The groundwork for machine learning

Machine learning in a nutshell

We formulate our stock selection model as a binary classification problem. We classify the stocks in our universe into two categories: outperformers and underperformers based on one-month forward stock returns. Outperformers are those stocks with highest one-month forward returns and underperformers are those with lowest one-month forward returns

We construct classifiers based on whether the stock is going to outperform or underperform. The inputs of the classifiers are our stock selection factors, and the outputs are confidence scores. A positive confidence score means we expect the stock to outperform. The higher the confidence score is, the more likely the stock will outperform and vise versa.

The construction of the confidence score has two steps. First is the training step. In this step, we use the end-of-month factor scores for each stock and one-month forward returns as training data to build the classifiers. Essentially, the input vector is the stock factor score and the output label is a binary label of outperformer or underperformer. The power inherent within machine learning methods is the ability to find a predictive relationship between the input vector and the output label. The second step is the actual prediction step. In this step we use the current month factor score as the input for the classifiers we built in the training step, and the output is the confidence score.

Data preparation

Before we build our machine learning framework, we need to "normalize" the data to ensure that the training data used to build the classifiers are as consistent as possible. Most machine learning algorithms are very sensitive to input data. If the training data are not consistent, then the algorithm is more likely to be overfit. Data preparation is crucial to the machine learning algorithms. To some extent, normalizing the data can be even more important than choosing the machine learning algorithm.

Input factor data
We use the cross-sectional ranking of the factors, rather than the factor score itself as the input, and we calculate the factor ranking each month for all the available stocks. This is because what we care about is the relative ranking rather than the absolute returns. The factor score itself will vary from time to time, making the training data inconsistent. Furthermore, the factor score is very easily influenced by market regimes. For example, in a bearish market certain factors may have low values (e.g. price to earnings) for all the stocks. Whereas in a bullish market, factor scores (e.g., price to earnings) may have higher values. If we were to utilize the actual factor scores instead of the "ranked" factor scores, the training data would be inconsistent over time. In addition, we use the ranking of the factors rather than normalizing the factor score by the typical z-score since several factors are heavily skewed. Factor rankings would always produce an even distribution.

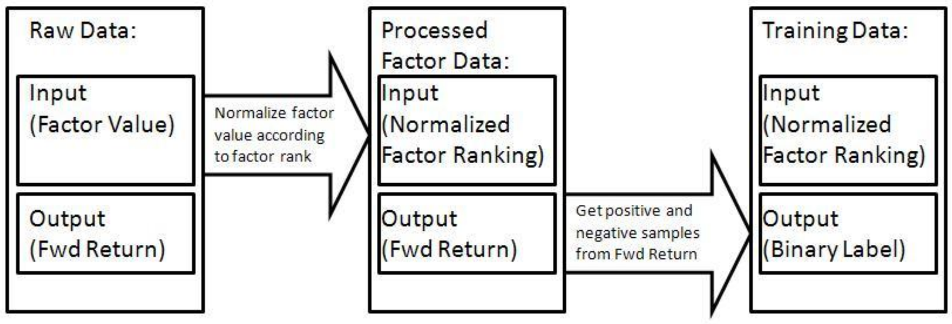
Once we rank each factor cross-sectionally, we divide the factor ranking by the number of stocks to normalize the factor ranking to between (0, 1]. The reason we normalize it between (0,1] is because coverage varies between factors and across time. Normalizing our factor ranking between (0,1] will enable us to be able to compare factors that have varying coverage in a consistent manner.

Labeling training data

After normalizing the factors by rank and coverage, we then assemble the training data. We label the training data using one-month forward returns. We label the top forward return stocks as outperformers, and the bottom forward return stocks as underperformers. We set the stocks in the top 30% as measured by one-month forward return as the outperformers. Similarly we set stocks in the bottom 30% as measured by one-month forward returns as the underperformers. We perform this labeling exercise on a monthly basis. Note that stocks not classified in the top or bottom 30% are disregarded. This is because stocks with insignificant positive or negative forward returns may just be noise, rather than true outperformers or underperformers. If we include those stocks in the training data we potentially reduce the accuracy of the classifiers.

Similar to the factor normalization, we only care about the relative performance; thus, we cross-sectionally label the stocks every month. This means in some good months, underperformers might even have a slightly positive forward return; and some bad months, outperformers might have a slightly negative forward return. In our strategy relative performance is more important, because we want to minimize the extrinsic effects that influence the market, since we want to long (or overweight) the relative good stocks and short (or underweight) the relative bad ones, so we don't care that much about the absolute performance. In addition, most of the binary classification problems in machine learning prefer similar number of outperformers and underperformers in the training data, and this way of collecting the training data will generate a similar number of outperformers and underperformers. Figure 4 illustrate how we prepare the training data.

Figure 4: Data preparation


Raw Data:
Input (Factor Value)
Output (Fwd Return)
Normalize factor value according to factor rank
Processed Factor Data:
Input (Normalized Factor Ranking)
Output (Fwd Return)
Get positive and negative samples from Fwd Return
Training Data:
Input (Normalized Factor Ranking)
Output (Binary Label)

Source: Deutsche Bank Quantitative Strategy

The AdaBoost algorithm

We use the machine learning algorithm called AdaBoost to build classifiers that can combine the best performing factors in previous months and get a confidence score. The higher the confidence score, the more likely the stock is going to outperform in the next month.

AdaBoost is a very effective machine learning method for classification; the main idea of AdaBoost is that it adaptively builds a sequence of classifiers that are constantly being tweaked to emphasize *misclassified* stocks, thereby slowly improving the classification of stocks that would normally be incorrectly classified. Although certain classifiers can be weak, as long as their performance is not random, the performance of the final model will improve.

In our case, a weak classifier is simply defined by a factor. We divide the factor into quantiles, and calculate the weights of outperformers and underperformers in each quantile. Intuitively, the most effective factors are those which have the largest difference between the weights of outperformers and underperformers in each quantile. Therefore, when the new data falls into a certain quantile, the label of the new data can be determined by the majority of the labels of the training data in that quantile. As such, the value of the weak classifier is determined by the weights of outperformers and underperformers in that quantile. The higher the weight of outperformers relative to the weights of underperformers in that quantile, the higher value the output of that weak classifier will be.

Dividing factors into quantiles is a natural extension of the decision tree (TREE model), in which case the quantile number equals two. By setting quantiles to more than two (in our case we set it to be five because setting this number too large increases the risk of overfitting) our model can much better capture the non-linear payoffs described in our recent paper, Cahan [2012a]. For example, for some factors, the middle quantiles show better performance than the top and bottom quantiles, in which case not only a linear model but also the TREE model would fail to capture the factor payoffs. For the TREE model, no matter where we set the threshold to split the node, the performance would not be satisfying. Therefore, the TREE model would need multiple levels of splits to capture a highly nonlinear pattern. However, in our model, the weak classifier would naturally learn and apply lower value for the top and bottom quantiles and higher value for the middle quantiles, since there are be more underperformers in the middle quantiles for this factor.

Initially, we equally-weighted each observation in the training data, and then the weights are updated in each round after a new weak classifier is found. The weight of each incorrectly classified stock is increased and the weight of each correctly classified stock is decreased, so that the next classifier would focus on the stocks which have so far not been correctly classified.

As discussed above, in each round we choose the most effective weak classifier, which can distinguish the underperformers and underperformers the most. Therefore, that corresponding factor is best performing factor with the current set of weights. Notice that the current best factor itself might not be the best performing factor; it is best performing in the sense that it performs best for those stocks that cannot be classified correctly by previous classifiers. This means the current best performing factors is less correlated with the previous selected factors. This is a good feature of the model, because many of the well performing factors are highly correlated, therefore, combining them can hardly improve the performance. What our machine learning system does is select those factors that are complementary to each other.

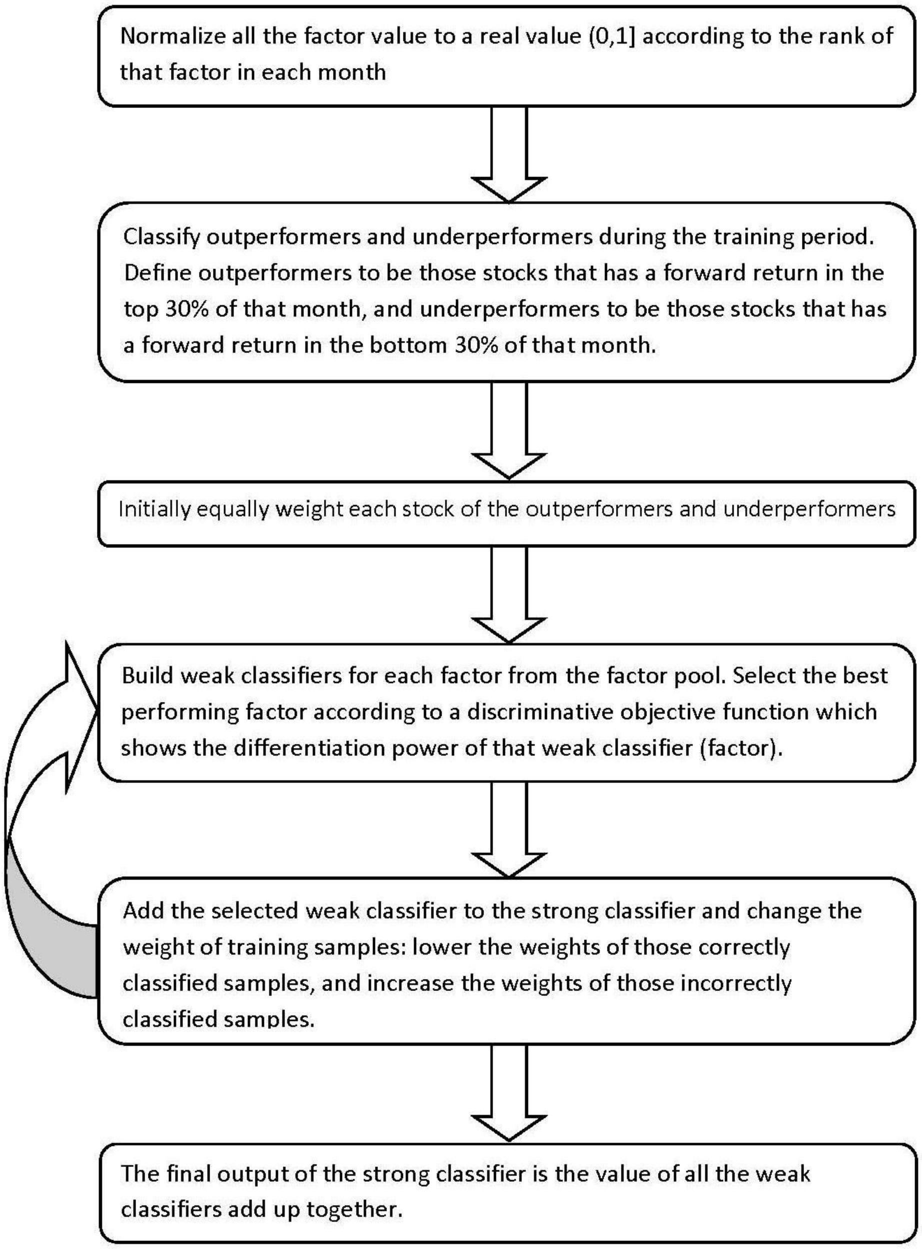
The output of the strong classifier is the sum of all the weak classifiers; it is a real value confidence score of how likely the stock is to be an outperformer. Essentially we can use this confidence score as a new composite factor, which hopefully has better performance than any of the factors comprising it. Notice that the way we combine the factors is non-linear in the sense that the output of current weak classifier is based on the previous factor performance on the training data, since the weights of the training data would change each round after a new classifier is constructed.

The strong classifier we build has good predicting power as long as the factor performance of the current month is similar to the factor performance in our training data. Although the factor performance differs from time to time, the advantage of our model is that even if some factors fail to work, as long as the majority of the factors work our model would still have a good predictive power.

In addition, this method is easily scalable with more factors, because the algorithm will automatically pick the best factors and is not easily overfit. Ideally, the more factors we included in the training process the better performance will be. It is also adaptive, because we train new classifiers each month, which captures the seasonality and evolution in factor performance.

Figure 5 shows a detailed step by step flow chart of our machine learning model.

Figure 5: Flow chart of the machine learning model


Normalize all the factor value to a real value (0,1] according to the rank of that factor in each month
Classify outperformers and underperformers during the training period. Define outperformers to be those stocks that has a forward return in the top 30% of that month, and underperformers to be those stocks that has a forward return in the bottom 30% of that month.
Initially equally weight each stock of the outperformers and underperformers
Build weak classifiers for each factor from the factor pool. Select the best performing factor according to a discriminative objective function which shows the differentiation power of that weak classifier (factor).
Add the selected weak classifier to the strong classifier and change the weight of training samples: lower the weights of those correctly classified samples, and increase the weights of those incorrectly classified samples.
The final output of the strong classifier is the value of all the weak classifiers add up together.

Source: Deutsche Bank Quantitative Strategy

Model algorithm details

In this part, we will show the model process and detailed calculation steps.

Given a labeled set of stock $\boldsymbol{S}= \{(x_1, y_1), (x_2, y_2),\ldots(x_N, y_N)\}$, where N is the number of stocks in the training data, x_i is the factor score vector for stock $\boldsymbol{i}$, y_i is the corresponding labeling (or classification) for stock $\boldsymbol{i}$, i.e. $y_i=1$ if a stock has a top 30% forward return, $y_i=-1$ if a stock has a bottom 30% forward return.

In the AdaBoost framework, one weak classifier h is built for each factor k in the factor pool F. The weak classifier is a function from the real value factor space to a real valued classification confidence space. For a stock x_i the factor score is denoted as $f^k(x_i)$ for factor k, we assign a weight to each stock denoted as $w(x_i)$.

The weak classifier h is trained as a piecewise function:

If $f(x)$ is in the j^{th} quantile denoted as $f(x) \in \text{quantile}_j$, the value of the weak classifier is:

$$h(x) = \frac{1}{2}\ln\left(\frac{W_+^j + \varepsilon}{W_-^j + \varepsilon}\right)$$

where ε is a small value set as $1/N$ to make the function robust (so that the nominator and denominator won't be 0), and $j = 1,2,\ldots,Q$ is the number of quantiles (in our experiments we set $Q=5$), and $W_\pm^j$ is the sum of the weights in quantile j.

$$W_\pm^j = \sum_{y_i=\pm1,\, f(x_i)\in quantile_j} w(x_i)$$

Intuitively, the larger the total weight of outperformers in a quantile, the more likely the stock is going to outperform if the future factor score falls into that quantile. Therefore, the more positive $h(x)$ will be. Similarly if the weights of outperformers are smaller than the weights of underperformer, $h(x)$ will be negative.

We define a discriminative objective function, to show how good the weak classifier is:

$$Z = \sum_{j=1}^{n} \sqrt{W_+^j W_-^j}$$

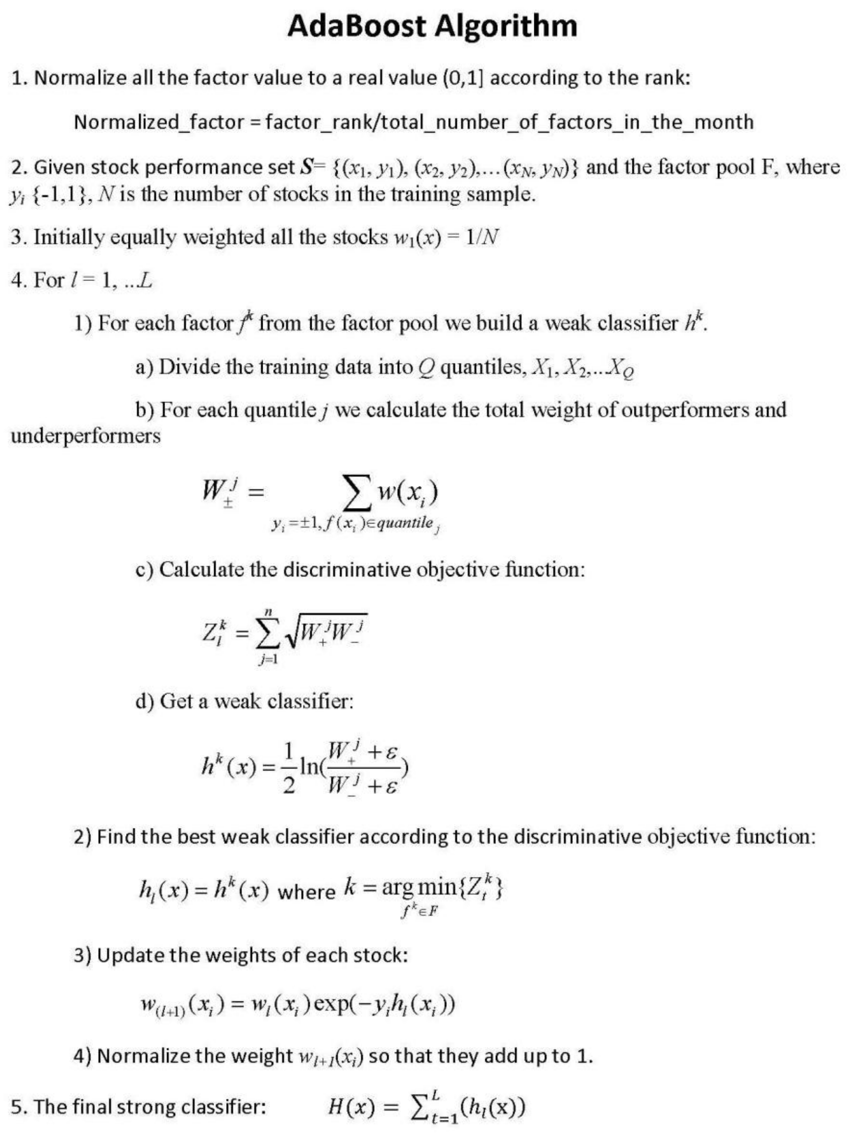
The intuition of this objective function is that a good weak classifier should have strong differentiation power. Because all the weights add up to 1, in each quantile if the difference between W_+^j and W_-^j is large (meaning the weak classifier has strong differentiation power) then the objective function will be small.

We update the weights each round of weak classifier $w_{(l+1)}(x_i) = w_l(x_i)\exp(-y_i\, h_l(x_i))$, where l is the l^{th} layer of weak classifier. If currently the weak classifier correctly classified the stock we decrease the weight $w_l(x_i)$ otherwise we increase the weight. The higher the absolute value of $h_l(x_i)$, the smaller the weight of correctly classified stocks and conversely the larger the weight on the incorrectly classified stocks. In that way, for the next weak classifier we will focus more on the previously misclassified stocks.

We use the AdaBoost algorithm to choose weak classifiers to build one strong classifier that returns the confidence value. Assume $h_l(x)$ is the l-th weak classifier, and $\boldsymbol{H(x)}$ is the strong classifier built by weak classifier: $\boldsymbol{H(x)} = \Sigma(h_l(x))$.

Figure 6 gives the full algorithm of our AdaBoost stock selection model.

Figure 6: Algorithm of AdaBoost Stock Selection Model



AdaBoost Algorithm

1. Normalize all the factor value to a real value (0,1] according to the rank:

   Normalized_factor = factor_rank/total_number_of_factors_in_the_month

2. Given stock performance set $S = \{(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N)\}$ and the factor pool F, where y_i {-1,1}, N is the number of stocks in the training sample.

3. Initially equally weighted all the stocks $w_1(x) = 1/N$

4. For $l = 1, \ldots L$

   1) For each factor f^k from the factor pool we build a weak classifier h^k.

      a) Divide the training data into Q quantiles, $X_1, X_2, \ldots X_Q$

      b) For each quantile j we calculate the total weight of outperformers and underperformers

$$W_{\pm}^{j} = \sum_{y_i = \pm 1, f(x_i) \in quantile_j} w(x_i)$$

      c) Calculate the discriminative objective function:

$$Z_l^k = \sum_{j=1}^{n} \sqrt{W_+^j W_-^j}$$

      d) Get a weak classifier:

$$h^k(x) = \frac{1}{2} \ln\left(\frac{W_+^j + \varepsilon}{W_-^j + \varepsilon}\right)$$

   2) Find the best weak classifier according to the discriminative objective function:

$$h_l(x) = h^k(x) \text{ where } k = \arg\min_{f^k \in F} \{Z_l^k\}$$

   3) Update the weights of each stock:

$$w_{(l+1)}(x_i) = w_l(x_i) \exp(-y_i h_l(x_i))$$

   4) Normalize the weight $w_{l+1}(x_i)$ so that they add up to 1.

5. The final strong classifier: $H(x) = \sum_{t=1}^{L} (h_l(\mathrm{x}))$

Source: Deutsche Bank Quantitative Strategy

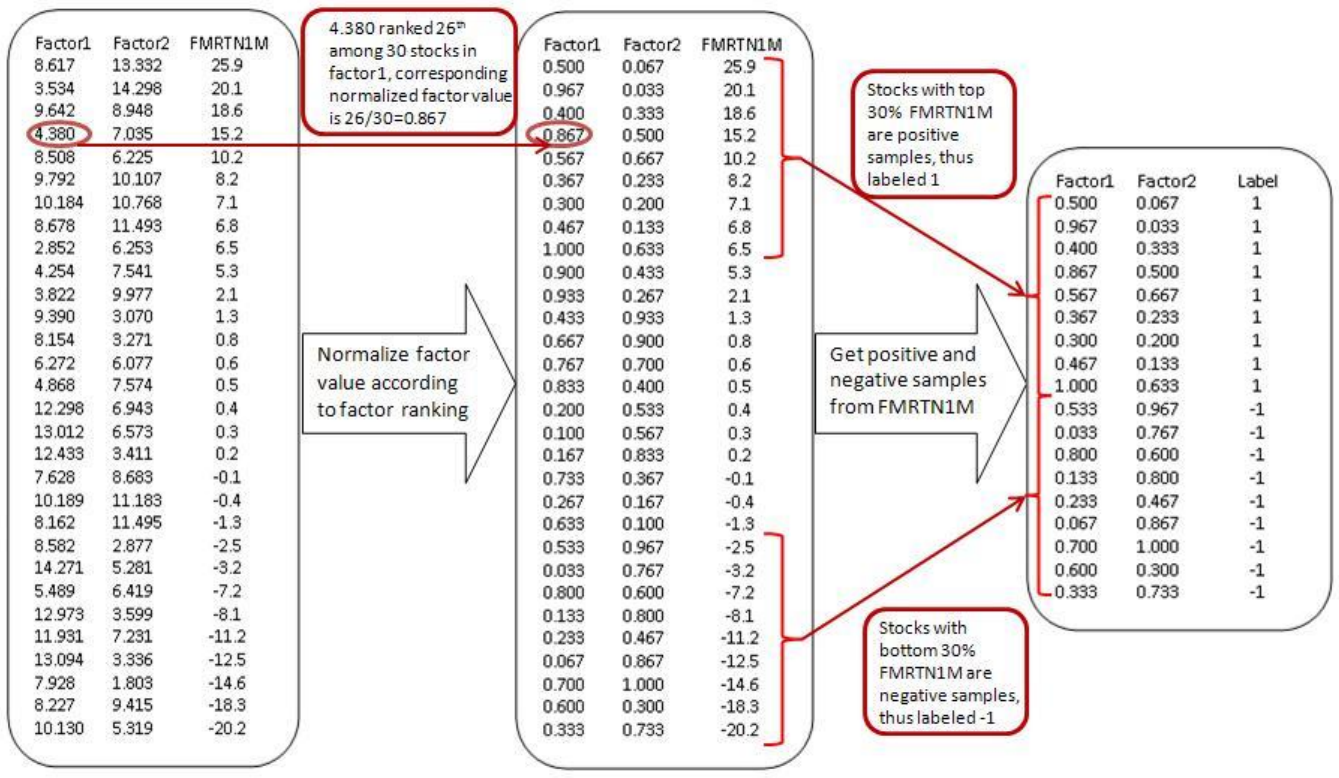
A simple example of machine learning

The best way to understand how our AdaBoost algorithm works is to work through an example. For this simple example we use artificially generated factor scores. Suppose we want to predict month-ahead returns using two factors in the factor pool.

Data preparation

The first step is to construct training set. As discussed earlier, we need to normalize each factor score according to factor ranking and then get the top 30% as outperformers and the bottom 30% as underperformers. Figure 7 shows detailed example of how we prepare the data for training.

Figure 7: Data Preparation for Training



Factor1	Factor2	FMRTN1M
8.617	13.332	25.9
3.534	14.298	20.1
9.642	8.948	18.6
4.380	7.035	15.2
8.508	6.225	10.2
9.792	10.107	8.2
10.184	10.768	7.1
8.678	11.493	6.8
2.852	6.253	6.5
4.254	7.541	5.3
3.822	9.977	2.1
9.390	3.070	1.3
8.154	3.271	0.8
6.272	6.077	0.6
4.868	7.574	0.5
12.298	6.943	0.4
13.012	6.573	0.3
12.433	3.411	0.2
7.628	8.683	-0.1
10.189	11.183	-0.4
8.162	11.495	-1.3
8.582	2.877	-2.5
14.271	5.281	-3.2
5.489	6.419	-7.2
12.973	3.599	-8.1
11.931	7.231	-11.2
13.094	3.336	-12.5
7.928	1.803	-14.6
8.227	9.415	-18.3
10.130	5.319	-20.2

4.380 ranked 26th among 30 stocks in factor1, corresponding normalized factor value is 26/30=0.867

Normalize factor value according to factor ranking

Factor1	Factor2	FMRTN1M
0.500	0.067	25.9
0.967	0.033	20.1
0.400	0.333	18.6
0.867	0.500	15.2
0.567	0.667	10.2
0.367	0.233	8.2
0.300	0.200	7.1
0.467	0.133	6.8
1.000	0.633	6.5
0.900	0.433	5.3
0.933	0.267	2.1
0.433	0.933	1.3
0.667	0.900	0.8
0.767	0.700	0.6
0.833	0.400	0.5
0.200	0.533	0.4
0.100	0.567	0.3
0.167	0.833	0.2
0.733	0.367	-0.1
0.267	0.167	-0.4
0.633	0.100	-1.3
0.533	0.967	-2.5
0.033	0.767	-3.2
0.800	0.600	-7.2
0.133	0.800	-8.1
0.233	0.467	-11.2
0.067	0.867	-12.5
0.700	1.000	-14.6
0.600	0.300	-18.3
0.333	0.733	-20.2

Stocks with top 30% FMRTN1M are positive samples, thus labeled 1

Get positive and negative samples from FMRTN1M

Stocks with bottom 30% FMRTN1M are negative samples, thus labeled -1

Factor1	Factor2	Label
0.500	0.067	1
0.967	0.033	1
0.400	0.333	1
0.867	0.500	1
0.567	0.667	1
0.367	0.233	1
0.300	0.200	1
0.467	0.133	1
1.000	0.633	1
0.533	0.967	-1
0.033	0.767	-1
0.800	0.600	-1
0.133	0.800	-1
0.233	0.467	-1
0.067	0.867	-1
0.700	1.000	-1
0.600	0.300	-1
0.333	0.733	-1

Source: Deutsche Bank Quantitative Strategy

Building a weak classifier

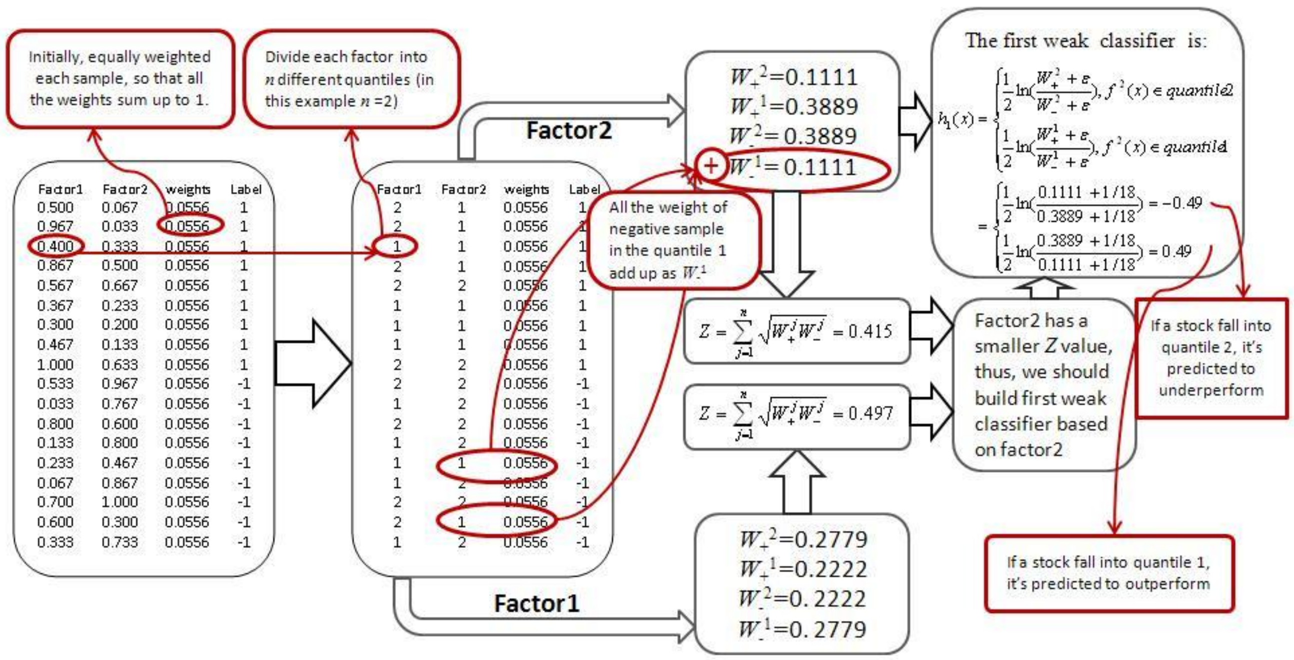
Once we have the training data ready, we equally weight each stock, so that all the weights sum up to 1. In this example, we divide each factor into 2 quantiles. For each factor we calculate the sum of all the outperformers and underperformers' weights in quantile 1 and quantile 2, which are denoted as $W_{\pm}^{j}$, where j refers to quantiles and ± refers to outperforming and underperforming. We calculate the discriminative objective function $Z=\mathrm{sqrt}(W_{+}^{j}W_{-}^{j})$ for each factor. It turns out factor 2 has a smaller Z value, which means factor 2 is currently the best performing factor and can better differentiate outperformers from underperformers. So we construct the weak classifier using factor 2.

The output of the weak classifier is denoted as a piecewise function. If a stock has a factor 2 value in quantile 1 the output of the weak classifier is 0.49, meaning it is more likely to be an outperformer. This is because in the training set there are more outperformers than underperformers in quantile 2. This is naturally defined by the function determine the value of weak classifier:

$$h(x)=\frac{1}{2}\ln\left(\frac{W_{+}^{j}+\varepsilon}{W_{-}^{j}+\varepsilon}\right)$$

We can see that if the weight of underperformers is larger than the weight of outperformers, the output of the weak classifier would have a negative value, indicating the stock is more likely to be underperformers if its factor score falls into this quantile. This is the case when the factor 2 value falls into quantile 2 in this example, the output of the weak classifier would be negative. Figure 8 illustrate how we choose the best performing factor and build the first weak classifier.

Figure 8: Build Weak Classifier


Initially, equally weighted each sample, so that all the weights sum up to 1.
Divide each factor into n different quantiles (in this example n =2)
Factor2
Factor1
W_+^2=0.1111
W_+^1=0.3889
W_-^2=0.3889
W_-^1=0.1111
All the weight of negative sample in the quantile 1 add up as W_-^1
W_+^2=0.2779
W_+^1=0.2222
W_-^2=0.2222
W_-^1=0.2779
$Z=\sum_{j=1}^{n}\sqrt{W_+^jW_-^j}=0.415$
$Z=\sum_{j=1}^{n}\sqrt{W_+^jW_-^j}=0.497$
The first weak classifier is:
$h_1(x)=\begin{cases}\frac{1}{2}\ln(\frac{W_+^2+\varepsilon}{W_-^2+\varepsilon}), f^2(x)\in quantile2\\ \frac{1}{2}\ln(\frac{W_+^1+\varepsilon}{W_-^1+\varepsilon}), f^2(x)\in quantile1\end{cases}=\begin{cases}\frac{1}{2}\ln(\frac{0.1111+1/18}{0.3889+1/18})=-0.49\\ \frac{1}{2}\ln(\frac{0.3889+1/18}{0.1111+1/18})=0.49\end{cases}$
Factor2 has a smaller Z value, thus, we should build first weak classifier based on factor2
If a stock fall into quantile 2, it's predicted to underperform
If a stock fall into quantile 1, it's predicted to outperform

Factor1	Factor2	weights	Label
0.500	0.067	0.0556	1
0.967	0.033	0.0556	1
0.400	0.333	0.0556	1
0.867	0.500	0.0556	1
0.567	0.667	0.0556	1
0.367	0.233	0.0556	1
0.300	0.200	0.0556	1
0.467	0.133	0.0556	1
1.000	0.633	0.0556	1
0.533	0.967	0.0556	-1
0.033	0.767	0.0556	-1
0.800	0.600	0.0556	-1
0.133	0.800	0.0556	-1
0.233	0.467	0.0556	-1
0.067	0.867	0.0556	-1
0.700	1.000	0.0556	-1
0.600	0.300	0.0556	-1
0.333	0.733	0.0556	-1

Factor1	Factor2	weights	Label
2	1	0.0556	1
2	1	0.0556	1
1	1	0.0556	1
2	1	0.0556	1
2	2	0.0556	1
1	1	0.0556	1
1	1	0.0556	1
1	1	0.0556	1
2	2	0.0556	1
2	2	0.0556	-1
1	2	0.0556	-1
2	2	0.0556	-1
1	2	0.0556	-1
1	1	0.0556	-1
1	2	0.0556	-1
2	2	0.0556	-1
2	1	0.0556	-1
1	2	0.0556	-1

Source: Deutsche Bank Quantitative Strategy

Building a strong classifier

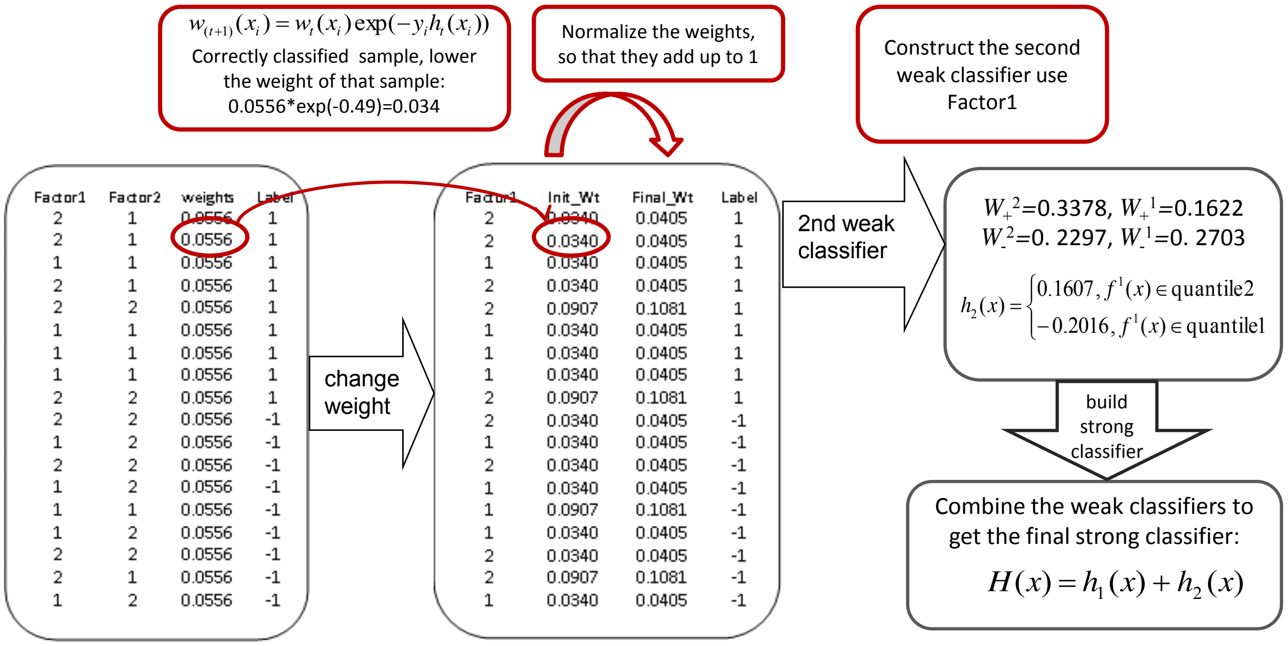
After a weak classifier is built, we change the weights of all the stocks by a multiple of exp(-$y_ih(x_i)$). This means if the stock label y_i and the output of the latest classifier $h(x_i)$ have the same sign (meaning the last weak classifier correctly classified this stock) we would lower the weights of those stocks. Similar, this multiple can ensure we increase the weights assigned to those misclassified stocks. Therefore, in the next round, the weak classifier would focus on those misclassified stocks. In addition, the larger the absolute value of the weak classifier for those correctly classified stocks, the lower the weights of those stocks will be; and the larger the absolute value of the weak classifier for those misclassified stocks, the higher the weights of those stocks will be. This also makes sense; if the previous weak classifier is confident about the label of a stock but in fact it got misclassified, then this stock should get special attention.

Again we normalized all the weights so that they add up to 1, and find the next weak classifier the same way: choose the factor with the smallest discriminative objective function Z to build the current weak classifier. Notice that, we don't need to exclude previous selected factors, because after the change of weights according to the weak classifier built by factor 2, now factor 1 would have a smaller discriminative objective function value than factor 2. So normally the same factor would not be selected again right away, although it could be selected again after many rounds. But by that time the weights of the training stocks would have changed, so the weak classifier constructed with the same factor would be totally different.

The determination of how many layers of weak classifiers can be set as a fixed number. Normally, the more layers of weak classifiers, the better the performance for the new data will be. However, the incremental benefit from each new classifier will diminish because after certain number of factors have been chosen the rest are correlated with some of the factors already selected. For example, there is no need to set the number of weak classifiers greater than the number of factors. For a small list of uncorrelated factors, we could set the number of layers equal to the factor number, but with a larger set of factors, it is almost certain that many factors would be highly correlated. However, the good thing about the AdaBoost algorithm is that it is less subject to the problem of overfitting, or multi-colinearity. Therefore, the performance of the strong classifier won't decrease significantly when trained with too many layers (although of course training with too many layers is computationally inefficient and unnecessary). Only with larger factor pools and less correlated factors could we train more layers of weak classifiers and still improve performance.

Finally the combination of all the weak classifier is easy; the output of the strong classifier is simply the sum of all the weak classifiers. Figure 9 shows the example of how to build strong classifier after a weak classifier is built in Figure 8.

Figure 9: Build Strong Classifier


$w_{(t+1)}(x_i) = w_t(x_i)\exp(-y_i h_t(x_i))$
Correctly classified sample, lower the weight of that sample: 0.0556*exp(-0.49)=0.034
Normalize the weights, so that they add up to 1
Construct the second weak classifier use Factor1
change weight
2nd weak classifier
W_+^2=0.3378, W_+^1=0.1622
W_-^2=0. 2297, W_-^1=0. 2703
$h_2(x) = \begin{cases} 0.1607, f^1(x) \in \text{quantile2} \\ -0.2016, f^1(x) \in \text{quantile1} \end{cases}$
build strong classifier
Combine the weak classifiers to get the final strong classifier:
$H(x) = h_1(x) + h_2(x)$

Factor1	Factor2	weights	Label
2	1	0.0556	1
2	1	0.0556	1
1	1	0.0556	1
2	1	0.0556	1
2	2	0.0556	1
1	1	0.0556	1
1	1	0.0556	1
1	1	0.0556	1
2	2	0.0556	1
2	2	0.0556	-1
1	2	0.0556	-1
2	2	0.0556	-1
1	2	0.0556	-1
1	1	0.0556	-1
1	2	0.0556	-1
2	2	0.0556	-1
2	1	0.0556	-1
1	2	0.0556	-1

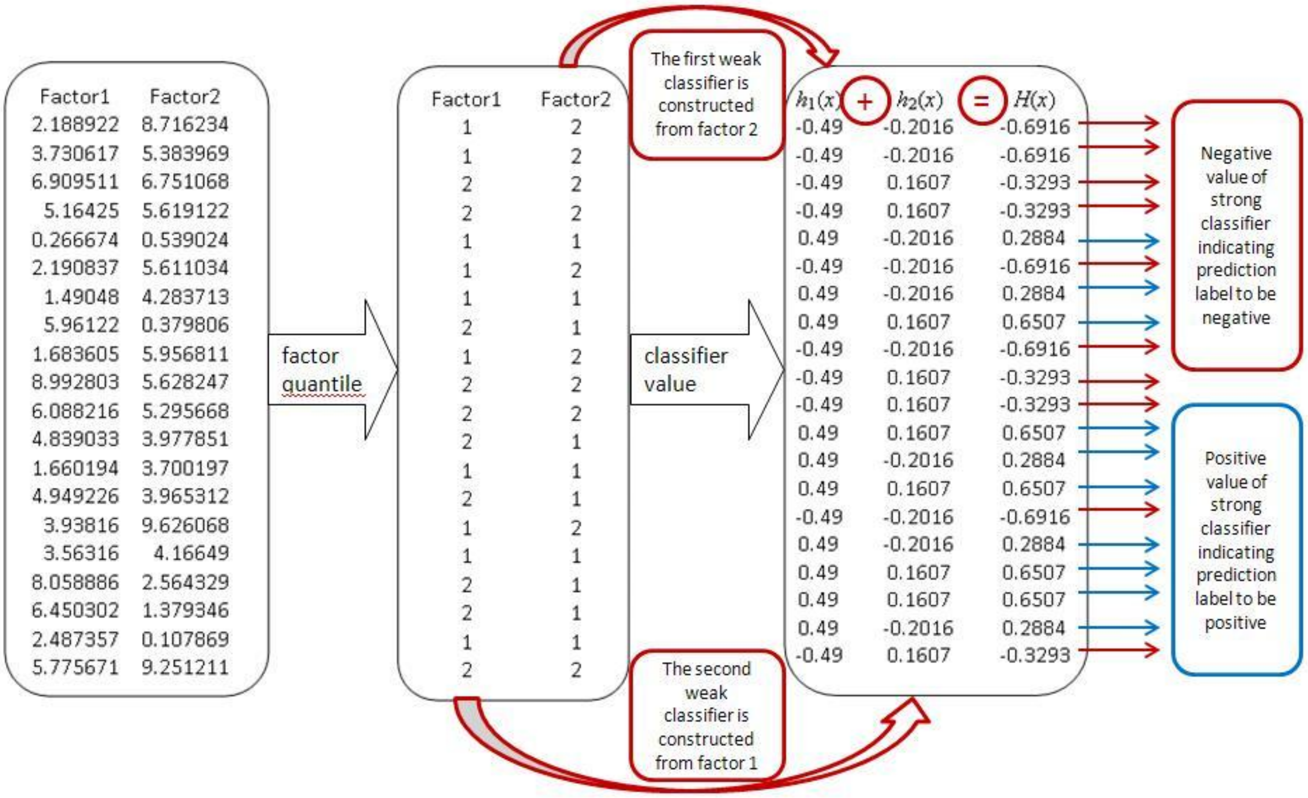
Factor1	Init_Wt	Final_Wt	Label
2	0.0340	0.0405	1
2	0.0340	0.0405	1
1	0.0340	0.0405	1
2	0.0340	0.0405	1
2	0.0907	0.1081	1
1	0.0340	0.0405	1
1	0.0340	0.0405	1
1	0.0340	0.0405	1
2	0.0907	0.1081	1
2	0.0340	0.0405	-1
1	0.0340	0.0405	-1
2	0.0340	0.0405	-1
1	0.0340	0.0405	-1
1	0.0907	0.1081	-1
1	0.0340	0.0405	-1
2	0.0340	0.0405	-1
2	0.0907	0.1081	-1
1	0.0340	0.0405	-1

Source: Deutsche Bank Quantitative Strategy

Predicting with new data

After we train our machine learning model, we can predict month-ahead returns using the constructed classifiers with new data. First we transform factor scores into quantiles, and get the corresponding value of the weak classifier, adding up all the weak classifiers, we get the final value of the strong classifier. A positive value of the strong classifier indicates a stock is likely to be an outperformer, and negative value of the strong classifier indicates the opposite. The illustration is shown in Figure 10.

Figure 10: Predicting new data


factor quantile
classifier value
The first weak classifier is constructed from factor 2
The second weak classifier is constructed from factor 1
Negative value of strong classifier indicating prediction label to be negative
Positive value of strong classifier indicating prediction label to be positive

Factor1	Factor2
2.188922	8.716234
3.730617	5.383969
6.909511	6.751068
5.16425	5.619122
0.266674	0.539024
2.190837	5.611034
1.49048	4.283713
5.96122	0.379806
1.683605	5.956811
8.992803	5.628247
6.088216	5.295668
4.839033	3.977851
1.660194	3.700197
4.949226	3.965312
3.93816	9.626068
3.56316	4.16649
8.058886	2.564329
6.450302	1.379346
2.487357	0.107869
5.775671	9.251211

Factor1	Factor2
1	2
1	2
2	2
2	2
1	1
1	2
1	1
2	1
1	2
2	2
2	2
2	1
1	1
2	1
1	2
1	1
2	1
2	1
1	1
2	2

$h_1(x)$	+	$h_2(x)$	=	$H(x)$
-0.49		-0.2016		-0.6916
-0.49		-0.2016		-0.6916
-0.49		0.1607		-0.3293
-0.49		0.1607		-0.3293
0.49		-0.2016		0.2884
-0.49		-0.2016		-0.6916
0.49		-0.2016		0.2884
0.49		0.1607		0.6507
-0.49		-0.2016		-0.6916
-0.49		0.1607		-0.3293
-0.49		0.1607		-0.3293
0.49		0.1607		0.6507
0.49		-0.2016		0.2884
0.49		0.1607		0.6507
-0.49		-0.2016		-0.6916
0.49		-0.2016		0.2884
0.49		0.1607		0.6507
0.49		0.1607		0.6507
0.49		-0.2016		0.2884
-0.49		0.1607		-0.3293

Source: Deutsche Bank Quantitative Strategy