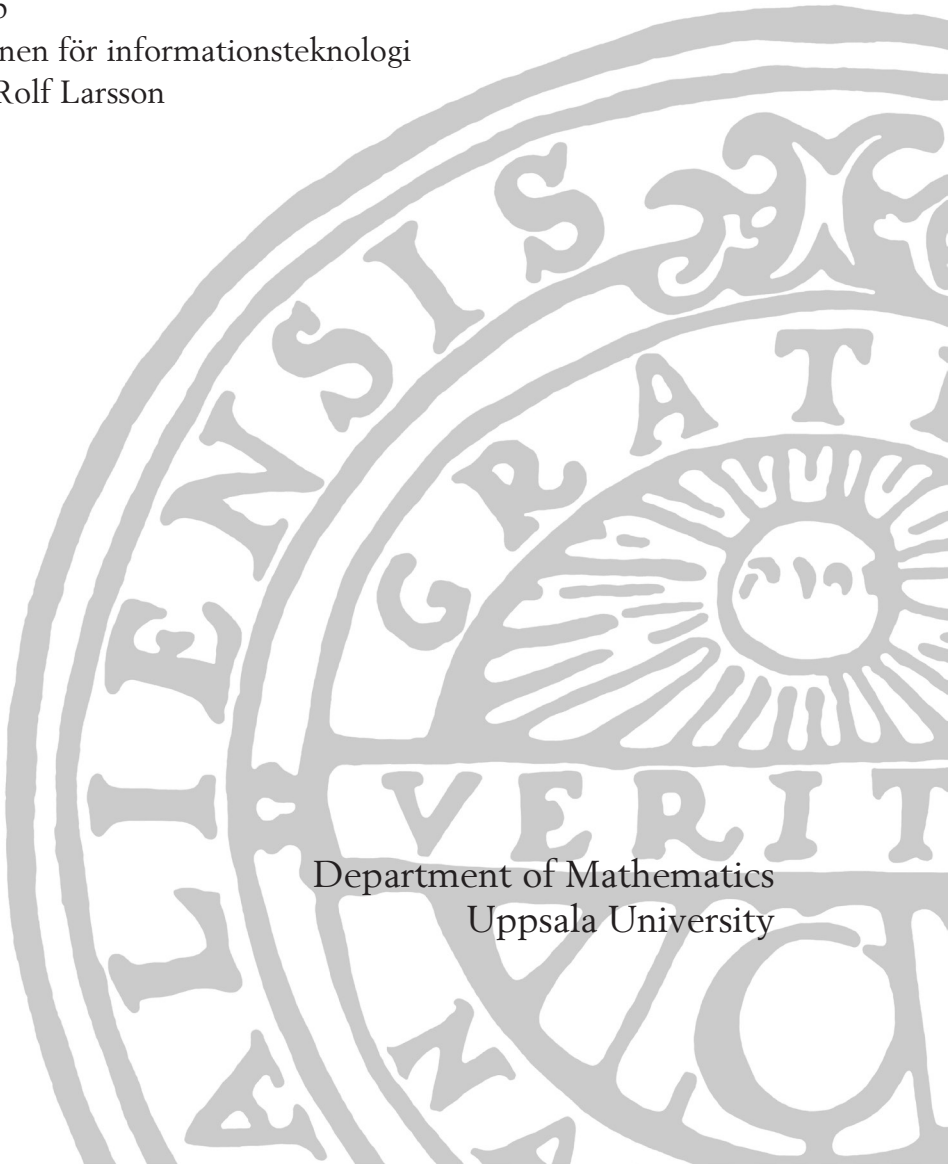
UPPSALA
UNIVERSITET

U.U.D.M. Project Report 2015:17

Dimension Reduction Methods for Predicting Financial Data

Denniz Falk Soylu

Examensarbete i matematik, 15 hp
Handledare: Josef Höök, Institutionen för informationsteknologi
Ämnesgranskare och examinator: Rolf Larsson
Juni 2015



Department of Mathematics
Uppsala University

Contents

Part I

Abstract

A common approach in econometrics is to study financial time series, where you attempt to explain the variation in the time series with a smaller number of *factors*.

Our aim with this thesis is to attemp to predict a number of stocks, on the American stock market SP500, monthly log-returns under the assumption that the dimension of the market data is "too large" in some sense. We will use different dimension reduction methods, such as *Principal component regression, Partial least squares* and *Reduced rank regression,* to reduce the dimensionality and use cross-validation to choose which model we will use in an "explanatory model" to construct these factors.

Part II

Introduction

A major part in econometrics is about making models to explain the economy including the ability to explain the evolution of financial derivatives. A common approach is to study financial time series, where you attempt to explain the variation in the time series with a smaller number of *factors,* for example factors taken from an external time series such as inflation data, business cycle data, etc.

In this paper we will predict a number of stocks, on the American stock market SP500, monthly log-returns under the assumption that the dimension of the market data is "too large" in some sense. We will therefore reduce the dimensionality with different methods and use cross-validation to determine how accurately they predict the data in an "explanatory model", i.e. estimate the prediction error.

Instead of taking factors from an external time series we will focus on constructing the factors from the time series itself.

1 Main Framework

Let $y_1, y_2, ..., y_n$, with $y_k = (y_{ki})_{i=1}^m$ be n vectors in $\mathbb{R}^m$. These are the dependent variables. Let also $x_1, x_2, ..., x_n$, be n vectors in $\mathbb{R}^p$. These are the independent variables.

Our basic task is to predict y using x under the assumption that the dimension of the data is "too large", i.e. p is too large. In many cases when the number of predictors is large compared to the number of observations, the data matrix is likely to be ill-conditional or even singular and the regression approach is no longer possible because of multicollinearity. Thus, we want to find the "best", in some sense, d-dimensional subspace ($d < p$) of x for predicting y linearly.

By a d-dimensional subspace we mean a linear transformation to predictors $z_k = G^T x_k$ with $G \in \mathbb{R}^{p \times d}$. G will be a projection $\mathbb{R}^p \to \mathbb{R}^d$ and is the actual dimension reduction. We will refer to the columns of G as *feature directions* since these vectors map the original data x into *features* z_k.

The linear predictor of y will then be

$$\hat{y} = B^T z = B^T G^T x$$

where $B \in \mathbb{R}^{d \times m}$ is the regression matrix.

Our approach in this thesis will be to

- Use different methods to:

  - Apply cross-validation for determine reduced dimensionality d
  - Find projection matrix G
  - Find regression matrix B

- Check model assumptions, i.e. investigate residuals $Y - XGB$ and goodness of fit for the different methods.

Which method is to be preferred in the sense that they admit analytic or computational solutions?

Part III

Data

The data is taken from the American stock market index *Standard & Poor's 500* (S&P 500) which is one of the most commonly used benchmarks for the overall United States. It includes the 500 leading companies.

There are close prices available from 1980-01-02. We have to make a tradeoff since we want to include as many stocks as possible in our market data but not all companies did exist at that time. So, we sacrifice some companies to get equal date range on all stocks. Those companies that have prices available from the first trade day 1998, i.e. from 1998-01-02 are the ones that are chosen. The time period will be monthly and only the first trade day in each month is selected.

Here we will handle returns instead of price indices for two main reasons. A return series have more useful statistical properties and is a scale-free summary of the investment opportunity. In fact, we will use log-returns since it has some advantages over returns, which we will discuss later.

A standard assumption is that the returns are independent of each other but then the prediction will be useless. Thus we will assume that there exists a little correlation between some stocks, for example it could be companies in the same field, such as the oil business.

Before we will give some definitions we will summarize our market data. We have chosen 355 stocks with monthly log-returns from 1998-01-02, which gives us 206 months.

2 Definitions

Let P_t be the price of an asset at time index t.

2.1 Simple Return

Holding an asset for one-period from a date $t-1$ to a date t would result in a so called *simple gross return*

$$1 + R_t = \frac{P_t}{P_{t-1}} \quad \text{or} \quad P_t = P_{t-1}(1 + R_t).$$

The corresponding simple returns for an asset held for k periods, between dates $t-k$ and t are called *k-period simple gross returns*

$$
\begin{aligned}
1 + R_t[k] = \frac{P_t}{P_{t-k}} &= \frac{P_t}{P_{t-1}} \cdot \frac{P_{t-1}}{P_{t-2}} \cdot \ldots \cdot \frac{P_{t-k+1}}{P_{t-k}} \\
&= (1 + R_t)(1 + R_{t-1}) \ldots (1 + R_{t-k+1}) \\
&= \prod_{i=0}^{k-1} (1 + R_{t-i}).
\end{aligned}
$$

The k-period simple gross returns are just the product of the k one-period simple gross returns and is called a compound return.

2.2 Log Return

Log return or *continuously compound return* is the natural logarithm of the simple gross return of an asset

$$
r_t = ln(1 + R_t) = ln \frac{P_t}{P_{t-1}} = p_t - p_{t-1},
$$

where $p_t = ln(P_t)$.

Let

$$
\mathbf{r_t} = (r_{1t}, r_{2t}, \ldots, r_{kt})^T
$$

be the log returns of k assets at time t.

One of the advantages that log returns have over simple returns is that the multiperiod return becomes the sum of continuously compounded one-period returns, instead of a product as in the former case.

$$
\begin{aligned}
r_t[k] = ln(1 + R_t[k]) &= ln((1 + R_t)(1 + R_{t-1}) \ldots (1 + R_{t-k+1})) \\
&= ln(1 + R_t) + ln(1 + R_{t-1}) + \ldots + ln(1 + R_{t-k+1}) \\
&= r_t + r_{t-1} + \ldots + r_{t-k+1}
\end{aligned}
$$

3 Distribution

In financial studies a common assumption is that the simple returns are independently and identically distributed as normal with fixed mean and variance. One of the problems with this assumption is that the multiperiod simple return, $R_t[k]$, will not be normally distributed since in general a product of normally distributed variables is not normally distributed.

Thus, we will use another common assumption which assumes that the log returns of an asset is independent and identically distributed as normal with mean

μ and variance σ^2. The continuously compounded multiperiod return, $r_t[k]$, is normally distributed (under the normal assumption for $\{r_t\}$) since the sum of normally distributed variables is normally distributed (under joint normality).

A positive excess kurtosis can occur when using many stock returns but that is something we have to accept.

Part IV

Methods

From here and on the data we are working with (if nothing else is stated) are standardized. Otherwise, if one or some of the coordinates of y have a significant larger variance than others, it will tend to dominate the choice of the d-dimensional prediction subspace.

For the dimension reduction we have used three main methods. First we will give some background about SVD and cross-validation after that we will present the different methods.

4 Singular value decomposition

Singular value decomposition (SVD) is a technique that allows an exact representation of any matrix. SVD can represent a high-dimensional matrix into a low-dimensional by eliminating the less important parts and produce an approximative representation with any desired number of dimensions (rank). The fewer dimensions we choose, the less accurate will the approximation be.

Let A be a $m \times n$ matrix, then SVD can express the matrix as

$$A = USV^T$$

where

S	is a $r \times r$ diagonal matrix. The diagonal values are called *singular values* of A and are in a decreasing order with the biggest on the top.
U	is a $m \times r$ column-orthogonal matrix. The columns of U are called the *left singular vectors* for corresponding singular values.
V	is a $n \times r$ column-orthogonal matrix. The columns of V are called the *right singular vectors* for corresponding singular values.

The best way to reduce the dimensionality of the three matrices is to set the smallest singular values to zero. We will not go into detail why this works, for more information, see [2].

How many singular values you should retain is something you choose, but as said previously the fewer dimensions the less accurate the approximation will be. [2] has a good explanation how to do and we quote:

"A useful rule of thumb is to retain enough singular values to make up 90% of the energy in Σ. That is, the sum of the squares of the retained singular values should be at least 90% of the sum of the squares of all the singular values."

The sum of squares of the retained singular values is, in some literature, called the explained variance.

5 Cross-validation

Cross-validation is a statistical method of evaluating and comparing learning algorithms by dividing data into two parts, one group which is used to train the model and another group to test the model. The parts are usually called training and validation set.

The problem that can occur in a regression model is that the model may show an adequate prediction capability on the training data but then fail to predict the future unseen data. In reality, the data may often have a random noise and thus trying to explain the model can give a negative effect on the predictive power. This problem is known as *overfitting*.

One of the main goals in cross-validation is to avoid overfitting by testing different numbers of components to be the reduced dimensionalities (ranks) and then select the model that predicts most accurately, this is called *model selection* in some areas. Another goal is to compare the performance of two or more different methods and find the best of them for the available data.

There are several variants of cross-validation but the most basic one, *k-fold* cross-validation is the one we will use. In k-fold cross-validation the data is first divided into K equally (or nearly equally) sized parts randomly. Then it uses $K-1$ parts (training set) to fit the model and then calculates the prediction error when predicting the kth part. We do this for $k = 1, \ldots, K$ and combine the K estimates of the prediction error. More formally, let

$$\kappa : \{1, \ldots, N\} \rightarrowtail \{1, \ldots, K\} \tag{1}$$

be an indexing function which divides each observation i in the data into one of the K parts randomly. Let also $\hat{f}^{-k}(x)$ denote the fitted function with the kth part removed. Then the cross-validation estimate of prediction error is

$$CV(\hat{f}) \quad = \quad \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{f}^{-\kappa(i)}(x_i)\right)^2 \tag{2}$$

The fitted model in our case will be that we use GB (the projection matrix times the regression matrix), calculated without the kth part.

Let $f(x, \alpha)$ be a given set of models indexed by a tuning parameter α and denote the αth model fit with the kth part of the data removed by $\hat{f}^{-k}(x, \alpha)$. For this

set of models we define

$$CV(\hat{f}, \alpha) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{f}^{-\kappa(i)}(x_i, \alpha)\right)^2 \tag{3}$$

Function (3) provides an estimate of the test error curve and we find the tuning parameter $\hat{\alpha}$ that minimizes it. Our final model is $f(x, \hat{\alpha})$, which we then fit to all the data, i.e. this $\hat{\alpha}$ will be our reduced dimensionality d, i.e. the rank of GB. Another alternative for choosing the value of the tuning parameter for the model selection is the *one-standard error* rule, which chooses the most parsimonious model whose error is no more than one standard error above the error of the best model.

The most common choices for k are 5 or 10. We will use $k = 10$ since the cross-validation will have lower variance than if we would have for example $k = N$ and it is better than $k = 5$ to estimate the standard error. The problem which can occur when using tenfold cross-validation is that the bias can be big but we have approximately 200 observations and then tenfold cross-validation will not suffer so much from bias because the training set will have about 180 observations and behave much like the original data. For more information see [4].

6 Reduced rank regression

Let X be the $n \times p$ matrix of all independent data and Y the $n \times m$ matrix of all dependent data, where each row corresponds to one observation vector. $X_{\cdot j}$ and $Y_{\cdot j}$ are the j-th column of respective matrix.

As stated before we have the predictor

$$\hat{y} = B^T G^T x = B^T z \tag{4}$$

Collecting data points into matrices X and Y for independent and dependent data respectively we will get the predictor

$$\hat{Y} = XGB = ZB \tag{5}$$

Use the least squares estimator for B, so we have in our case

$$B = (Z^T Z)^{-1} Z^T Y$$

and the predictor becomes

$$\hat{Y} = Z(Z^T Z)^{-1} Z^T Y = XG(G^T X^T XG)^{-1} G^T X^T Y \tag{6}$$

To find G we will minimize the sum of squared errors when regressing each $Y_{\cdot j}$

onto $Z_{.j}$. The objective function to be minimized is

$$\begin{aligned}
L(G) &= \sum_{j=1}^{m} \inf_{b_j \in \mathbb{R}^d} ||Y_{.j} - Zb_j||^2 = \sum_{j=1}^{m} \inf_{b_j \in \mathbb{R}^d} ||Y_{.j} - XGb_j||^2 \\
&= \inf_{b_j \in \mathbb{R}^{d \times m}} trace[(Y - XGB)^T (Y - XGB)] \\
&= trace[(Y - XG(G^T X^T XG)^{-1} G^T X^T Y)^T \cdot \\
&\cdot \quad (Y - XG(G^T X^T XG)^{-1} G^T X^T Y)] \\
&= trace[(Y^T - Y^T XG(G^T X^T XG)^{-1} G^T X^T) \cdot \\
&\cdot \quad (Y - XG(G^T X^T XG)^{-1} G^T X^T Y)] \\
&= trace[Y^T Y - Y^T XG(G^T X^T XG)^{-1} G^T X^T Y - \\
&- \quad Y^T XG(G^T X^T XG)^{-1} G^T X^T Y + \\
&+ \quad Y^T XG(G^T X^T XG)^{-1} G^T X^T XG(G^T X^T XG)^{-1} G^T X^T Y)] \\
&= trace[Y^T Y - Y^T XG(G^T X^T XG)^{-1} G^T X^T Y - \\
&- \quad Y^T XG(G^T X^T XG)^{-1} G^T X^T Y + \\
&+ \quad Y^T XG(G^T X^T XG)^{-1} G^T X^T Y)] \\
&= trace[Y^T Y - Y^T XG(G^T X^T XG)^{-1} G^T X^T Y] \qquad (7)
\end{aligned}$$

The problem of minimizing Equation (4) is known as *reduced rank regression* (RRR) in statistics and the signal processing literature.

In some literature $GB = T$ is used, where T has the reduced rank d, so for simplicity we will use the same notation for a while. To find the reduced rank d of T we will use cross-validation.

The objective function will then be

$$L(G) = trace[Y^T Y - Y^T XT - T^T X^T Y + T^T X^T XT]. \qquad (8)$$

In [5] they estimate such that $T \in \mathbb{R}^{p \times m}$ for the objective function

$$J_{tr} = trace[C_{yy} - C_{yx}T - T^T C_{yx}^T + T^T C_{xx} T] \qquad (9)$$

where

C_{yy}	auto-correlation matrix of y
C_{xx}	auto-correlation matrix of x
C_{yx}	cross-correlation matrix between y and x

as

$$\hat{T} = C_{xx}^{-\frac{T}{2}} V_1(R_{tr}) V_1(R_{tr})^T C_{xx}^{-\frac{1}{2}} C_{yx}^T \qquad (10)$$

where $V_1(R_{tr})$ is the first d right singular vectors of $R_{tr} = C_{yx} C_{xx}^{-\frac{T}{2}}$.

If we now go back to our objective function (7) we can write it like

$$\begin{aligned} L(G) &= trace[(n-1)C_{yy} - (n-1)C_{yx}T - (n-1)T^T C_{xy} + (n-1)T^T C_{xx}T] \\ &= trace[(n-1)(C_{yy} - C_{yx}T - T^T C_{yx}^T + T^T C_{xx}T)] \end{aligned} \tag{11}$$

since $C_{yy} = (n-1)^{-1}Y^T Y$, etc.

It doesn't matter if we want to minimize (9) or (11) because $(n-1)$ is a positive constant and will not contribute to the choice of T. Thus, our choice of T will be

$$\begin{aligned} \hat{T} &= C_{xx}^{-\frac{T}{2}} V_1(R_{tr}) V_1(R_{tr})^T C_{xx}^{-\frac{1}{2}} C_{yx}^T \\ &= ((n-1)^{-1}X^T X)^{-\frac{T}{2}} V_1(R_{tr}) \cdot \\ &\cdot \quad V_1(R_{tr})^T ((n-1)^{-1}X^T X)^{-\frac{1}{2}} ((n-1)^{-1}Y^T X)^T \end{aligned} \tag{12}$$

Since we compound T we can split it into the projection matrix G and the regression matrix B

$$G = ((n-1)^{-1}X^T X)^{-\frac{T}{2}} V_1(R_{tr}) \tag{13}$$

$$B = V_1(R_{tr})^T ((n-1)^{-1}X^T X)^{-\frac{1}{2}} ((n-1)^{-1}Y^T X)^T \tag{14}$$

7 Principal component regression

Principal components analysis (PCA) is a technique for projecting high-dimensional data, $X \in \mathbb{R}^{n \times p}$, onto the most important axes to construct a lower-dimensional data set.

The idea of PCA is to find linear combinations of $g_i \in \mathbb{R}^p$ such that $g_i^T X$ and $g_j^T X$ are uncorrelated for $i \neq j$ and the variances of $g_i^T X$ are as large as possible. So, the

1. *First principal component* of X is the linear combination $z_1 = g_1^T X$ that maximizes $Var(z_1)$ subject to the constraint $g_1^T g_1 = 1$.
2. *Second principal component* of X is the linear combination $z_2 = g_2^T X$ that maximizes $Var(z_2)$ subject to the constraints $g_2^T g_2 = 1$ and $Cov(z_i, z_1) = 0$.
3. *i-th principal components* of X is the linear combination $z_i = g_i^T X$ that maximizes $Var(z_i)$ subject to the constraints $g_i^T g_i = 1$ and $Cov(z_i, z_j) = 0$ for $j = 1, \ldots, i-1$.

The solution is that the i-th principal component score of X is

$$z_i = \mathbf{e}_i^T X \tag{15}$$

for $i = 1, \ldots, k$, where $\mathbf{e}_i = (e_{i1}, e_{i2}, \ldots, e_{ik})^T$ are the unit eigenvectors of the covariance matrix for X. Here, $\mathbf{e_1}$ corresponds to the largest eigenvalue of the covariance matrix for X, $\mathbf{e_2}$ to the second largest eigenvalue and so on. Now E, the matrix of eigenvectors $\mathbf{e_i}$, will be our projection matrix G where d is determined by using cross-validation.

The principal eigenvectors correspond to the projected axes where the variance of the data is maximized.

We have G and thus we know Z and we can define our regression matrix as

$$B = (Z^T Z)^{-1} Z^T Y$$

Note that principal component regression does not pay any attention to the correlation between x and y, it is only interested to find a smaller subspace with the largest possible variance for the variables, i.e. capture the most variable directions in the X space.

8 Partial least squares regression

Partial least squares (PLS) is a technique that also constructs a set of linear combinations of the input for regression, but it uses not only X for this task, it uses Y as well. The model is based on the principal components on both the independent data and the dependent data and the idea is to find the principal scores of $X \in \mathbb{R}^{n \times p}$ and $Y \in \mathbb{R}^{n \times m}$ and use them to build a regression model between the scores. We have

$$X = TP^T + E, \tag{16}$$
$$Y = TQ^T + F. \tag{17}$$

Here, the matrix X is decomposed into two matrices, $T \in \mathbb{R}^{n \times d}$ which is the matrix that produces d linear combinations (scores) and $P^T \in \mathbb{R}^{d \times p}$ which is referred as *X-loading*, plus an error matrix $E \in \mathbb{R}^{n \times p}$. Y is decomposed likewise into T, $Q^T \in \mathbb{R}^{d \times q}$ (*Y-loadings*) and $F \in \mathbb{R}^{n \times q}$.

The matrix T is estimated as the linear combinations

$$T = XW \tag{18}$$

where W are referred as the *weights*. These weights will be our projection matrix G. There are many different approaches of finding W but we have focused on the statistically inspired modification of PLS (SIMPLS). In [1] the criterion of SIMPLS is stated as

$$w_j = \underset{w}{argmax}\ w^T \sigma_{XY} \sigma_{XY}^T w \tag{19}$$

$$\text{subject to } w^T w = 1, \quad w^T \Sigma_{XX} w_j = 0, \quad \text{for } j = 1, \ldots, k-1$$

where w_j are the columns of W and σ_{XY} is the covariance of X and Y.

When T is estimated, loadings are estimated by *ordinary least squares* for the model (17).

The regression matrix for PLS is formulated as

$$\beta^{PLS} = WQ^T \tag{20}$$

since

$$Y \quad = \quad TQ^T + F = XWQ^T + F = X\beta^{PLS} + F.$$

Algorithm 1 SIMPLS formulated as in [3]

1: A_0=X^TY, M_0=X^TX, C_0=I
2: **for** i=1,. . . ,d **do**
3: Compute q_i, the dominant eigenvector of $A_i^T A_i$
4: $w_i = A_i q_i,\ c_i = w_i^T M_i w_i,\ w_i = \frac{w_i}{\sqrt{c_i}}$ and store w_i into W as a column
5: $p_i = M_i w_i$ and store p_i into P as a column
6: $q_i = A_i^T w_i$ and store q_i into Q as a column
7: $v_i = C_i p_i$ and $v_i = \frac{v_i}{||v_i||}$
8: $C_{i+1} = C_i - v_i v_i^T$ and $M_{i+1} = M_i - p_i p_i^T$
9: $A_{i+1} = C_i A_i$
10: **end for**
11: $T = XW$ and $B = WQ^T$

We want to have a separate projection matrix and a separate regression matrix, so our regression matrix will be

$$B = Q^T. \tag{21}$$

To sum things up, PLS regression is particularly useful when to predict a set of dependent variables where the number of independent variables is (very) large since the data X could be singular. It utilizes dimension reduction to find a smaller number of latent components that are linear combinations of the original variables to overcome the singularity.

PLS will not only capture the most variable directions in the X space. It also finds the directions that relate X to Y.

To find the dimension d we use cross-validation as in the former cases.

Part V

Results

We start by discussing the structure of the regression matrix and the projection matrix by analyzing the cross-validation result and after that we will check the model assumptions.

In the following results we have used an "explanatory" model, i.e. the independent variable X and the dependent variable Y are the same.

9 Structure of regression matrix and projection matrix

To determine the reduced dimensionality d we want to find the subset size α that gives the most parsimonious model whose error is no more than one standard error above the error of the best model.

We have divided the observations (months) into 10 nearly equally sized folds by using 206 observations and then applied cross-validation.

Figure 1: Prediction error for PCR

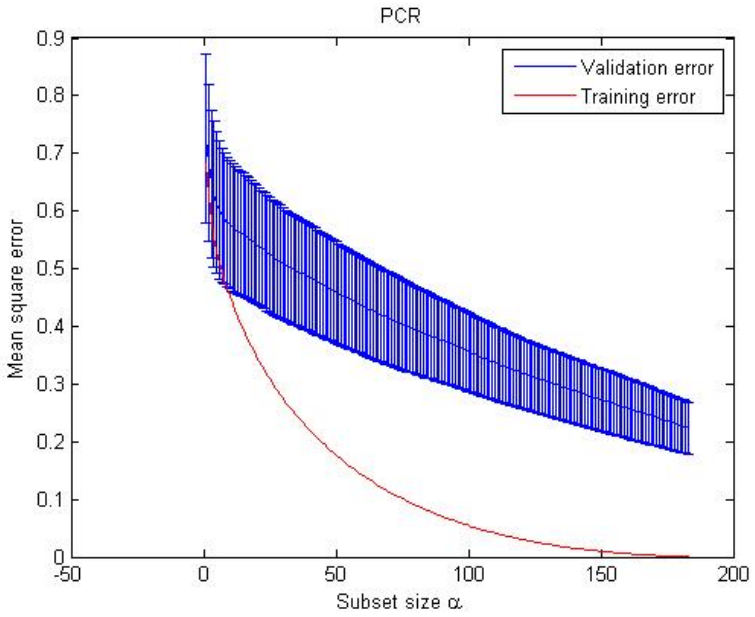


Figure 2: Prediction error for PLSR

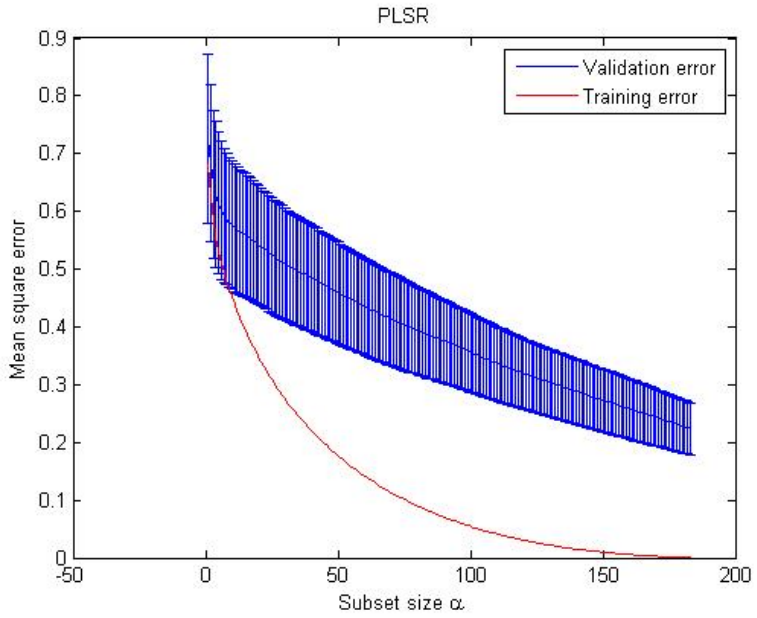
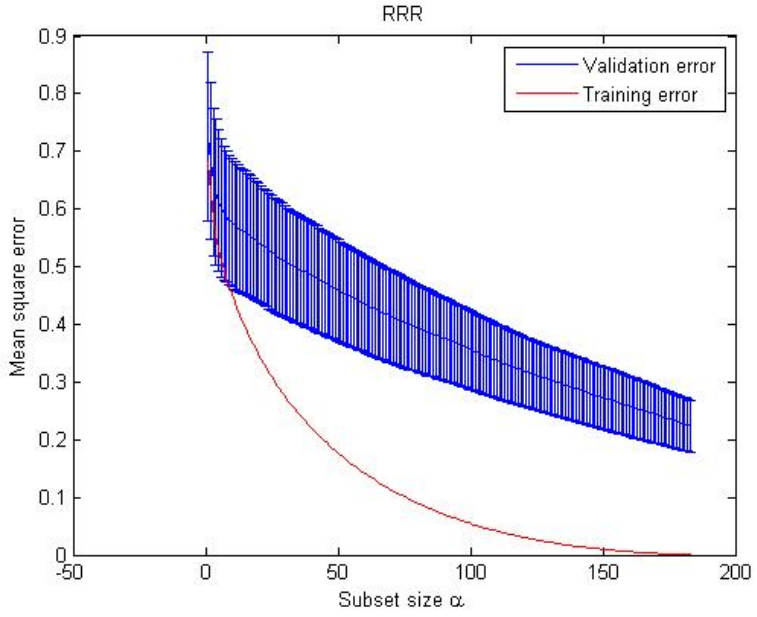


Figure 3: Prediction error for RRR

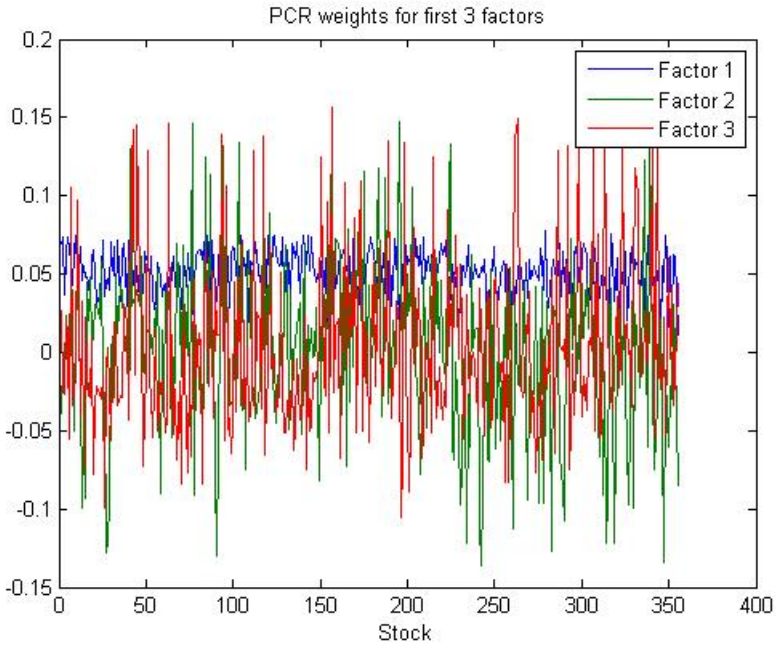


In the three plots above we can see that both the mean square error for the training model and the prediction error decrease when the number of components increases. The prediction error starts around 0.7 and decreases to around 0.2. When we use the one standard rule we get that we should choose d to be 154 for all three methods to avoid overfitting. This gives us the dimensions of the projection matrices

$$\begin{aligned} G^{RRR} &: 355 \times 154 \\ G^{PCR} &: 355 \times 154 \\ G^{PLSR} &: 355 \times 154 \end{aligned}$$

The projection matrices are very large and if you want to see them you can use the functions in the Appendix to simulate in MATLAB. However we have plot the first three columns, which will show the weights for the three first factors.

Figure 4:



The weights for factor 2 and factor 3 are both wide unlike factor 1 which only has positive weights. We can see some peaks in the plot and that indicates that a higher weight is used for that stock, for example the weights for factor 3 has a peak around stock 150, thus stock 150 will affect factor 3 more than some others.

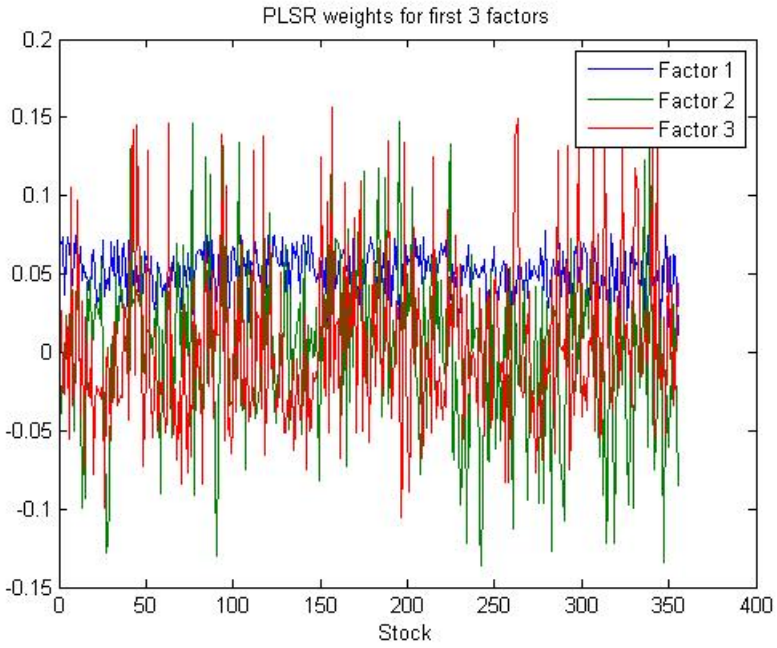
We have also grouped the stocks by the *Global industry classification standard* to check how each of the ten sectors affect the five first factors. To get a measure for each sector we will use the mean square of the weights of each stock in the specific sector.

Table 1: Mean squares for each sector with PCR

	Factor 1	Factor 2	Factor 3	Factor 4	Factor 5
Energy	0.0030	0.0033	0.0034	0.0031	0.0025
Materials	0.0030	0.0018	0.0029	0.0026	0.0044
Industrials	0.0034	0.0032	0.0023	0.0023	0.0014
Consumer Dis.	0.0024	0.0033	0.0024	0.0029	0.0035
Consumer Sta.	0.0027	0.0023	0.0027	0.0031	0.0037
Health care	0.0026	0.0025	0.0044	0.0032	0.0022
Financials	0.0025	0.0034	0.0029	0.0027	0.0028
IT	0.0030	0.0025	0.0017	0.0032	0.0034
Tele	0.0029	0.0013	0.0038	0.0017	0.0045
Utilities	0.0031	0.0015	0.0033	0.0025	0.0016

Here we have used PCR´s projection matrix to calculate the mean square of the weights for each sector. The values are between 0.0013 and 0.0045, at least for the first five factors.

Figure 5:



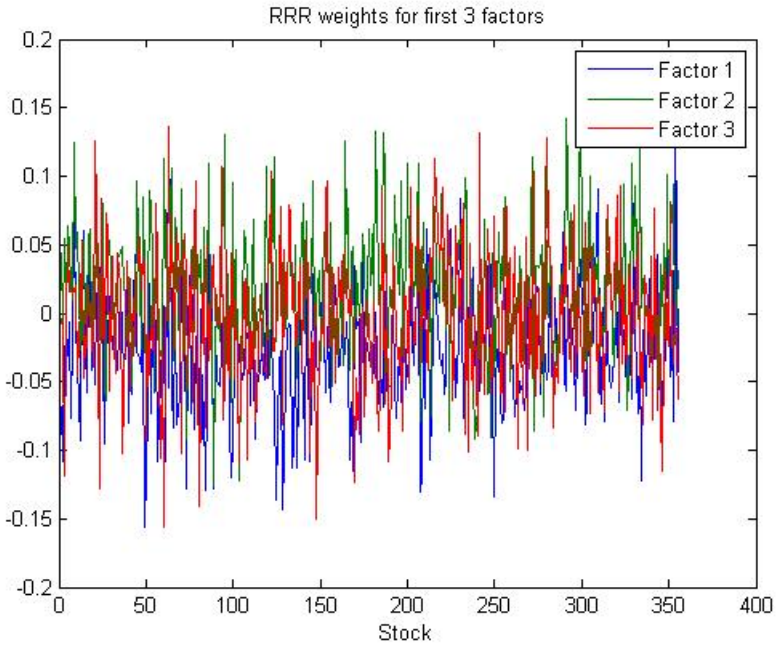
The weights for PLSR are very similar to figure 4. Later we will see why the weights for PLSR are so similar to PCR.

Table 2: Mean squares for each sector with PLSR

	Factor 1	Factor 2	Factor 3	Factor 4	Factor 5
Energy	0.0030	0.0033	0.0034	0.0031	0.0025
Materials	0.0030	0.0018	0.0029	0.0026	0.0044
Industrials	0.0034	0.0032	0.0023	0.0023	0.0014
Consumer Dis.	0.0024	0.0033	0.0024	0.0029	0.0035
Consumer Sta.	0.0027	0.0023	0.0027	0.0031	0.0037
Health care	0.0026	0.0025	0.0044	0.0032	0.0022
Financials	0.0025	0.0034	0.0029	0.0027	0.0028
IT	0.0030	0.0025	0.0017	0.0032	0.0034
Tele	0.0029	0.0013	0.0038	0.0017	0.0045
Utilities	0.0031	0.0015	0.0033	0.0025	0.0016

Same values as for PCR.

Figure 6:



For RRR the plot is similar to figure 4 and 5, but the weights for factor 1 are a bit wider here. One weight for factor 1 is as low as -0.15, unlike the weights for factor 1 in PCR and PLSR which all are positive.

Table 3: Mean squares for each sector with RRR

	Factor 1	Factor 2	Factor 3	Factor 4	Factor 5
Energy	0.0023	0.0040	0.0033	0.0014	0.0021
Materials	0.0038	0.0020	0.0015	0.0037	0.0028
Industrials	0.0023	0.0035	0.0026	0.0031	0.0022
Consumer Dis.	0.0025	0.0024	0.0025	0.0028	0.0022
Consumer Sta.	0.0040	0.0023	0.0027	0.0033	0.0031
Health care	0.0019	0.0030	0.0035	0.0030	0.0034
Financials	0.0025	0.0028	0.0029	0.0030	0.0044
IT	0.0038	0.0023	0.0033	0.0028	0.0016
Tele	0.0038	0.0019	0.0028	0.0023	0.0033
Utilities	0.0035	0.0030	0.0028	0.0017	0.0032

The mean squares of the weights are in the same range as for PCR and PLSR, but not completely the same.

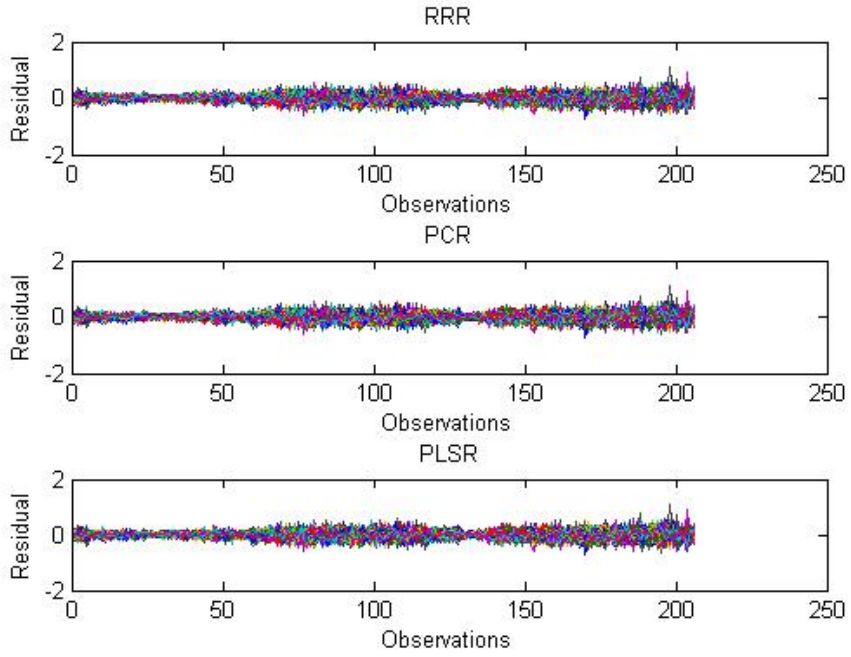
When analyzing the projection matrices and the regression matrices we can see that PCR and PLSR will have the same G and B while RRR will not have the same as the others, but all three methods will have the same predictors $\hat{Y} = XGB$, i.e. GB is the same for each method.

10 Model assumptions

For the following calculations we have used $d = 154$ in all cases.

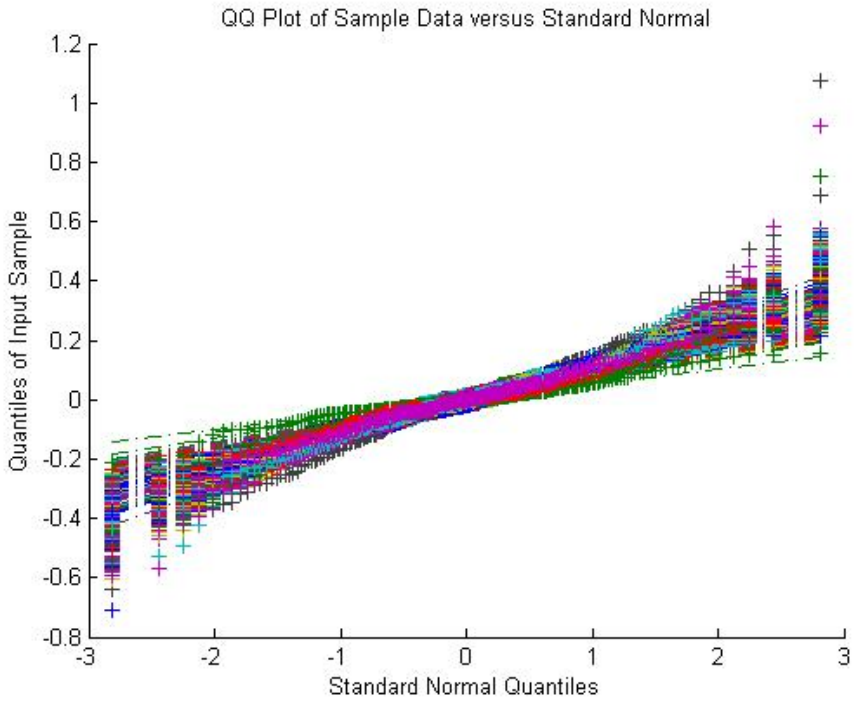
First we look at some plots for the residuals, $Y - XGB$, for the different methods.

Figure 7: Residual plots



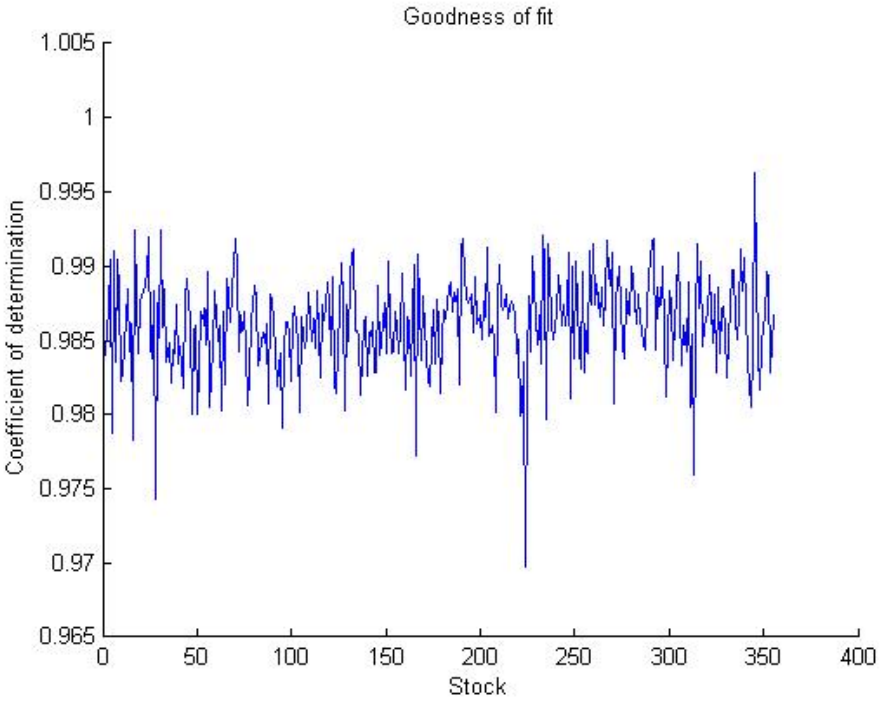
We can see that the residuals for each of the methods behave similarly, not a surprise since the predictors are the same. There are many curves (one for each stock) and it is hard to see exactly how they behave but they are not bigger than ± 2. The variance for the residuals of each stock is in size with 0.015 and the mean square error for each method is 0.0137.

Figure 8:



We will only show the quantile-quantile plot for one method since they all have the same residuals. We can see that it looks like a straight line but with heavy tails. Whether it follows a normal distribution or not is hard to tell, but we can see that it doesn't behave very hooked or ill-shaped. The heavy tails usually indicate that there are a high kurtosis. The mean of all kurtosises is 4.0988.

Figure 9:



We have plot the coefficient of determination (R^2) for each method but they are similar so we will only show one.

R^2 is a statistic that will give some information about the goodness of fit of a model and the definition of it is to determine the proportion of variance "explained" by the regression model. This makes it useful as a measure of success of predicting the dependent variable from the independent variables.

We can see that the coefficient of determination are high. It is around 0.985 in general but for some stocks it is higher, see for example stock 345 it is as high as 0.9962. So, about 98% of the variance is explained by the regression model.

The explained variance we mentioned in the beginning is 0.9862 which is very similar to the variance explained by the regression model.

Part VI

Conclusion

With the different methods we could make our data which was "too large" to a much smaller data set by capture a smaller part of the explained variance. With principal component regression we could get Z (d-dimensional subspace) that has the dimensions 206×154 instead of X which has 206×355 and still preserve 98% of the information. Same size on Z for reduced rank regression and for partial least squares regression. This result is not expected in an economic perspective, the reduced dimensionality is too large. In a simple economic model you maybe use 5-10 factors which for example could be different market factors.

That the mean square error decreases when we are using bigger subsets is expected since the more components that are used the more accurate the regression will be. On the other hand that the prediction error would decrease so much is surprising, often when the variance is high it will affect the prediction error and the prediction error will increase when using many components.

It could be interesting to know how each stock contributes to each factor and in figure 4-6 we can see how the stocks affect the three first factors in each method by analyzing the weights. The weights for factor 1 in figure 4 and 5 are smaller and not as wide as for factor 2 and factor 3 and that could depend on that often factor 1 measures the "general market conditions" and the weights should be about the same. In table 1 we can see how each sector affects the first five factors. The measures for factor 1 in the different sectors are pretty much the same. In factor 2, for example energy and financials are a little bit bigger than telecommunication services. There are not some sectors that are much more dominant than the others, sometimes some measures are about 2 times bigger than some measures, at least for the first five factors. There is a small difference in figure 6 and table 3, the measures are a little bit more spread for factor 1 than for the other tables and figures.

That the projection matrices and the regression matrices would not be equal for RRR and PCR is not so surprising since they are using different methods, but that the predictors are the same is a little bit surprising. Also a surprise was that PLSR and PCR had the same G and B, for example, PCR only preserves the most variable directions in the X space and not taking any care about the Y variable and has the same GB as for example PLSR, which also takes care about the relationship between Y and X. At first sight it seems impossible that RRR which has different projection and regression matrix has the same predictor as PCR and PLSR, since the column space of $B^T G^T$ is uniquely determined and so will the matrix GG^T which projects from the space $\mathbb{R}^p$ to the subspace of $\mathbb{R}^p$. However, the d-dimensional subspace of $\mathbb{R}^p$ is not uniquely determined by how it is identified with $\mathbb{R}^d$, thus we can add an orthonormal matrix, call it $H \in \mathbb{R}^{d \times d}$,

and write the projection matrix as GH without that something changes

$$(GH)(GH)^T = GHH^TG^T = GIG^T = GG^T.$$

So, we can have different projection matrices in different methods but GG^T must be the same and by checking this with our projection matrices, GG^T is the same for all three methods.

In the beginning we assumed that the log-returns were distributed as a normal distribution and that a high kurtosis could occur. The kurtosis is about 4 and it is a little to high to say directly that there is a normal distribution since normal distributions has a kurtosis of 3.

Goodness of fit for the different methods are the same since the residuals are the same. It is pretty high which could indicate that the model is good in some sense, at least about 98-99% of the variance is explained by the regression model.

In an explanatory model the predictors will be the same in each method, so the accurancy will not differ from method to method and the methods are equal in that sense. However, we like principal component regression better since the calculations are easier and not as heavy as the other methods. For example when to find the projection matrix it only uses the independent variables.

In summary, when to predict monthly log-returns on the American stock market, Standard & Poor's 500, using a smaller set of factors in an explanatory model, we would use *Principal component regression* to construct these factors for further analysis.

Part VII

Acknowledgements

I would like to say my sincerest thanks to my supervisors Tobias Rydén, Lina von Sydow, Josef Höök, Elisabeth Larsson and Per Lötstedt for their ideas and commitment through this project. I would also like to thank my examinator Rolf Larsson for his thoughts.

Part VIII

References

[**1**] H. Chun and S. Keleşi. *Sparse partial least squares regression for simultaneous dimension reduction and variable selection.*

[**2**] J. Leskovec, A. Rajaraman and J.D. Ullman (2014). *Mining of Massive Datasets.* 2nd ed. Cambridge University Press.

[**3**] Online Statistics Education: A Multimedia Course of Study *(http://onlinestatbook.com/).* Project Leader: D. M. Lane, Rice University.

[**4**] T. Hastie, R. Tibshirani and J. Friedman (2009). *The Elements of Statistical Learning. Data Mining, Inference, and Prediction,* 2nd ed. Springer.

[**5**] Y. Hua. M. Nikpour and P. Stoica (2001). *Optimal reduced-rank estimation and filtering.* IEEE Trans. Signal Process. vol. 49**.**

Part IX

Appendix

The following function gets the data we want from an excel file (which contains all stocks with prices from 1998 with corresponding trade days on S&P 500) and then calculates the log-returns.

```
function [dataMonthly, logR]=reduce1998

clear all                                          % Clear workspace

format short                                       % View four decimals

orgdata=xlsread('SP500 reduced.xlsx');             % Read data

dataWoDa=orgdata(2:end,1:2:end);                   % Choose only values, not dates

[m1,n1]=size(dataWoDa);                            % Number of obs and stocks

data=zeros(4330,n1);                               % Preallocate

for i=1:n1

    mend=m1;

    while (isnan(dataWoDa(mend,i)))

        mend=mend-1;

    end

    data(:,i)=dataWoDa(mend-4329:mend,i);

end                                                %Remove NaN

[m2,n2]=size(data);                                % Number of obs and stocks
                                                   % without NaN values
x=cumsum([0 20 19 22 21 20 22 22 21 21 22 ...
          20 22 19 19 23 21 20 22 21 22 21 ...
          21 21 22 20 20 23 19 22 22 20 23 ...
          20 22 21 20 21 22 19 20 22 21 21 ...
          23 15 23 21 20 21 19 20 22 22 20 ...
          22 22 20 23 20 21 21 19 21 21 21 ...
          22 21 21 21 23 19 22 20 19 23 21 ...
```

```
          20 21 21 22 21 21 21 22 20 19 22 ...
          21 21 22 20 23 21 21 21 21 20 19 ...
          23 19 22 22 20 23 20 22 21 20 20 ...
          19 22 20 22 21 21 23 19 23 21 20 ...
          21 20 20 22 21 21 22 21 21 23 19 ...
          22 20 19 22 21 20 22 22 21 21 22 ...
          20 22 19 19 23 21 20 22 21 22 21 ...
          21 21 22 20 19 23 20 21 22 20 23 ...
          21 21 21 21 20 20 22 20 22 21 21 ...
          23 19 21 21 20 21 19 20 22 22 20 ...
          22 22 20 23 20 21 21 19 21 21 21 ...
          21 22 21 21 23 19 22 20 19])+1;      % Index for first tradeday in each month

dataMonthly=zeros(207,n2);                     % Preallocate

for j=1:n2

     dataMonthly(:,j)=data(x,j);

end                                            % One value per month

[m3,n3]=size(dataMonthly);                     % Number of obs (one per month) and stocks

logR=zeros(m3-1,n3);                           % Preallocate

for k=1:n3

    for l=1:m3-1

        logR(l,k)=log(dataMonthly(l+1,k)/dataMonthly(l,k));

    end
end                                             % Calculate log return
```

Performs principal component regression.

```
function [X, Y, B, G, MSE, R_square]=pcr(x, y, nrComp)

X=zscore(x);                              % Standardized X-values

Y=zscore(y);                              % Standardized Y-values

[~,m]=size(Y);                            % Size of Y-data

[coeff,~,~,~,explained] = pca(X);         % Calculate pc
```

```
d=nrComp;                                % Reduced dimensionality
CumExp=cumsum(explained);                % Expl. variances
expvar=CumExp(d)                         % Expl. variance for d-comp.
G=coeff(:,1:d);                          % Projection matrix
Z=X*G;                                   % Reduced matrix
B=(Z'*Z)^-1*Z'*Y;                        % Regression matrix
ressq=(X*G*B).^2;                        % Squared errors
MSE=mean(ressq(:));                      % Mean square error
if (m==1)
    SStot=sum((Y-mean(Y)).^2);           % Total sum of squares
    SSres=sum((Y-X*G*B).^2);             % Residual sum of squares
    R_square=1-SSres/SStot;              % Coefficient of determination
else
    SStot=zeros(1,m);                    % Preallocate space
    SSres=zeros(1,m);                    % Preallocate space
    R_square=zeros(1,m);                 % Preallocate space
    for i=1:m
        SStot(i)=sum((Y(:,i)-mean(Y(:,i))).^2);  % Total sum of squares
        SSres(i)=sum((Y(:,i)-X*G*B(:,i)).^2);    % Residual sum of squares
        R_square(i)=1-SSres(i)/SStot(i);         % Coefficient of determination
    end
end
```

Performs partial least squares regression.

```
function [X, Y, B, G, MSE, R_square, expvar]=plsr(x, y, nrComp)
X=zscore(x);                                % Standardized X-values
Y=zscore(y);                                % Standardized Y-values
[~,m]=size(Y);                              % Size of Y-data
d=nrComp;                                   % Reduced dimensionality
[~,~,~,~,B_pls,PCTVAR,~,stats] = plsregress(X,Y,d);     % PLSR with d-comp.
cumexp=cumsum(PCTVAR(1,:));                 % Expl. variances
expvar=cumexp(end)*100;                     % Expl. variance for d-comp.
G=stats.W;                                  % projection matrix
for i=1:size(G,2)
    G(:,i)=G(:,i)/norm(G(:,i));             % Orthogonalize
end
B=G\B_pls(2:end,:);                         % Regression matrix
ressq=(X*G*B-Y).^2;                         % Squared errors
MSE=mean(ressq(:));                         % Mean square error
if (m==1)
    SStot=sum((Y-mean(Y)).^2);              % Total sum of squares
    SSres=sum((Y-X*G*B).^2);                % Residual sum of squares
    R_square=1-SSres/SStot;                 % Coefficient of determination
else
    SStot=zeros(1,m);                       % Preallocate space
    SSres=zeros(1,m);                       % Preallocate space
    R_square=zeros(1,m);                    % Preallocate space
```

```
    for i=1:m
        SStot(i)=sum((Y(:,i)-mean(Y(:,i))).^2);  % Total sum of squares
        SSres(i)=sum((Y(:,i)-X*G*B(:,i)).^2);    % Residual sum of squares
        R_square(i)=1-SSres(i)/SStot(i);         % Coefficient of determination
    end
end
```

Performs reduced rank regression.

```
function [X, Y, B, G, MSE, R_square, expvar]=rrr(x, y, nrComp)
X = zscore(x);                              % Standardized X-values
Y=zscore(y);                                % Standardized Y-values
[m1,n1]=size(X);                            % Size of X-data
[m2,n2]=size(Y);                            % Size of Y-data
d=nrComp;                                   % Reduced dimensionality
[~, s, ~]=svd(X);                           % Singular value decomposition
ssq=cumsum(diag(s).^2);                     % Sum of squares
ssqproc=ssq/ssq(end);                       % Calculate procent of ssq
expvar=ssqproc(d)*100;                      % Expl. variance for d-comp.
Cyx=(Y'*X)/(m1-1);                          % Corr. matrix y and x
Cxx=(X'*X)/(m1-1);                          % Corr. matrix of x
[E, D]=eigs(Cxx,n1);                        % Eigenvalue decomp.
Rank=rank(D);                               % Rank off D
dSqrtInv=D(1:Rank,1:Rank)^(-1/2);           % Squareroot & inverse
Diag=diag(dSqrtInv);                        % Diagonal of dSqrtInv
```

```
Diag=[Diag;zeros(n1-Rank,1)];              % Add zeros to diagonal
DsqInv=diag(Diag,0);                       % Diagonalmatrix for Sq. & Inv.
CxxSqInv=E*DsqInv*E^-1;                    % Squareroot & inverse of Cxx
R_tr=Cyx*(CxxSqInv)';                      % Matrix R_tr
[~, ~, V]=svd(R_tr);                       % SVD of matrix R_tr
V1=(V(:,1:d));                             % First d right singular vectors
B=(R_tr*V1)';                              % Regression matrix
G=(V1'*(CxxSqInv))';                       % Projection matrix
T=G*B;                                     % Composted matrix
[u1, s1, v1]=svd(T);                       % Singular value decomposition
Rank=rank(s1);                             % Rank of diagonal matrix s1
if Rank<d
   for i=1:Rank
       G(:,i)=G(:,1)/norm(G(:,i));         % Orthogonalize
   end
else
   G=v1(:,1:Rank);                         % Orthogonalize projection matrix
   B=(u1(:,1:Rank)*s1(1:Rank,1:Rank))';    % Corresponding regress. matrix
end
ressq=(X*G*B-Y).^2;                        % Squared errors
MSE=mean(ressq(:));                        % Mean square error
if (n2==1)
    SStot=sum((Y-mean(Y)).^2);             % Total sum of squares
```

```
    SSres=sum((Y-X*G*B).^2);                % Residual sum of squares
    R_square=1-SSres/SStot;                 % Coefficient of determination
else
    SStot=zeros(1,n2);                      % Preallocate space
    SSres=zeros(1,n2);                       % Preallocate space
    R_square=zeros(1,n2);                    % Preallocate space
    for i=1:n2
        SStot(i)=sum((Y(:,i)-mean(Y(:,i))).^2);  % Total sum of squares
        SSres(i)=sum((Y(:,i)-X*G*B(:,i)).^2);    % Residual sum of squares
        R_square(i)=1-SSres(i)/SStot(i);         % Coefficient of determination
    end
end
```

The following function plots the figures that we want.

```
function plots
load('data.mat')      % Load in data
x=logR;               % Define X values
y=logR;               % Define Y values
%%%%%% Calculations with multivariate respons variable %%%%%%
nrComp1=154;
[X1_rrr, Y1_rrr, B1_rrr, G1_rrr, MSE1_rrr, R_square1_rrr]=...
rrr(x, y, nrComp1);
[X1_pcr, Y1_pcr, B1_pcr, G1_pcr, MSE1_pcr, R_square1_pcr]=...
pcr(x, y, nrComp1);
```

```
[X1_plsr, Y1_plsr, B1_plsr, G1_plsr, MSE1_plsr, R_square1_plsr]=...
plsr(x, y, nrComp1);

%%%%%% Calculations with single respons vector %%%%%%
yU=logR(:,1);

nrComp2=154;

[X2_rrr, Y2_rrr, B2_rrr, G2_rrr, MSE2_rrr, R_square2_rrr]=...
rrr(x, yU, nrComp2);

[X2_pcr, Y2_pcr, B2_pcr, G2_pcr, MSE2_pcr, R_square2_pcr]=...
pcr(x, yU, nrComp2);

[X2_plsr, Y2_plsr, B2_plsr, G2_plsr, MSE2_plsr, R_square2_plsr]=...
plsr(x, yU, nrComp2);

%%%%%% R-squares for the different methods %%%%%%

figure()

hold on

plot(R_square1_rrr)

title('Goodness of fit')

xlabel('Stock')

ylabel('Coefficient of determination')

%%%%%% Residuals for the different methods %%%%%%

figure()

subplot(3,1,1)

plot(Y1_rrr-X2_rrr*G1_rrr*B1_rrr);

xlabel('Observations');

ylabel('Residual');

title('RRR')
```

```
subplot(3,1,2)

plot(Y1_pcr-X1_pcr*G1_pcr*B1_pcr);

xlabel('Observations');

ylabel('Residual');

title('PCR')

subplot(3,1,3)

plot(Y1_plsr-X1_plsr*G1_plsr*B1_plsr);

xlabel('Observations');

ylabel('Residual');

title('PLSR')

%%%%%% qq-plots different methods %%%%%%

figure()

qqplot(Y1_rrr-X2_rrr*G1_rrr*B1_rrr);

%%%%%% Residuals for single respons vector %%%%%%

figure()

subplot(3,1,1)

plot(Y2_rrr-X2_rrr*G2_rrr*B2_rrr)

xlabel('Observations');

ylabel('Residual');

title('Single respons vector for RRR')

subplot(3,1,2)

plot(Y2_pcr-X2_pcr*G2_pcr*B2_pcr,'r')

xlabel('Observations');
```

```
ylabel('Residual');

title('Single respons vector for PCR')

subplot(3,1,3)

plot(Y2_plsr-X2_plsr*G2_plsr*B2_plsr,'g')

xlabel('Observations');

ylabel('Residual');

title('Single respons vector for PLSR')

%%%%% Weights for the different methdos %%%%%

figure

plot(G1_rrr(:,1:3))

title('RRR weights for first 3 factors')

xlabel('Stock')

legend('Factor 1','Factor 2','Factor 3')

figure

plot(G1_pcr(:,1:3))

title('PCR weights for first 3 factors')

xlabel('Stock')

legend('Factor 1','Factor 2','Factor 3')

figure

plot(G1_plsr(:,1:3))

title('PLSR weights for first 3 factors')

xlabel('Stock')

legend('Factor 1','Factor 2','Factor 3')
```

Performs cross-validation and plot result.

```
function [mseT, mse]=cvK(x,y,N,method)

K=10;

ind=crossvalind('Kfold', N, K);    % Generate index for cross-val.

nrComp=20;                         % Number of components

totalSS=zeros(nrComp,1);           % Preallocate

totalSSt=zeros(nrComp,1);          % Preallocate

mse=zeros(nrComp,1);               % Preallocate

mseT=zeros(nrComp,1);              % Preallocate

meanVal=zeros(K,1);                % Preallocate

NrElem=zeros(K,1);                 % Preallocate

NrElemT=zeros(K,1);                % Preallocate

SE=zeros(nrComp,1);                % Preallocate

for k=1:nrComp

    for j=1:K

        trainInd = (ind ~= j);     % Training index

        valInd = (ind == j);       % Validation index

        trainX = x(trainInd,:);    % Training data

        valX = x(valInd,:);        % Validation data

        trainY = y(trainInd,:);    % Training response

        valY = y(valInd,:);        % Validation response

        [trainX,meanX,stdX] = zscore(trainX);  % Standardize

        [trainY,meanY,stdY] = zscore(trainY);  % Standardize
```

```
        if strcmp(method,'pcr')
           [X, Y, B, G, ~]=pcr(trainX,trainY,k);
        elseif strcmp(method,'plsr')
           [X, Y, B, G, ~]=plsr(trainX,trainY,k);
        else
           [X, Y, B, G, ~]=rrr(trainX,trainY,k);
        end
        valX = bsxfun(@times,bsxfun(@minus,valX,meanX),1./stdX); % Standardize
        valY = bsxfun(@times,bsxfun(@minus,valY,meanY),1./stdY); % Standardize
        resSqT=(X*G*B-Y).^2;                    % Training squared errors
        resSq=(valX*G*B-valY).^2;               %Validation squared errors
        totalSSt(k)=totalSSt(k)+sum(resSqT(:)); % Total sum of squares tr.
        totalSS(k)=totalSS(k)+sum(resSq(:));    % Total sum of squares val.
        meanVal(j)=mean(resSq(:));              % Avg. of the val. error in each fold
        NrElem(j)=numel(resSq);                    % Number of elements for val.
        NrElemT(j)=numel(resSqT);                  % Number of elements for tr.
    end
    mse(k)=totalSS(k)/sum(NrElem);     % MSE for val.
    mseT(k)=totalSSt(k)/sum(NrElemT);  % MSE for tr.
    SE(k)=sqrt(var(meanVal))/sqrt(K);  % Standard deviation of each part
end
errorbar(mse,SE);
hold on
```

```
plot(1:nrComp,mseT,'r')

hold off

xlabel('Subset size \alpha')

ylabel('Mean square error')

legend('Validation error','Training error')
```