear model

In the generalized linear model, one add non-linear transformations of all predictors additively to the linear model. Here we will see a simple example of adding squared transformations as well as all interactions among predictors. This model will then be estimated trough the principles of the penalized linear regression above, yet we will not to hyperparameter optimization here.

```
% First construct expanded data set with non-linear transforms (here squared and cubed values)

% Predictors expanded
gwPredictors_expand = [gwPredictors gwPredictors.^2 gwPredictors.^3];

% Preallocations prior to loop
actual_GLM      = NaN(nFrcst,1);
bench_GLM       = NaN(nFrcst,1);
retOOS_GLM      = NaN(nFrcst,1);
indxSave_GLM    = NaN(nFrcst,size(gwPredictors_expand,2));

% Specify grid og hyperparameters
% Here we fix alpha = 0.5, which yields the elastic net, as done in GKU.
% One could use the grid window used in the GKU paper, which can be found in their Table A.5 ir
alpha   = 0.5;
lambda  = 10^(-3);

for iFrcst = 1:nFrcst

    x                 = gwPredictors_expand(1:initEstimationSample+iFrcst-2,:);
    y                 = retExcess(2:initEstimationSample+iFrcst-1,1);

    [coef,fitInfo]    = lasso(x,y,'Lambda',lambdaOpt,'Alpha',alphaGrid);
    coef              = [fitInfo.Intercept; coef];

    % Construct forecasts
    retOOS_GLM(iFrcst,1)    = [1 gwPredictors_expand(initEstimationSample+iFrcst-1,:)]*coef;

    % Actual realized returns
    actual_GLM(iFrcst,1)    = retExcess(initEstimationSample+iFrcst,1);

    % Historical average benchmark (because we consider the entire
    % stock market - recall the discussion in class or in GKU)
    bench_GLM(iFrcst,1)     = mean(retExcess(1:initEstimationSample+iFrcst-1,1));

    % Save for inspection
    indx_param                = coef(2:end) > 0;
    noParam_GLM(iFrcst,1)     = sum(indx_param);
    indxSave_GLM(iFrcst,:)    = indx_param;

end
```

Plot the variation in the no. of variables it implies to be selected to get a sense of the variation over time.

```
% Plotting hyperparameter and no. of implied variables
figure;
hold on
b1 = bar(datenumIdx(1+(initEstimationSample):end,1),recDates(1+(initEstimationSample):end,1).*2
b2 = bar(datenumIdx(1+(initEstimationSample):end,1),-recDates(1+(initEstimationSample):end,1).*
b1.EdgeColor = [0.8 0.8 0.8];
b1.FaceColor = [0.8 0.8 0.8];
b2.EdgeColor = [0.8 0.8 0.8];
b2.FaceColor = [0.8 0.8 0.8];
b1.ShowBaseLine = 'Off';
b2.ShowBaseLine = 'Off';
p1 = plot(datenumIdx(1+(initEstimationSample):end,1),noParam_GLM);
p1.Color = colorBrewer(1);
p1.LineWidth = 1.4;
box on
datetick('x','yyyy');
axis([-inf inf 0 12]);
ylabel('No. of variables','Interpreter','latex');
```



Inspect which variable is chosen most often (computing inclusion frequency). One could also inspect the time series of those inclusions to assess when certain variables are chosen.

```
% Computing inclusion frequency
incFreq     = mean(indxSave_GLM)
```

```
incFreq = 1×42
    0.7847    0.6248         0    0.2692    0.5514    0.0131         0         0···
```

... and then some statistical evaluation of the forecasts.

```
% Computing out-of-sample R2
R2oos    = 100*(1 - mean((retOOS_GLM-actual_GLM).^2)./mean((bench_GLM-actual_GLM).^2))
```

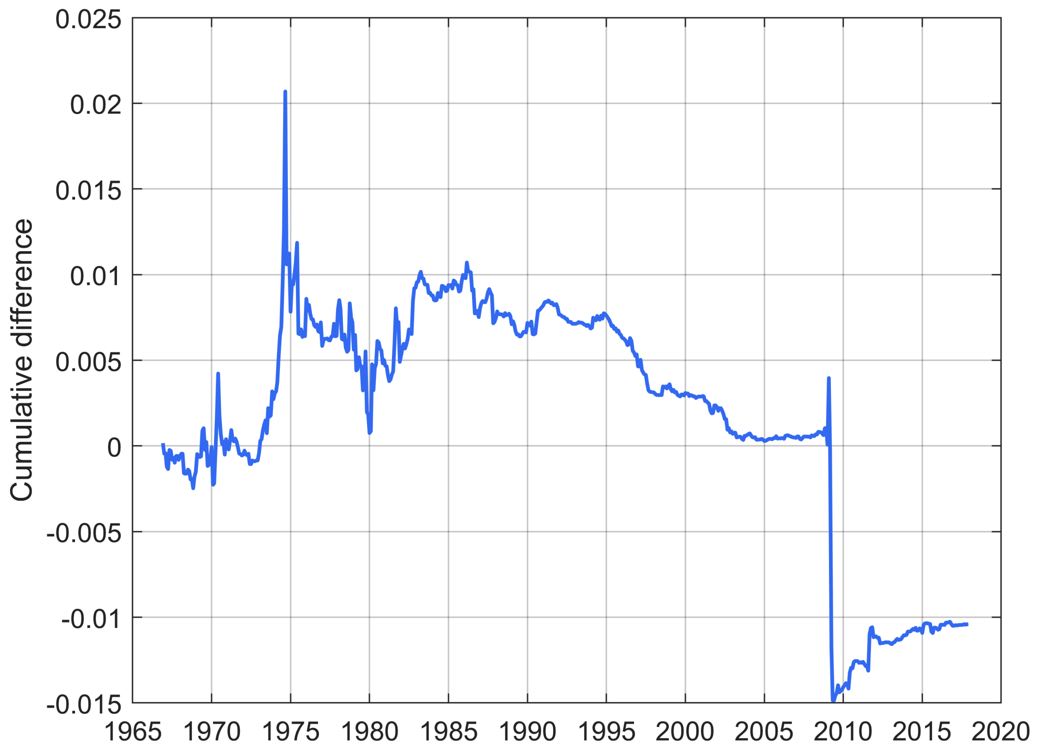
```
R2oos = -0.8988
```

```

% Conducting Diebold-Mariano test
ft       = (bench_GLM-actual_GLM).^2 - (retOOS_GLM-actual_GLM).^2;
dmTest   = nwRegress(ft,ones(size(ft,1),1),0,3);
dmPval   = 1-normcdf(dmTest.tbv,0,1)
```

```
dmPval = 0.6766
```

```

% Computing CDSFE
cdsfe = cumsum( (bench_GLM-actual_GLM).^2 ) - cumsum( (retOOS_GLM-actual_GLM).^2 );
```

```
% Plotting CDSFE
figure;
p1 = plot(datenumIdx(1+(initEstimationSample):end,1),cdsfe);
p1(1).Color = colorBrewer(1);

p1(1).LineWidth = 1.4;
datetick('x','yyyy');
ylabel('Cumulative difference');
box on
grid on
```



Boosted regression trees

```
initEstimationSample  = 240;
nFrcst                = nObs - initEstimationSample;

% Preallocations prior to loop
actual_BRT      = NaN(nFrcst,1);
bench_BRT       = NaN(nFrcst,1);
retOOS_BRT      = NaN(nFrcst,1);

% Set hypterparameters
noEnsembles          = 300;
learnRate            = 0.005; %Shrinkage like GKU Appendix E

for iFrcst = 1:nFrcst

    x                  = gwPredictors(1:initEstimationSample+iFrcst-2,:);
    y                 = retExcess(2:initEstimationSample+iFrcst-1,1);

    fitInfo           = fitrensemble(x,y,'NumLearningCycles',noEnsembles,'LearnRate',learnRate);

    % Construct forecasts
    retOOS_BRT(iFrcst,1)    = predict(fitInfo,gwPredictors(initEstimationSample+iFrcst-1,:));

    % Actual realized returns
    actual_BRT(iFrcst,1)    = retExcess(initEstimationSample+iFrcst,1);

    % Historical average benchmark (because we consider the entire
    % stock market - recall the discussion in class or in GKU)
    bench_BRT(iFrcst,1)     = mean(retExcess(1:initEstimationSample+iFrcst-1,1));

end
```

... and then some statistical evaluation of the forecasts.

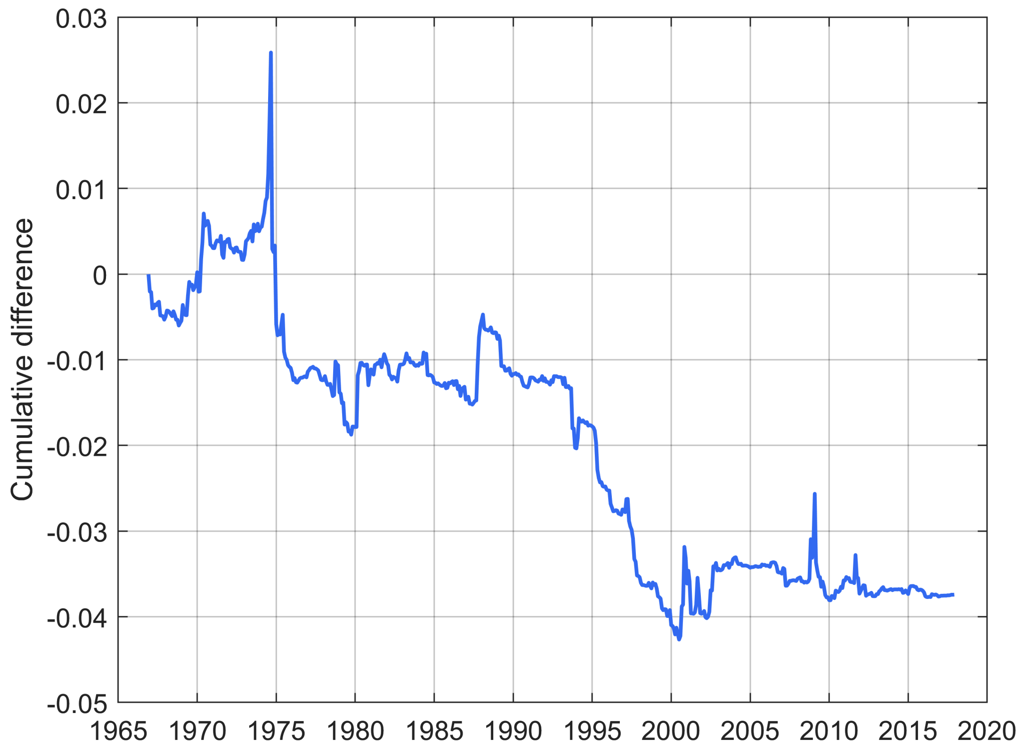
```
% Computing out-of-sample R2
R2oos    = 100*(1 - mean((retOOS_BRT-actual_BRT).^2)./mean((bench_BRT-actual_BRT).^2))
```

```
R2oos = -3.2334
```

```
% Conducting Diebold-Mariano test
ft       = (bench_BRT-actual_BRT).^2 - (retOOS_BRT-actual_BRT).^2;
dmTest   = nwRegress(ft,ones(size(ft,1),1),0,3);
dmPval   = 1-normcdf(dmTest.tbv,0,1)
```

```
dmPval = 0.8626
```

```
% Computing CDSFE
cdsfe = cumsum( (bench_BRT-actual_BRT).^2 ) - cumsum( (retOOS_BRT-actual_BRT).^2 );

% Plotting CDSFE
figure;
p1 = plot(datenumIdx(1+(initEstimationSample):end,1),cdsfe);
p1(1).Color = colorBrewer(1);

p1(1).LineWidth = 1.4;
datetick('x','yyyy');
ylabel('Cumulative difference');
box on
grid on
```



PCR and PLS

Forecasting using Principal Components or components extracted using Partial Least Squares have the same fundamental idea in that they reduced the dimension of the predictor set to a set of components that capture the most important variation (either unsupervised or supervised).

```
initEstimationSample  = 240;
nFrcst                = nObs - initEstimationSample;

% Preallocations prior to loop
actual_PCA      = NaN(nFrcst,1);
bench_PCA       = NaN(nFrcst,1);
retOOS_PCA      = NaN(nFrcst,1);

% Set hypterparameters
noPCA        = 3; %set no. of common components to extract via PCA
noPLS        = 3; %set no. of common components to extract via PCA

for iFrcst = 1:nFrcst

    x                 = gwPredictors(1:initEstimationSample+iFrcst-1,:); % use all information p
    y                 = retExcess(2:initEstimationSample+iFrcst-1,1);

    % Estimate PCA components
    [~,compPCA]       = pca(zscore(x));

    % Pick the number of components that explain the most, according to
    % your choice of noPCA
    pickPCA           = compPCA(:,1:noPCA);

    % Estimate PCR model
    resPCA            = nwRegress(y,pickPCA(1:end-1,:),1,3);

    % Construct forecasts
    retOOS_PCA(iFrcst,1)  = [1 pickPCA(end,:)]*resPCA.bv;

    % Actual realized returns
    actual_PCA(iFrcst,1)   = retExcess(initEstimationSample+iFrcst,1);

    % Historical average benchmark (because we consider the entire
    % stock market - recall the discussion in class or in GKU)
    bench_PCA(iFrcst,1)    = mean(retExcess(1:initEstimationSample+iFrcst-1,1));

    % Estimate PLS weigths
    weightPLS         = plsregress(zscore(x(1:end-1,:)),y,noPLS);

    % Estimate compPLS (use estimated weights (that are not optimized to x(end))
    compPLS           = zscore(x)*weightPLS;

    % Estimate PLS model
    resPLS            = nwRegress(y,compPLS(1:end-1,:),1,3);

    % Construct forecasts
    retOOS_PLS(iFrcst,1)  = [1 compPLS(end,:)]*resPLS.bv;

    % Actual realized returns
    actual_PLS(iFrcst,1)   = retExcess(initEstimationSample+iFrcst,1);

    % Historical average benchmark (because we consider the entire
    % stock market - recall the discussion in class or in GKU)
    bench_PLS(iFrcst,1)    = mean(retExcess(1:initEstimationSample+iFrcst-1,1));

end
```

... and then some statistical evaluation of the forecasts.

```
% PCA
% Computing out-of-sample R2
R2oos    = 100*(1 - mean((retOOS_PCA-actual_PCA).^2)./mean((bench_PCA-actual_PCA).^2))
```

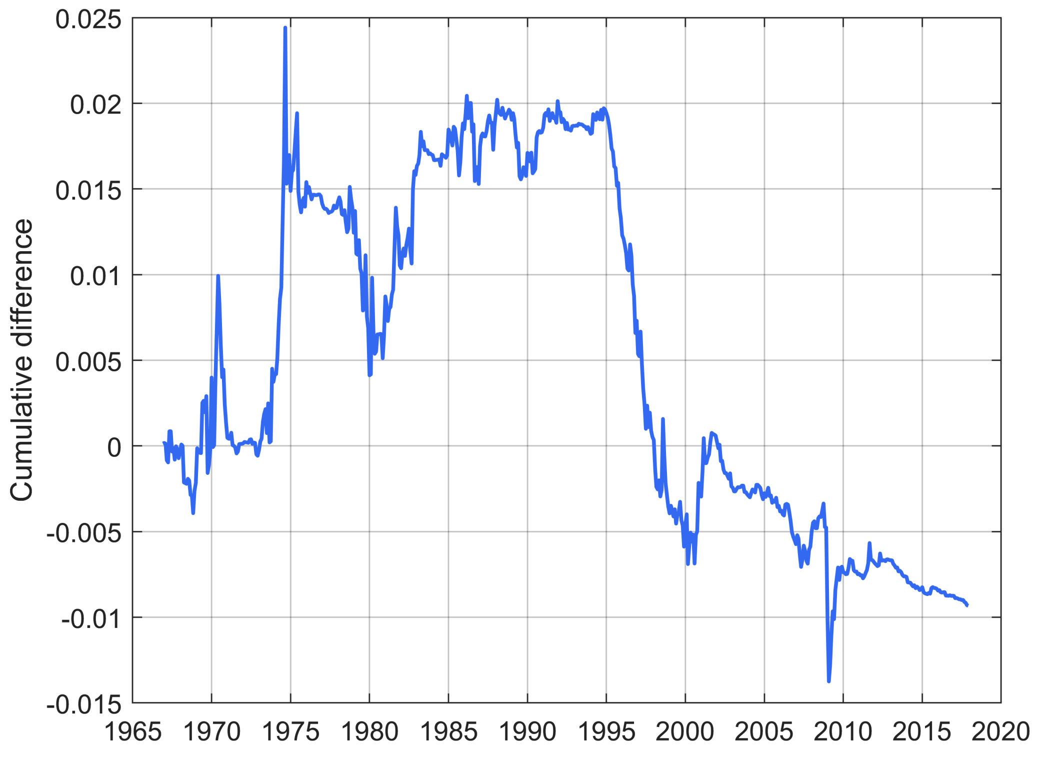
```
R2oos = -0.8063
```

```

% Conducting Diebold-Mariano test
ft       = (bench_PCA-actual_PCA).^2 - (retOOS_PCA-actual_PCA).^2;
dmTest   = nwRegress(ft,ones(size(ft,1),1),0,3);
dmPval   = 1-normcdf(dmTest.tbv,0,1)
```

```
dmPval = 0.6436
```

```

% Computing CDSFE
cdsfe = cumsum( (bench_PCA-actual_PCA).^2 ) - cumsum( (retOOS_PCA-actual_PCA).^2 );
```

```
% Plotting CDSFE
figure;
p1 = plot(datenumIdx(1+(initEstimationSample):end,1),cdsfe);
p1(1).Color = colorBrewer(1);

p1(1).LineWidth = 1.4;
datetick('x','yyyy');
ylabel('Cumulative difference');
box on
grid on
```



```
% PLS
% Computing out-of-sample R2
R2oos    = 100*(1 - mean((retOOS_PLS-actual_PLS).^2)./mean((bench_PLS-actual_PLS).^2))
```

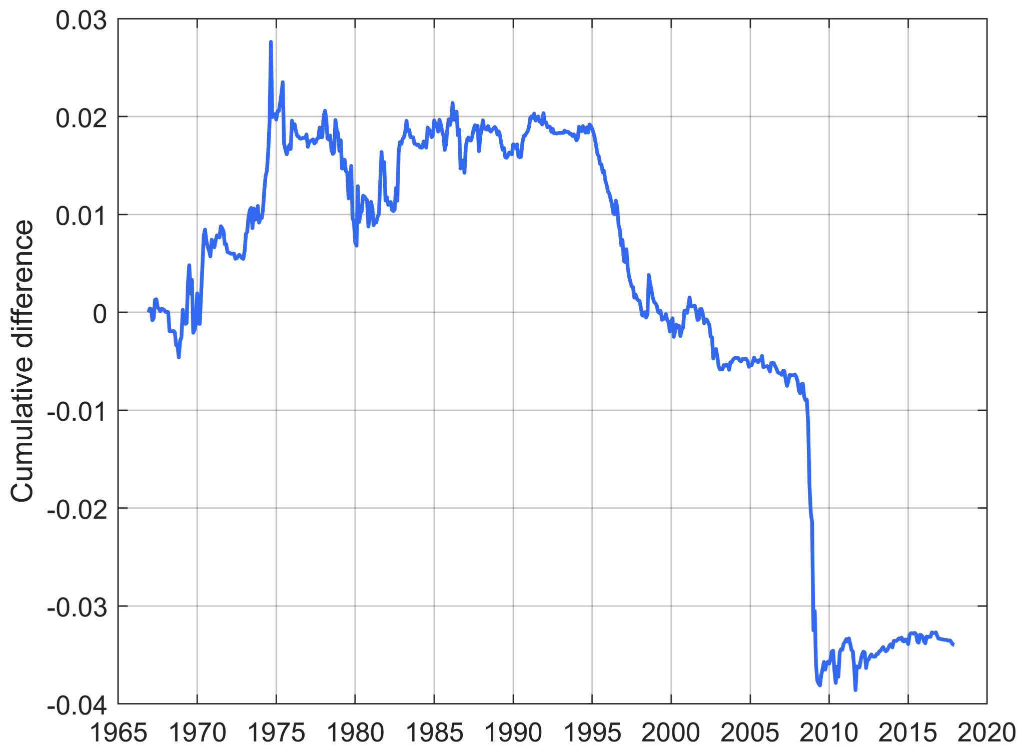
```
R2oos = -2.9357
```

```
% Conducting Diebold-Mariano test
ft       = (bench_PLS-actual_PLS).^2 - (retOOS_PLS-actual_PLS).^2;
dmTest   = nwRegress(ft,ones(size(ft,1),1),0,3);
dmPval   = 1-normcdf(dmTest.tbv,0,1)
```

```
dmPval = 0.8560
```

```
% Computing CDSFE
```

```
cdsfe = cumsum( (bench_PLS-actual_PLS).^2 ) - cumsum( (retOOS_PLS-actual_PLS).^2 );

% Plotting CDSFE
figure;
p1 = plot(datenumIdx(1+(initEstimationSample):end,1),cdsfe);
p1(1).Color = colorBrewer(1);

p1(1).LineWidth = 1.4;
datetick('x','yyyy');
ylabel('Cumulative difference');
box on
grid on
```


Cumulative difference