Predictive modelling for Forex trading with classical and ML techniques

Data

Intra-day data, five years (January 2007 - January 2011) worth of one-minute bar, on returns on the currency pair EUR/USD.

Goal

Predict the future return (e.g. 60 minutes into the future) so to trade using the predictions, using short-term returns and technical indicators.

Tools

Comparison between classical statistical approach with linear regression and the Machine Learning approach. Cross-validation is performed (train a model on the in-sample and test it on the out-sample).

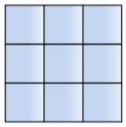
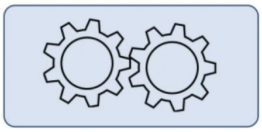
Backtesting

Backtesting on both the two approaches by taking into account bid/offer spread as trading cost and varying the length of in-sample and out-sample to check for robustness.

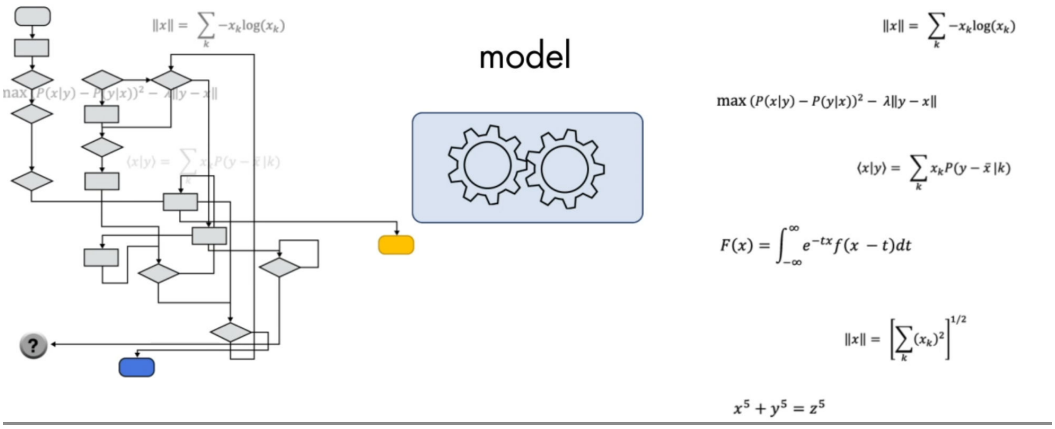
Machine Learning in a nutshell

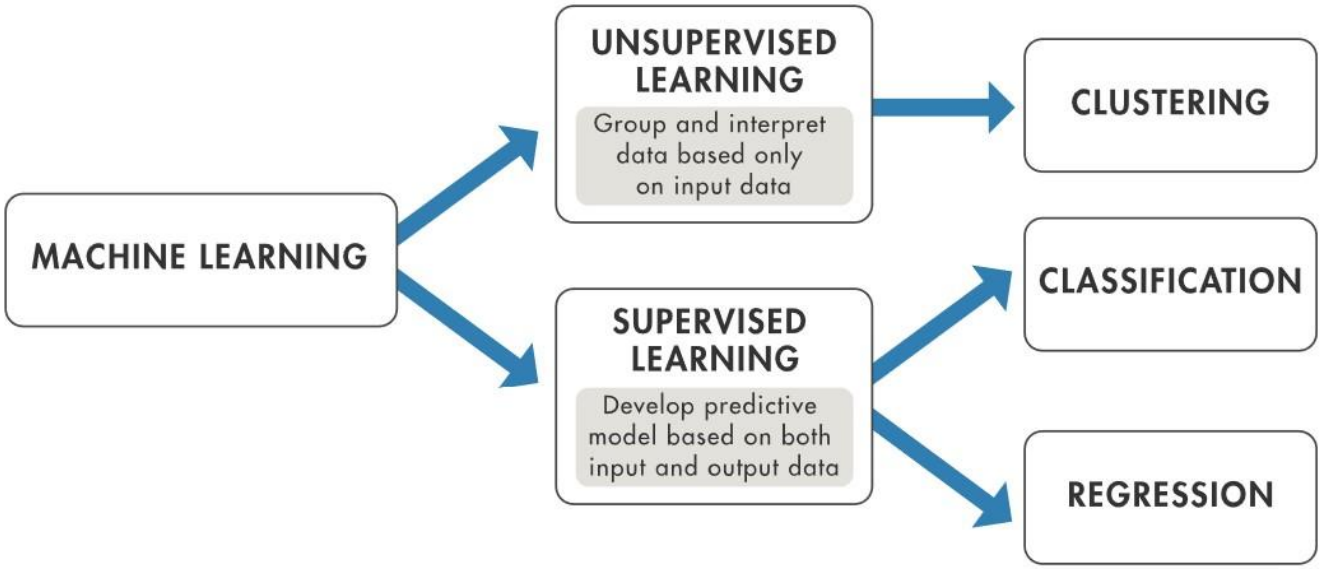
Machine Learning: extracting information on the underlying process generating the data, directly from implicit patterns in the data.

Machine Learning


Traditional approach





Some common scenarios where machine learning applies include:

- When a system is too complex for handwritten rules, like in face and speech recognition.
- When the rules of a task are constantly changing, as in fraud detection.
- When the nature of the data itself keeps changing, like in automated trading, energy demand forecasting, and predicting shopping trends.

Machine learning techniques are often used for financial analysis and decision-making tasks such as accurate forecasting, classification of risk, estimating probabilities of default, and data mining. However, implementing and comparing different machine learning techniques to choose the best approach can be challenging.

Find resources and examples in:

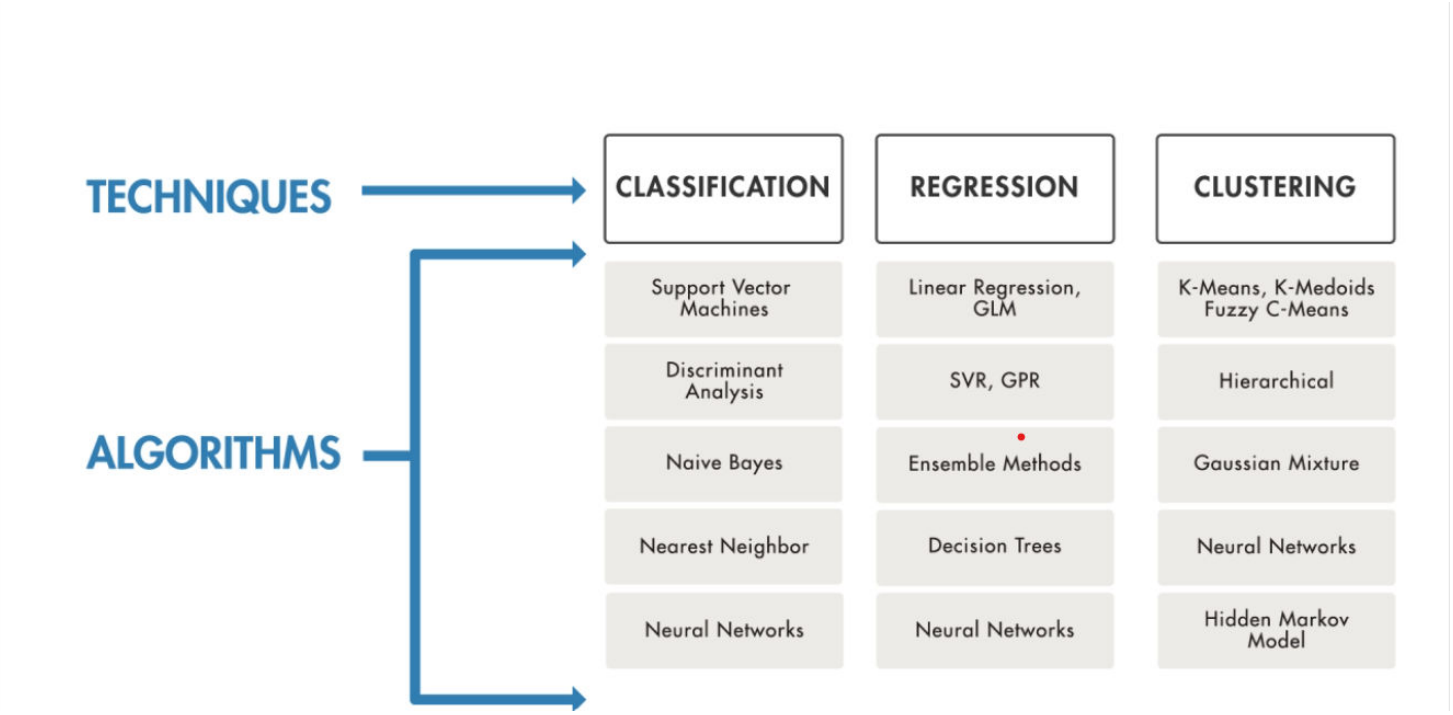
https://uk.mathworks.com/videos/series/introduction-to-machine-learning.html

Example of use in the industry (the Two-Sigma MIT spin-off).

https://www.twosigma.com/

https://www.twosigma.com/insights/article/iclr-2019-our-favorite-machine-learning-talks-and-papers/

https://www.twosigma.com/insights/article/two-sigma-factor-lens-forecasting-factor-returns/



Machine learning is synonymous with non-parametric modeling techniques.

The term non-parametric is not meant to imply that such models completely lack parameters but that the number and nature of the parameters are flexible and determined from data.

We will go through *supervised learning* by labelling the data as buy, sell and hold.

To make predictions, we have to follow the workflow:

use new data, preprocess them, send preprocessed data into the model and get the predictions.

Data (Ask, Bid, Mid)

Ask: lowest rate that someone in the market is willing to sell you the currency

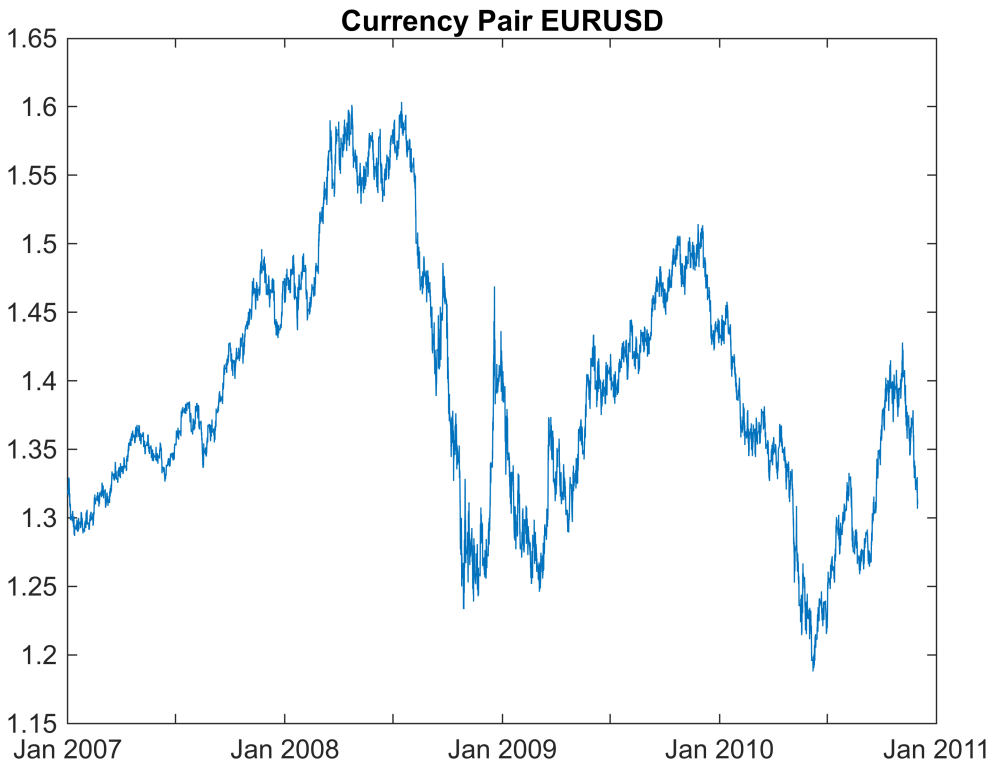
Bid: highest rate that someone is willing to buy the currency from you

Mid market rate is average of the bid and ask rates

```
currPair = 'EURUSD';   %create a character array

load EURUSD.mat
```

Visualization

```
figure

plot(prices.Time , prices.Mid)% plot the Mid, for example

title(['Currency Pair ' currPair]); %add a title

box on; %displays the box outline around the axes
```



Calculate predictor/response

Generate the predictor/response tables (or regressors/regressand) which we can then use for classical linear regression or machine learning.

I

N-Minute returns (over various window lengths)

MACD Moving Average Convergence/Divergence

RSI Relative Strength Index (over various window lengths)

Murphy, John J. *Technical Analysis of the Futures Market.* New York Institute of Finance, 1986, pp. 295–302.

MACD

One of the most popular methods of Technical Analysis is the MACD, Moving Average Convergence Divergence, indicator. The MACD uses exponentially smoothed averages to identify a trend reversal or a continuation of a trend.

The indicator, which was developed by Gerald Appel in 1979, reduces to two averages. The first, called the MACD1 indicator, is the difference between two exponential averages, usually a 26-day EMA (Exponential Moving Average)and a 12-day average.

The second, called Signal indicator, is the 9-day moving average of the MACD1 indicator.

The terms "convergence" and "divergence" refer to a narrowing and widening of the difference of the MACD1 and the Signal indicator:

A buy signal is given, when the more volatile average, the MACD1 indicator, crosses the less volatile average, the Signal indicator, from beneath. If the MACD1 line crosses the Signal line from above, a sell signal is given. The bigger the angle of the crossing, the more significant the buy or sell signal is.

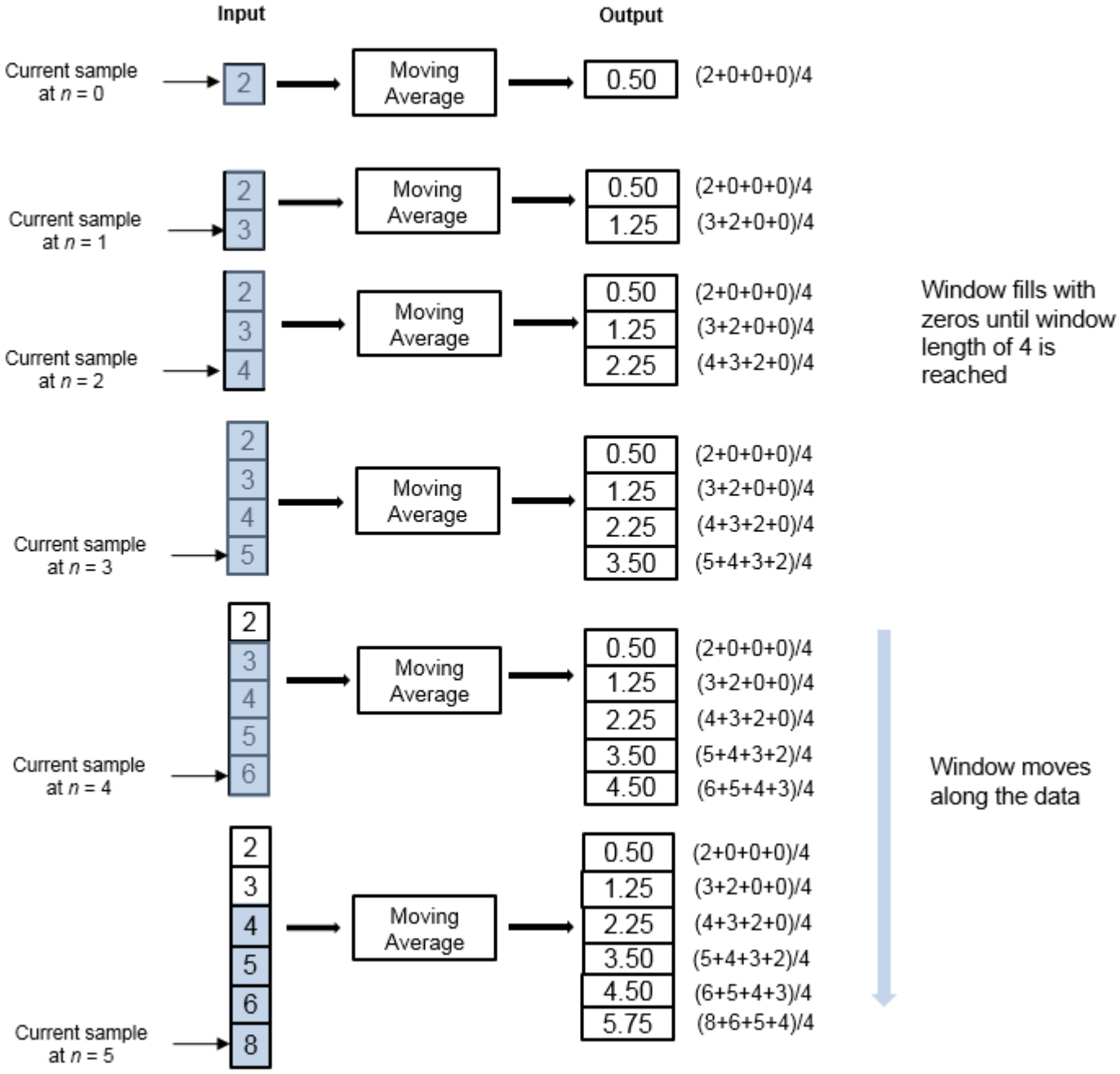
How to compute (see the example in Ex_MACD)

You can compute the moving average of a streaming input data using the sliding window method.

In the sliding window method, a window of specified length is moved over the data, sample by sample, and the average is computed over the data in the window.

Consider an algorithm that uses a window length of 4. At the first time step, the algorithm fills the window with three zeros to represent the first three samples. In the subsequent time steps, to fill the window, the algorithm uses samples from the previous data frame. The moving statistic algorithms have a state and remember the previous data.

Consider an example of computing the moving average of a streaming input data using the sliding window method. The algorithm uses a window length of 4. With each input sample that comes in, the window of length 4 moves along the data.



The window is of finite length, making the algorithm a finite impulse response filter. To analyze a statistic over a finite duration of data, use the sliding window method.

Effect of Window Length

The window length defines the length of the data over which the algorithm computes the statistic. The window moves as the new data comes in. If the window is large, the statistic computed is closer to the stationary statistic of the data. For data that does not change rapidly, use a long window to get a smoother statistic. For data that changes fast, use a smaller window.

Exponential Weighting Method

In the exponential weighting method, the object multiplies the data samples with a set of weighting factors. The average is computed by summing the weighted data.

In other words, **the recent data has more influence on the statistic** at the current sample than the older data.

MACD helps investors in understanding whether bullish or bearish movement in the price is strenghtented.

Relative Strength index

RSI is a famous indicator developed by John Welles Wilder Jr. in 1978 and used to find the phases of overselled/overbought (prices have risen too far in relation to their real value).

It is an indicator of momentum measuring the speed of prices movements; it oscillates on a scale from 0 to 100.

How to compute (see the example in Ex_RSI)

RSI is calculated by dividing the average of the gains by the average of the losses within a specified period.

```
RS = (average gains) / (average losses)
```

For the calculation of the average of the positive prices the total of the positive variations in the interested phase is summed.

RSI= 100-100/(1+(Gp/Gn)

with

Gp # positive days

Gn # negative days

or

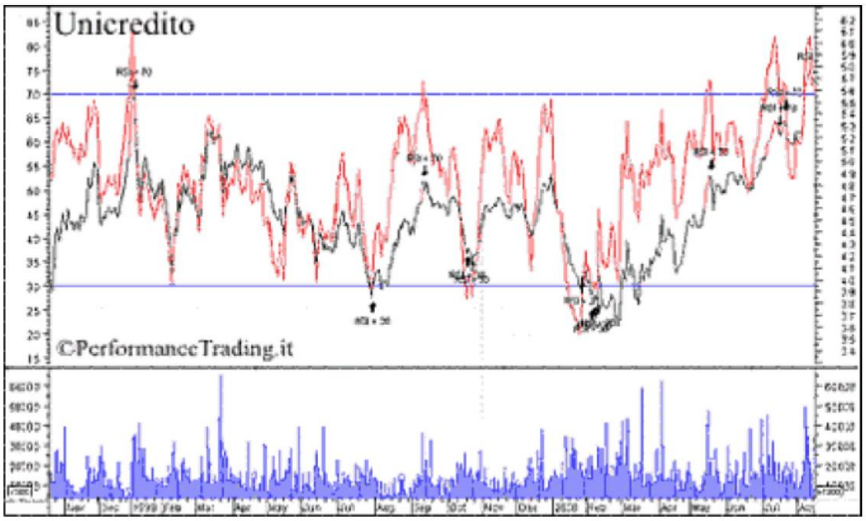
RSI= 100-100/(1+(GpM/GnM)

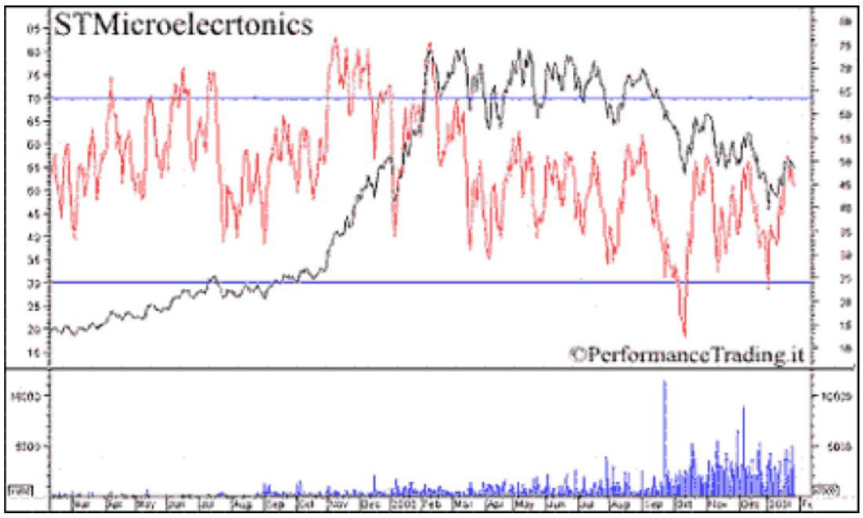
with

GpM average of positive days

GnM average of negative days

Over 70 there is the overbought phase and oversell under 30 points.





N-Minute returns (over various window lengths)

Defining the trading frequency (minutes)

```
tradingFrequency = 60;
```

Define the n-minute returns that we want to use to build the model

```
nMinReturns = [5 10 15 20 25 30 60];
```

Use the toolbox to get the technical indicators

MACD technical indicator

```
otherSignals = [];
otherSignals(1).fnHand = 'macd';
otherSignals(1).params = [];
otherSignals(1).field  = 'Mid';
```

RSI

```
for i = [5 10 15 20 25 30 60]% i=1:10
    otherSignals(end+1).fnHand = 'rsindex'; %#ok<SAGROW>
    otherSignals(end).params   = i;
```

```
    otherSignals(end).field    = 'Mid';
end
```

Now, we calculate the predictors X (timetable) and the response y which is the future returns (one hour in advance). The predictors are 15 factors.

```
[X , y] = fxAlgoMakeSignals(prices , nMinReturns , otherSignals ,
tradingFrequency , true);
```

```
03-Nov-2023 12:08:06    Creating 5 Minute Returns
03-Nov-2023 12:08:06    Creating 10 Minute Returns
03-Nov-2023 12:08:06    Creating 15 Minute Returns
03-Nov-2023 12:08:06    Creating 20 Minute Returns
03-Nov-2023 12:08:06    Creating 25 Minute Returns
03-Nov-2023 12:08:06    Creating 30 Minute Returns
03-Nov-2023 12:08:06    Creating 60 Minute Returns
03-Nov-2023 12:08:06    Creating factor macd
03-Nov-2023 12:08:07    Creating factor rsindex 5
03-Nov-2023 12:08:07    Creating factor rsindex 10
03-Nov-2023 12:08:07    Creating factor rsindex 15
03-Nov-2023 12:08:07    Creating factor rsindex 20
03-Nov-2023 12:08:07    Creating factor rsindex 25
03-Nov-2023 12:08:07    Creating factor rsindex 30
03-Nov-2023 12:08:07    Creating factor rsindex 60
03-Nov-2023 12:08:08    Calculating the 60 minute future return
03-Nov-2023 12:08:09    Downsampling the data to 60 minute bar
03-Nov-2023 12:08:10    Finished calculating factors and responses
```

Get an in-Sample and train the model

The in-sample range we want to take is two months worth of data

Select two month of data - to do this, let's use the timerange object which allows us to window into our timetable to select the time period we want.

Use the timetable structure to easily window into our data, by using the built-in functione timerange, which allows to set a time span, and window into the X and y matrices

```
tr = timerange('2007-01-01' , '2007-02-28'); %windowing into the data
```

For the regression methodology we need to change from timetable into a table

```
XInSample = timetable2table(X(tr , 4:end) , 'convertrowtimes' , false);
yInSample = timetable2table(y(tr , :) , 'convertrowtimes' , false);
tIn = X.Time(tr);
```

Train a linear model

```
modelTrain = fitlm([XInSample yInSample] , 'linear')
```

```
modelTrain =
Linear regression model:
    FutureReturn ~ 1 + Return_5 + Return_10 + Return_15 + Return_20 + Return_25 + Return_30 + Return_60 + macd + rsi

Estimated Coefficients:
                    Estimate          SE         tStat       pValue
                   __________    __________    ________    _________

    (Intercept)      0.0007597    0.00078932     0.96247      0.33605
    Return_5          0.076187       0.31856     0.23916      0.81103
    Return_10           0.1205       0.35748     0.33708      0.73613
    Return_15         -0.39926        0.3824     -1.0441      0.29669
    Return_20           1.2419       0.43161      2.8772    0.0040978
    Return_25         -0.85986       0.41231     -2.0854     0.037284
    Return_30         0.099041       0.27449     0.36082      0.71831
    Return_60        -0.034475       0.15454    -0.22308      0.82352
    macd               0.31829       0.83231     0.38242      0.70224
    rsindex_5       8.4148e-07    2.9315e-06     0.28705      0.77413
    rsindex_10     -5.5039e-06    7.0567e-06    -0.77996       0.4356
    rsindex_15      1.161e-05    1.1886e-05     0.97678      0.32892
    rsindex_20     -4.0866e-05     1.701e-05     -2.4024      0.01647
    rsindex_25      3.1297e-05    2.1001e-05      1.4902      0.13648
    rsindex_30     -1.5765e-05    1.6681e-05    -0.94508      0.34485
    rsindex_60      3.1985e-06    1.9634e-05     0.16291      0.87062


Number of observations: 1007, Error degrees of freedom: 991
```

```
Root Mean Squared Error: 0.000753
R-squared: 0.0248,  Adjusted R-Squared: 0.0101
F-statistic vs. constant model: 1.68, p-value = 0.0488
```

Let's call the predict method on the in-sample data

Run the in-sample prediction

```
retPredictionRegress = predict(modelTrain , XInSample)
```

```
retPredictionRegress = 1007×1
10^-3 ×
    0.0119
    0.0568
    0.0395
    0.0936
   -0.0726
   -0.0695
   -0.0064
   -0.0488
    0.0022
   -0.0316
      ⋮
```

So, we get return prediction for our dataset, given the model that we just fitted

Look for the effectiveness of the prediction by adopting a back-testing strategy.

We calculate the returns by simply multiplying the actual returns by the sign of the predicted returns, then apply a cumprod to generate a returns curve

Trading with the model: let's take a very indiscriminate strategy (realistically, we would put in some minimum return before we actually trade): whenever the returns prediction is greater than 0, go long, and whenever the returns prediction is less then 0, go short

```
positions = zeros(size(retPredictionRegress))
```

```
positions = 1007×1
     0
     0
```

```
     0
     0
     0
     0
     0
     0
     0
     0
     ⋮
```

```
positions(retPredictionRegress > 0) = 1
```

```
positions = 1007×1
     1
     1
     1
     1
     0
     0
     0
     0
     1
     0
     ⋮
```

```
positions(retPredictionRegress < 0) = -1
```

```
positions = 1007×1
     1
     1
     1
     1
    -1
    -1
    -1
    -1
     1
    -1
     ⋮
```

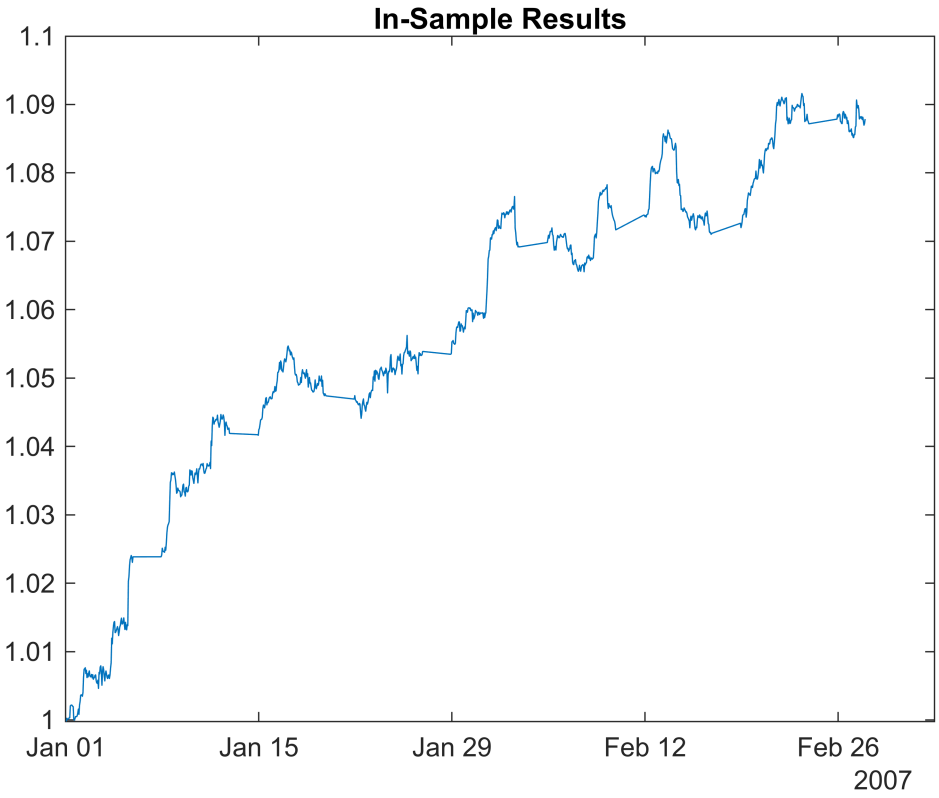
Save the data, both predictors and response, in a spredsheet

```
 table_ML=[XInSample, yInSample];

 writetable(table_ML,'table_ML.xlsx');
```

Backtesting in-sample

To backtest the strategy, I have to take the in-sample returns multiplied by the positions, apply cumulative product and get a returns graph

```
actualReturns = positions .* yInSample{:,1};
inSampleRegressionReturns = cumprod(1+actualReturns);

 f1 = figure('tag' , 'insamplefigure');
plot(tIn , inSampleRegressionReturns);
title('In-Sample Results');
```



We get graph for the in-sample returns of the window of two months.

The in-sampe result are very good for a in-sample of two-month period.

We can test the model on the out-sample as well.

Backtesting out-sample

Take our predictive model and apply it to the out sample - in this case we are taking an out of sample to be one month and run a similar process as before:

```
tr = timerange('2007-03-01' , '2007-03-31');
XOutSample = timetable2table(X(tr , 4:end) , 'convertrowtimes' , false);
yOutSample = timetable2table(y(tr , :) , 'convertrowtimes' , false);
tOut = X.Time(tr);
```

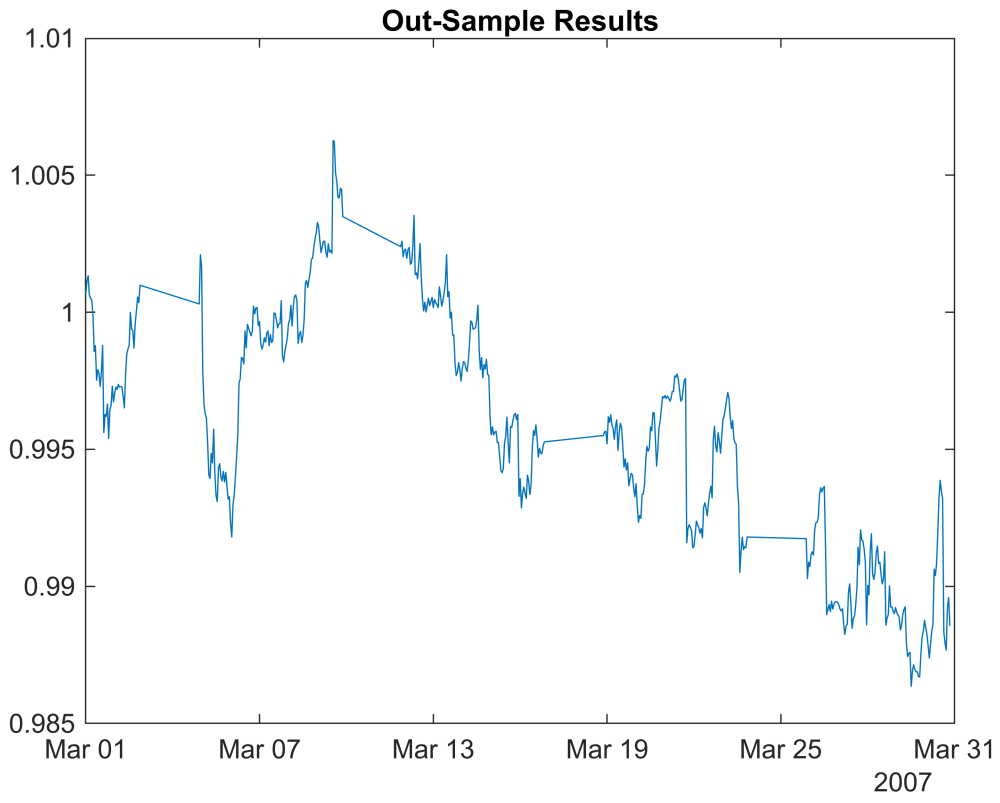
run the prediction

```
retPred = predict(modelTrain , XOutSample);
```

```
positions=zeros(size(XOutSample,1), 1);
positions(retPred > 0)=1;
positions(retPred < 0)=-1;
```

calculate the returns

```
actualReturns = positions .* yOutSample{:,1};
outSampleRegressionReturns = cumprod(1 + actualReturns);

f2 = figure('tag' , 'outsamplefigure');
plot(tOut , outSampleRegressionReturns)
title('Out-Sample Results');
```



Repeat this process using a stepwise

Using the *stepwiselm* function, we can re-run this and allow the stepwise algorithm to discard terms and retain only those predictors with the most predictive capability:

```
modelStepwise = stepwiselm([XInSample yInSample], 'linear' , 'upper' , 'linear')
```

```
1. Removing rsindex_60, FStat = 0.026539, pValue = 0.87062
2. Removing Return_60, FStat = 0.040557, pValue = 0.84044
3. Removing Return_5, FStat = 0.032114, pValue = 0.85781
4. Removing macd, FStat = 0.078829, pValue = 0.77895
5. Removing Return_10, FStat = 0.23527, pValue = 0.62775
6. Removing Return_30, FStat = 0.32967, pValue = 0.56598
7. Removing rsindex_5, FStat = 0.72464, pValue = 0.39483
8. Removing rsindex_10, FStat = 0.64967, pValue = 0.42042
9. Removing rsindex_15, FStat = 0.53786, pValue = 0.46349
10. Removing Return_15, FStat = 0.5122, pValue = 0.47435
11. Removing rsindex_30, FStat = 1.758, pValue = 0.18517
12. Removing rsindex_25, FStat = 1.7088, pValue = 0.19144
modelStepwise =
Linear regression model:
    FutureReturn ~ 1 + Return_20 + Return_25 + rsindex_20

Estimated Coefficients:
                    Estimate          SE         tStat       pValue
                   __________    __________    _______    __________

    (Intercept)     0.00096483    0.00037517     2.5717      0.010263
    Return_20          0.70075       0.21252     3.2973     0.0010105
    Return_25          -0.3597       0.10408    -3.4559    0.00057144
    rsindex_20     -1.9225e-05    7.4912e-06    -2.5664      0.010421


Number of observations: 1007, Error degrees of freedom: 1003
Root Mean Squared Error: 0.000751
R-squared: 0.0183,  Adjusted R-Squared: 0.0154
F-statistic vs. constant model: 6.25, p-value = 0.000334
```

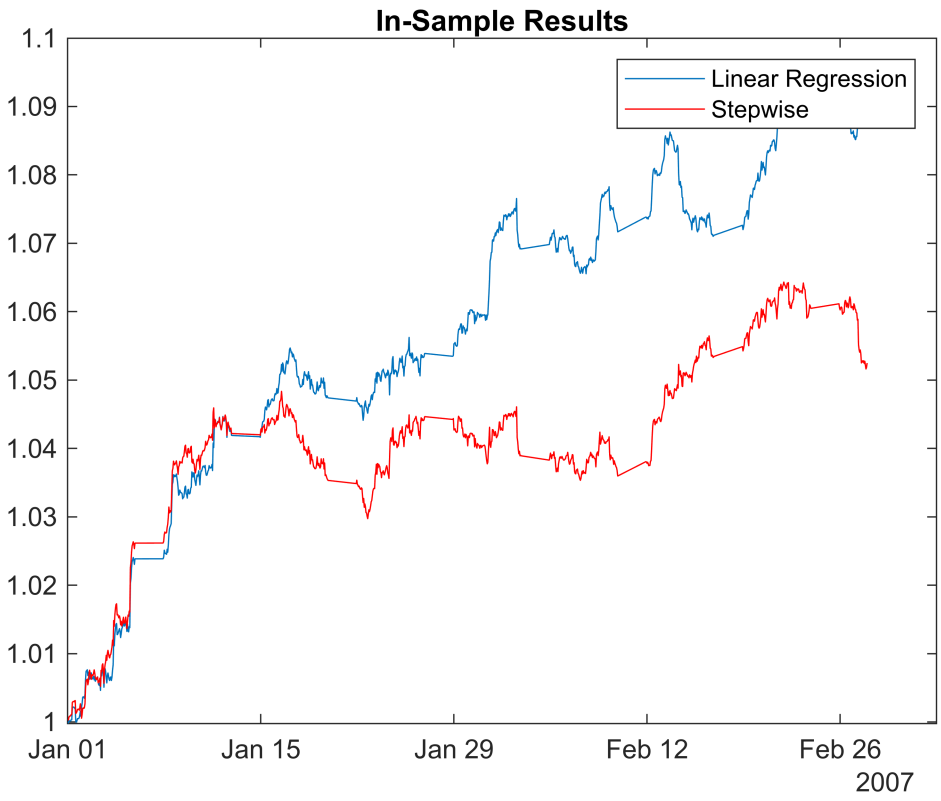
The algorithm removed all those factors that are less statistically significant, returning a more compact model.

Once again, use the predict method on the in-sample

Predict In-Sample results

```
retPrediction = predict(modelStepwise , XInSample);

positions = zeros(size(retPrediction));
positions(retPrediction > 0) = 1;
positions(retPrediction < 0) = -1;
```

```
actualReturns = positions .* yInSample{:,1};
inSampleStepwiseReturns = cumprod(1+actualReturns);


figure(f1);
hold on
plot(tIn , inSampleStepwiseReturns , 'r');
legend({'Linear Regression' , 'Stepwise'});
```

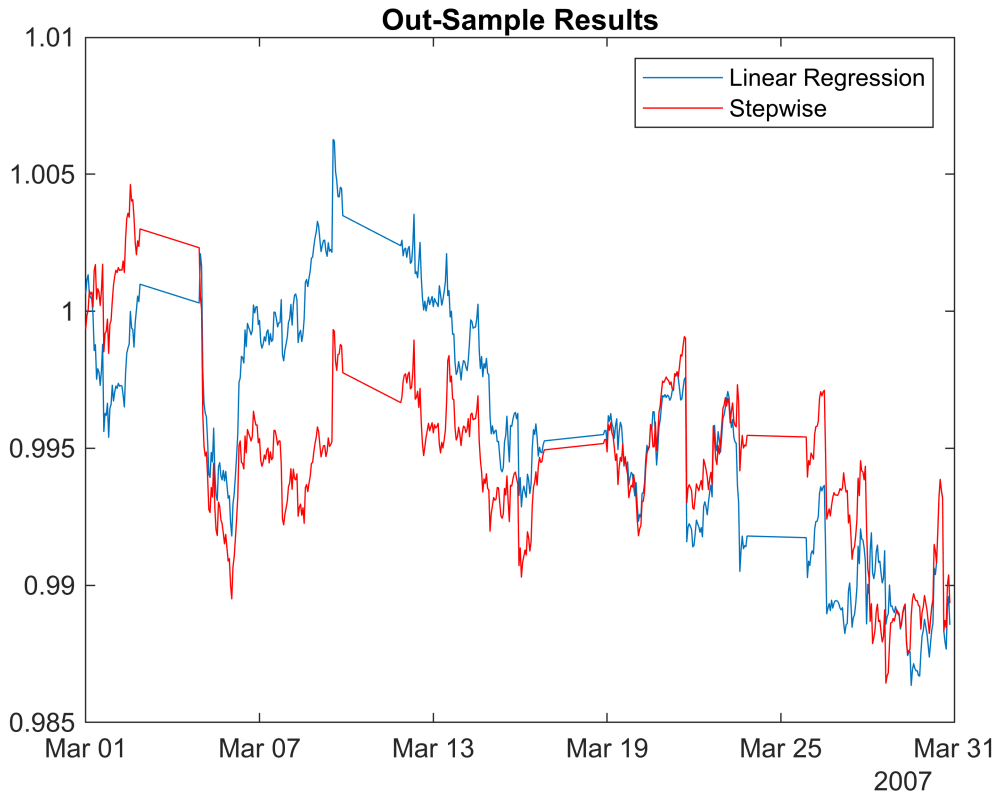



```

```

Run for out of sample

```
retPred = predict(modelStepwise , XOutSample);
positions=zeros(size(retPred,1), 1);
positions(retPred > 0)=1;
positions(retPred < 0)=-1;

actualReturns = positions .* yOutSample{:,1};

outSampleStepwiseReturns = cumprod(1 + actualReturns);
```

```
figure(f2)
hold on
plot(tOut , outSampleStepwiseReturns , 'r')
legend({'Linear Regression' , 'Stepwise'});
hold off
```



Machine Learning approach

Use a ML supervised algorithms by labelling future returns as buy/sell/hold.

We label by using the *median bid/offer spread* (in pips) and classify as

+1 if the future return is positive and it is greater than the spread (i.e. go long)

-1 if the future return is negative and it is greater than the spread (go short)

0 in all other cases

(Pip is an acronym for "percentage in point". A pip is the smallest price move that an exchange rate can make based on forex market convention. Most currency pairs are priced out to four decimal places and the pip change is the last (fourth) decimal point. A pip is thus equivalent to 1/100 of 1%)

Data subset

Prepare the data as before. Take our factor table and take a subset for training purposes using a timerange object

```
tr = timerange('2007-01-01' , '2007-02-28');
predictorInSample = timetable2table(X(tr , :) , 'ConvertRowTimes' , false);
responseInSample  = timetable2table(y(tr , :) , 'ConvertRowTimes' , false);
tIn = X.Time(tr);
```

Take the factors and create the predictor table

```
predictorTable = predictorInSample(:,4:end);
clear('predictorInSample');
```

Labelling the responses

Let's categorise the responses as buy, sell or hold. We will use the value of the return to classify the responses in our supervised learning problem.

We want to classify the problem into three different states. We will use the median bid/offer spread to define a minimum predicted return at which we wouldn't trade.

```
minRet = nanmedian(prices.Ask - prices.Bid);% Median value, ignoring NaNs
```

Let's round this UP to the nearest pip...

```
minRet = roundUpTo(minRet , 0.0001);
```

and classify the responses

```
futRet = responseInSample.FutureReturn;
```

Filter by min return

```
response = zeros(size(futRet))% set the matrix as zeros (label for hold)
```

```
response = 1007×1
     0
```

```
     0
     0
     0
     0
     0
     0
     0
     0
     0
     ⋮
```

```
response(futRet > minRet) = 1 %label to go long
```

```
response = 1007×1
     0
     1
     0
     0
     0
     0
     0
     0
     0
     0
     ⋮
```

```
response(futRet < -minRet) = -1% label to go short
```

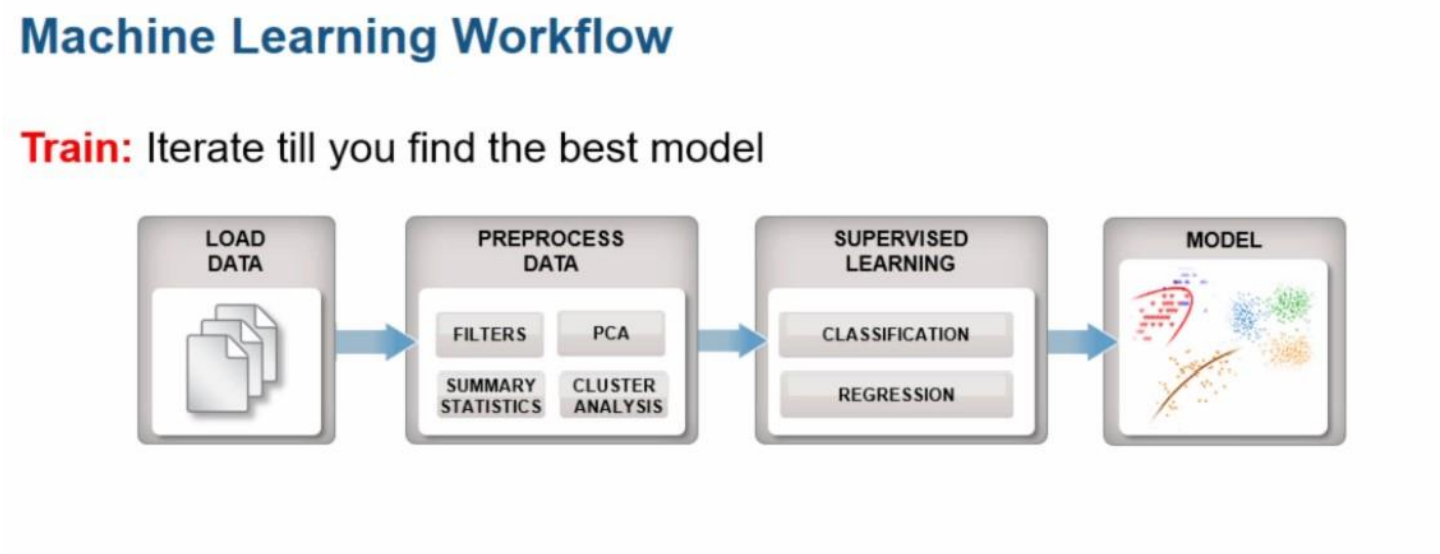
```
response = 1007×1
    -1
     1
    -1
     0
     0
    -1
     0
    -1
     0
     0
     ⋮
```

Add the response to the table

```
predictorTable.response = response;
```

So, now in the last column we have our labels (-1,0,1).

This is the data that we send to a Machine learning algorithm



You can run the dataset through a series of different models using the Regression Learner app.

Open the classification app.

Start a new session. We will choose the predictor table and set the response as buy/sell/hold. We will use cross-validation that randomly goes through the data .

From the list of variables, make sure you choose the variable predictorTable. Set the response variable to be the response.

You will get a scatter plot coloured by the classes:

blu for sell, red for hold and yellow for buy.

We do not observe much pattern. You can check for plotting different varables.

We can train on a series of models (Tree (Complex, Medium, Simple), KNN (Fine, Medium, Coarse).

Classification Tree

Decision Trees

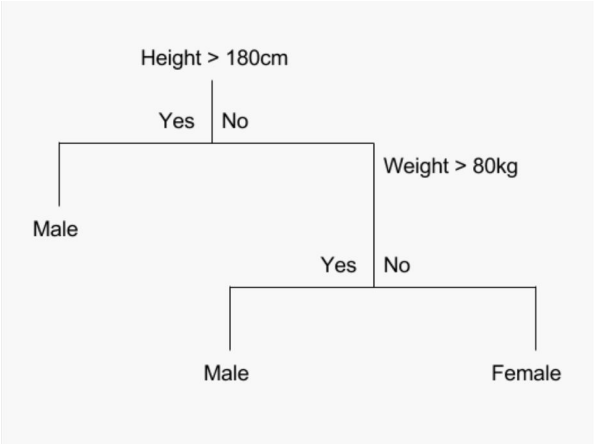
Classification and Regression Trees or CART for short is a term introduced by Leo Breimanto refer to algorithms that can be used for classification or regression predictive modeling problems.

Classically, this algorithm is referred to as “decision trees”.

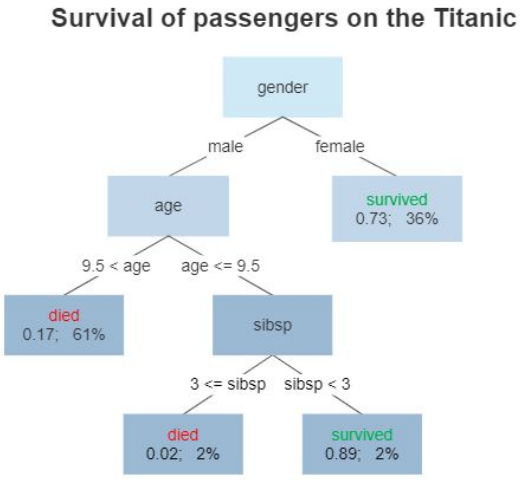
Classification trees and regression trees are two kinds of decision trees. A decision tree is a flow-chart like structure in which internal node represents test on an attribute, each branch represents outcome of test and each leaf node represents a response (decision taken after computing all attributes).

Algorithms for constructing decision trees usually work top-down, by choosing a variable at each step that best splits the set of items.

Classification trees give responses that are nominal.



Regression trees give numeric responses.



Amongst other data mining methods, decision trees have various advantages:

- **Simple to understand and interpret.** People are able to understand decision tree models after a brief explanation. Trees can also be displayed graphically in a way that is easy for non-experts to interpret.
- **Able to handle both numerical and categorical data.**Other techniques are usually specialized in analyzing datasets that have only one type of variable.
- **Requires little data preparation.** Other techniques often require data normalization.
- **Possible to validate a model using statistical tests.** That makes it possible to account for the reliability of the model.
- Non-statistical approach that makes no assumptions of the training data or prediction residuals; e.g., no distributional, independence, or constant variance assumptions
- **Performs well with large datasets.** Large amounts of data can be analyzed using standard computing resources in reasonable time.
- **Mirrors human decision making more closely than other approaches.**This could be useful when modeling human decisions/behavior.

Let's choose one of the methods. In this case, we aren't choosing the best performing method, but we choose one that's easy to interpret and explain, namely the classification tree. Use as input the predictor table and the predicted response and fit the model.

```
cTree = fitctree(predictorTable(:,1:end-1), predictorTable(:,end), ...
    'SplitCriterion', 'gdi', 'MaxNumSplits', 4,'Surrogate', 'off', 'ClassNames',
[-1; 0; 1]);

% returns a fitted binary classification decision tree based on the input variables
% (predictors or features) and output (response or labels).The returned binary
% tree splits branching nodes based on the values of a column of Tbl.
```

Test the results in sample

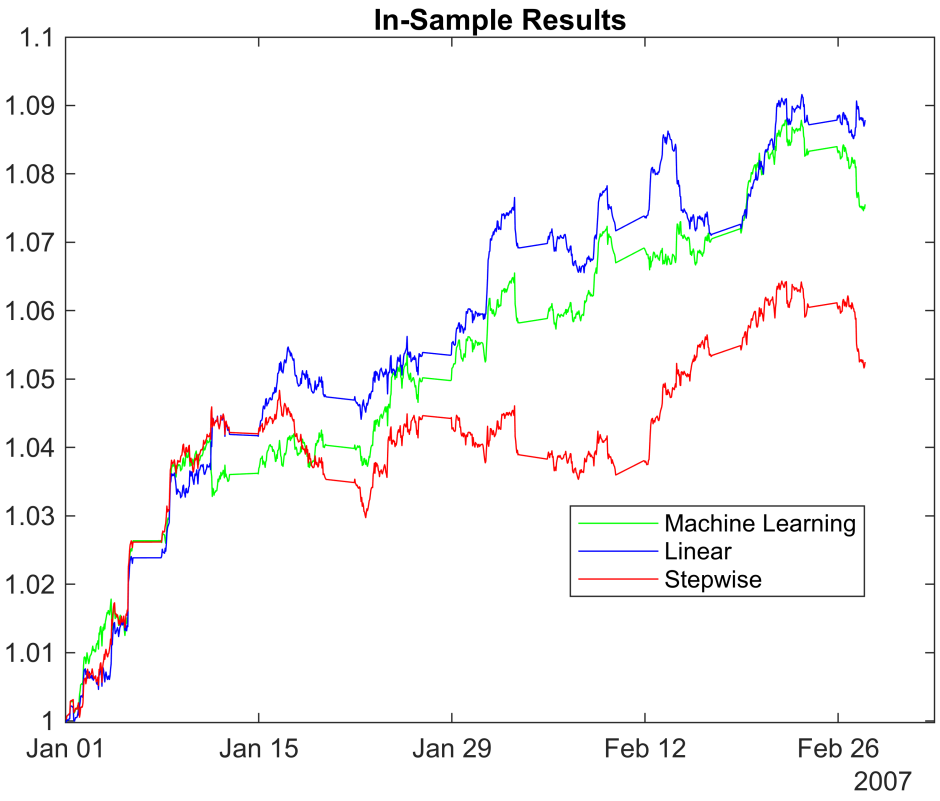
We send the original predictor table into the classification tree to get the predicted returns, and take a position long/short when the predicted class is non-zero.

```
retPredictionML = cTree.predict(predictorTable(:,1:end-1));

positions = zeros(size(futRet));
positions(retPredictionML > 0) = 1;
positions(retPredictionML < 0) = -1;
```

```
actualReturns = positions .* futRet;
inSampleRegressionReturns_ML = cumprod(1+actualReturns);
```

Let's plot the results for the three techniques:

```
plot(tIn , inSampleRegressionReturns_ML, 'g');
title('In-Sample Results');
hold on
plot(tIn , inSampleRegressionReturns,'b');

plot(tIn , inSampleStepwiseReturns , 'r');
legend('Machine Learning','Linear','Stepwise','Location','best')
hold off
```



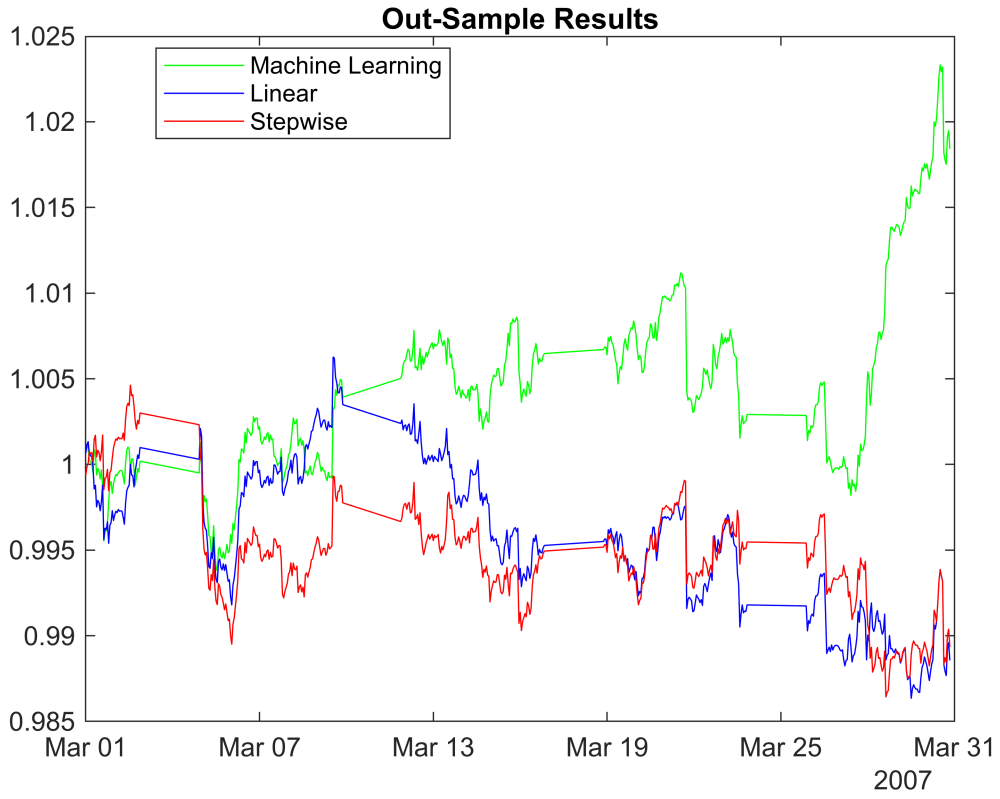
Out-sample

Repeat the process of generating the out-sample subset using a timerange object, and sending this into the classification tree to get a prediction for future returns, which we then apply against the true future returns.

On charting it, we see an improvement in the results for the machine learning technique, and we will continue with both the regression and the machine learning technique

```
tr = timerange('2007-03-01' , '2007-03-31');
predictorOutSample = timetable2table(X(tr , :) , 'ConvertRowTimes' , false);
responseOutSample  = timetable2table(y(tr , :) , 'ConvertRowTimes' , false);
tOut = X.Time(tr);
futRet = responseOutSample.FutureReturn;

% Take the factors and create out predictor table
predictorTable = predictorOutSample(:,4:end);

% Run our prediction of our classification tree
retPredictionML = cTree.predict(predictorTable(:,1:end));

% Gather the returns
positions = zeros(size(futRet));
positions(retPredictionML > 0) = 1;
positions(retPredictionML < 0) = -1;

actualReturns = positions .* futRet;
outSampleRegressionReturns_ML = cumprod(1+actualReturns);
```

```
plot(tOut , outSampleRegressionReturns_ML , 'g');
title('Out-Sample Results');
hold on
plot(tOut , outSampleRegressionReturns,'b');

plot(tOut , outSampleStepwiseReturns , 'r');
legend('Machine Learning','Linear','Stepwise','Location','best')
hold off
```




```
function [X , y] = fxAlgoMakeSignals(price , nMinReturns , otherSignals ,
nMinFutReturns , downSample)

allFactors = {};
factorNames = {};

for i = 1:length(nMinReturns)
    logCommand(sprintf('Creating %d Minute Returns' , nMinReturns(i)));%Write
formatted data to string and print on screen

    allFactors{i} = nMinuteReturn(price , nMinReturns(i)); %#ok<*AGROW>
    factorNames{i} = ['Return_' num2str(nMinReturns(i))];
end
factorCount = length(allFactors) + 1;

% Add in the other signals?
for i = 1:length(otherSignals)

    fHand = str2func(otherSignals(i).fnHand);
    data  = price.(otherSignals(i).field);

    if ~isempty(otherSignals(i).params)
```

```
        logCommand(sprintf('Creating factor %s %d' , otherSignals(i).fnHand ,
otherSignals(i).params));
        allFactors{factorCount} = fHand(data , otherSignals(i).params);
        factorNames{factorCount} = [otherSignals(i).fnHand '_'
num2str(otherSignals(i).params)];
    else
        logCommand(sprintf('Creating factor %s' , otherSignals(i).fnHand));
        allFactors{factorCount} = fHand(data);
        factorNames{factorCount} = otherSignals(i).fnHand;
    end
    factorCount = factorCount + 1;

end

% Create a new timetable
signal = timetable(price.Time , allFactors{:} , 'VariableNames' , factorNames);

% Merge with the other results
% synchronize provides a way to, in effect, horizontally concatenate two or
%     more timetables even when their row times are different. You can retain
%     all the times from the input timetables in the output, or synchonize one
%     timetable to another, or specify a completely new vector of row times for the
%     output. synchronize also provides several ways to adjust each timetable's
%     data to account for aligning to a different vector of row times.

price = synchronize(price , signal);

% Create the future return
logCommand(sprintf('Calculating the %d minute future return' , nMinFutReturns));

futReturns = nMinuteReturn(price , nMinFutReturns);% you can use nested functions

% Stagger
futureReturn = NaN(size(futReturns,1) , 1);

futureReturn(1:end-nMinFutReturns) = futReturns(nMinFutReturns+1:end);

% Create a timetable for y
y = timetable(price.Time , futureReturn , 'VariableNames' , {'FutureReturn'});

price = synchronize(price , y);

% Downsample, i.e. changing the frequency rate
logCommand(sprintf('Downsampling the data to %d minute bar' , nMinFutReturns));
if downSample
    price = price(1:nMinFutReturns:end,:);
end
price = rmmissing(price);

% Separate
```

```
X = price(:,1:end-1);
y = price(:,end);
logCommand('Finished calculating factors and responses');
end


function out = roundUpTo(num , roundTo)


out = ceil(num ./ roundTo) .* roundTo;
end


function logCommand(str)


fprintf('%s\t%s\n' , datestr(now , 0) , str);
end



function returns = nMinuteReturn(tt , nMins)


data = tt.Mid;
returns = NaN(size(data,1),1);
returns(nMins+1:end) = data(nMins+1:end) ./ data(1:end-nMins) - 1;
end
```