Universiteit van Amsterdam
Amsterdam School of Economics

Machine Learning in Finance

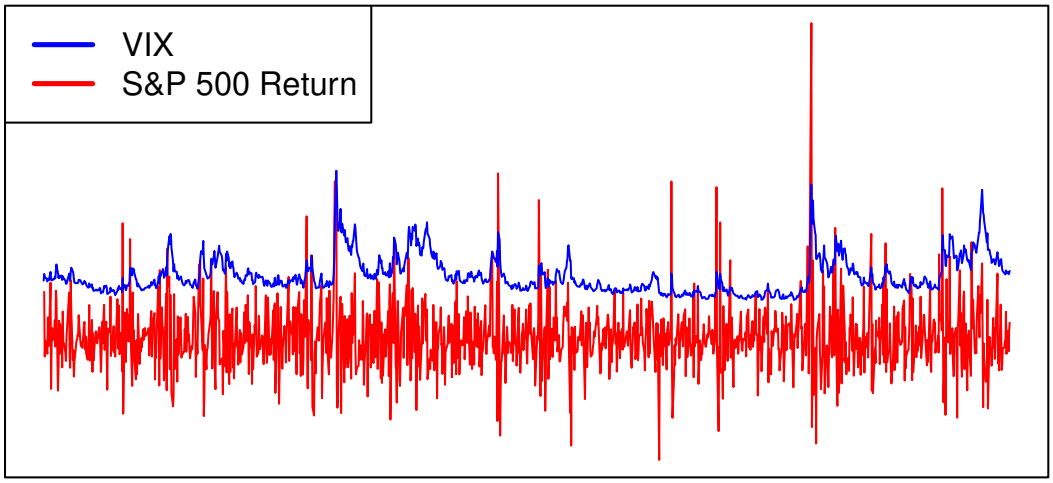
Assignment 1

AJ* and SU†

Spring 2022

*Student number: ∞
†Student number: π

Developing S&P 500 Trading Strategies based on the CBOE Volatility Index (VIX)



Executive Summary

Volatility comes in two flavours: past and future. By inferring the 30-day expected volatility of the S&P 500 index through observed option prices, the **CBOE S&P 500 Volatility Index (VIX)** is the primary mechanism of gauging market sentiment.

As a forward-looking metric, the question arises as to whether the the information contained in the VIX can be translated into actionable trading signals on the S&P 500 itself.

This report develops methods of answering two specific variations of this question; namely, can we predict the daily log return of the S&P 500 and in which direction will the market move, given the VIX on the same day. The report details the models, methodology, results and recommendations of the analysis, with technical notes appended for domain experts and interested readers.

The data indicates that a K-nearest neighbor (KNN) method for classifying market movements may provide useful trading signals on the S&P 500. The limitations of these findings are also discussed in detail. Although the initial findings show promise, there remain many unexplored techniques, which may need to be investigated in subsequent reports.

The Dataset

The dataset analysed throughout this report is contained in the comma-separated value (**.csv**) file **VIXSP500**, which accompanies this report.

It contains 1258 observations, where each observation comprises the level of the VIX and S&P500 on a given day, with the corresponding simple (R_t)[1] and log-return (r_t)[2] implied by the change in the equity index.

Technical Note 1 details the import and processing of the data in `R`.

Notation and Model Representation

Initially, we define our target variable Y_t to be the log-returns r_t, which we wish to predict using the feature variable X_t, being the VIX on the same day *i.e.* at time t.

In its most general form, our model may be written as:

$$Y = f(X) + \epsilon$$

where the structure present in the data is represented by the function f.

We will use two methods[3]; namely, K-nearest neighbour and linear models, to estimate this function (denoted $\hat{f}$). In modelling Y, we recognise that the data contains a random component ϵ, where $\epsilon \sim \mathrm{N}(0, \sigma^2)$. This is an initial modelling assumption regarding the distribution of the error term, and will be revisited later in the analysis.

Having modelled the log-returns r_t as the target variable Y_t, we modify our approach. The target variable Y_t is redefined as a categorical variable:

$$Y_t = \begin{cases} 0, & \text{if } R_t \leq 0.005 \\ 1, & \text{if } R_t > 0.005 \end{cases}$$

which we still wish to model using the VIX as a predictor (X_t). The objective is to accurately classify downward movements of the market as $Y_t = 0$ and upward movements as $Y_t = 1$, given X_t. The initial threshold of 0.005 is somewhat arbitrary, and the practical implications of choosing different thresholds will be discussed.

[1] $R_t = (V_t - V_{t-1}) / V_{t-1}$

[2] $r_t = log_e (V_t / V_{t-1})$

[3] See *Technical Note 2,3* for a detailed mathematical description of the respective method.

Implementation and Sampling Methods

The full R code to implement the KNN method for the target variable Y_t as log-returns (r_t) and the classification of Y_t as the market movement can be found in *Technical Note 4*.

Given that the dataset contains 1258 observations, which is small compared to most financial datasets, we have used a variety of resampling methods to obtain the most robust predictions and estimates of the optimal model parameters.

Figure 1: Measure of how the error in our predicted value changes with different choices of K, trained on the full test set.

Our chosen metric of determining how well our model fits the data is the Mean Squared Error (MSE), which measures how far our estimate is from the true value, on average.

In order to ensure our model will perform well when it encounters new data, and given that we wish to fit our model using only the dataset provided, we need to use resampling methods to determine robust estimates.

In the first method, the *validation set approach*, the data has been randomly split into a training set and a validation set, in a ratio of 80:20. We then choose the value of K that minimises this value.

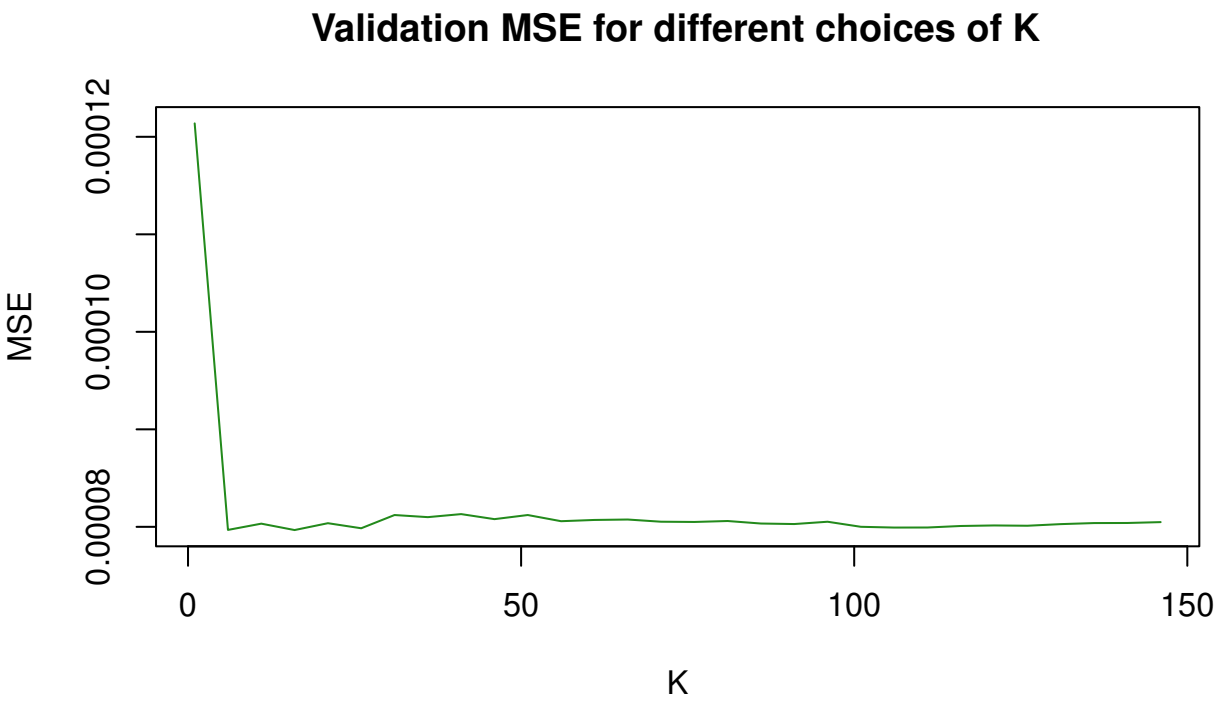


Figure 2: Prediction error on unseen data for different choices of K.

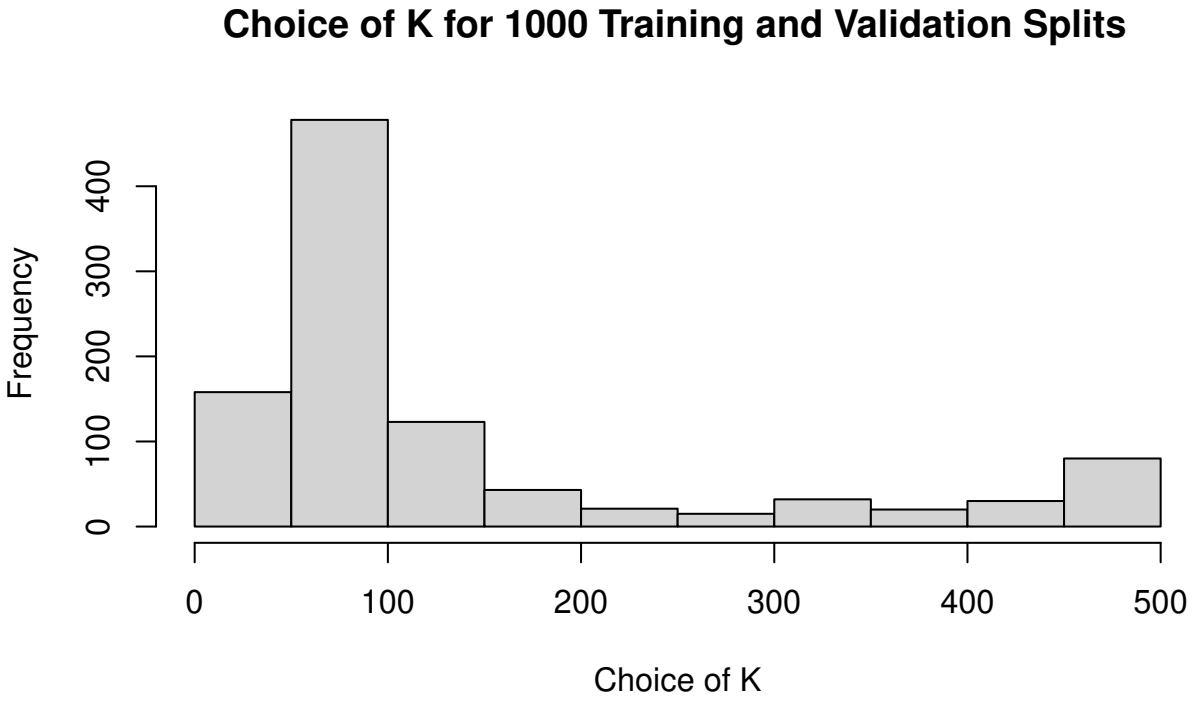


Figure 3: The optimal choice of K, using the MSE as the error metric, for one thousand random splits in the validation set approach.

Figure 3 would indicate that our optimal choice of K is in the region of 100. Intuitively, when we repeat this experiment many times, it would appear this is the region that would perform best.

Next, we implement *Leave-One-Out-Cross-Validation* (LOOCV). The idea behind this approach is to use as much of data as possible in the training of the model. Please refer to *Technical Note 2* for a precise definition.

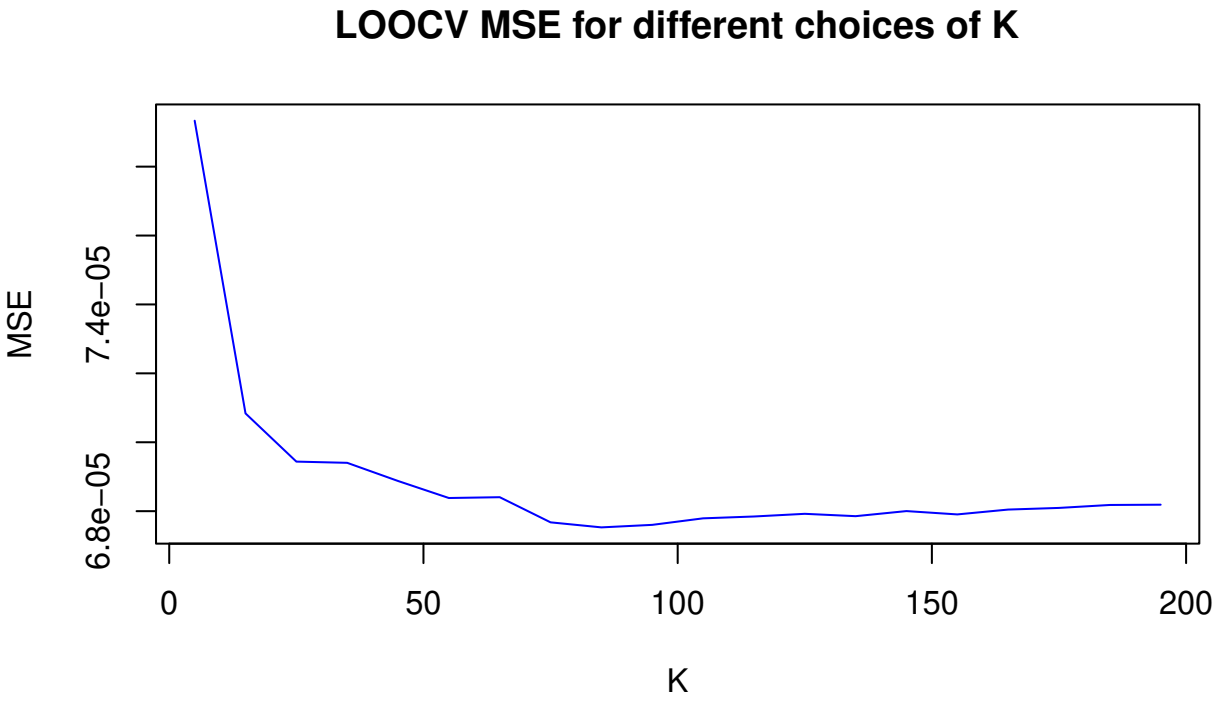


Figure 4: Prediction error for different choices of K using LOOCV.

We find that the optimal choice of K under this approach is approximately 85.

Finally, we use the *K-Fold Cross-Validation* approach to estimate K. This approach can be viewed as a middle-ground between the two previous approaches, where we split the data into K-chunks and use one chunk to validate our model.

Figure 5 on the next page illustrates the results. The choice of K for both methods is 65. Viewing all these results together, we proceed with the estimate of $K = 85$. We note that this choice is somewhat arbitrary, and the implications of this will be discussed in the *Limitations and Extensions* section.

In addition, the use of each resampling method and its effect on the predictions will be discussed under *Statistical Inference*.

We now fit the model using our optimal value of $K = 85$ on the next page.

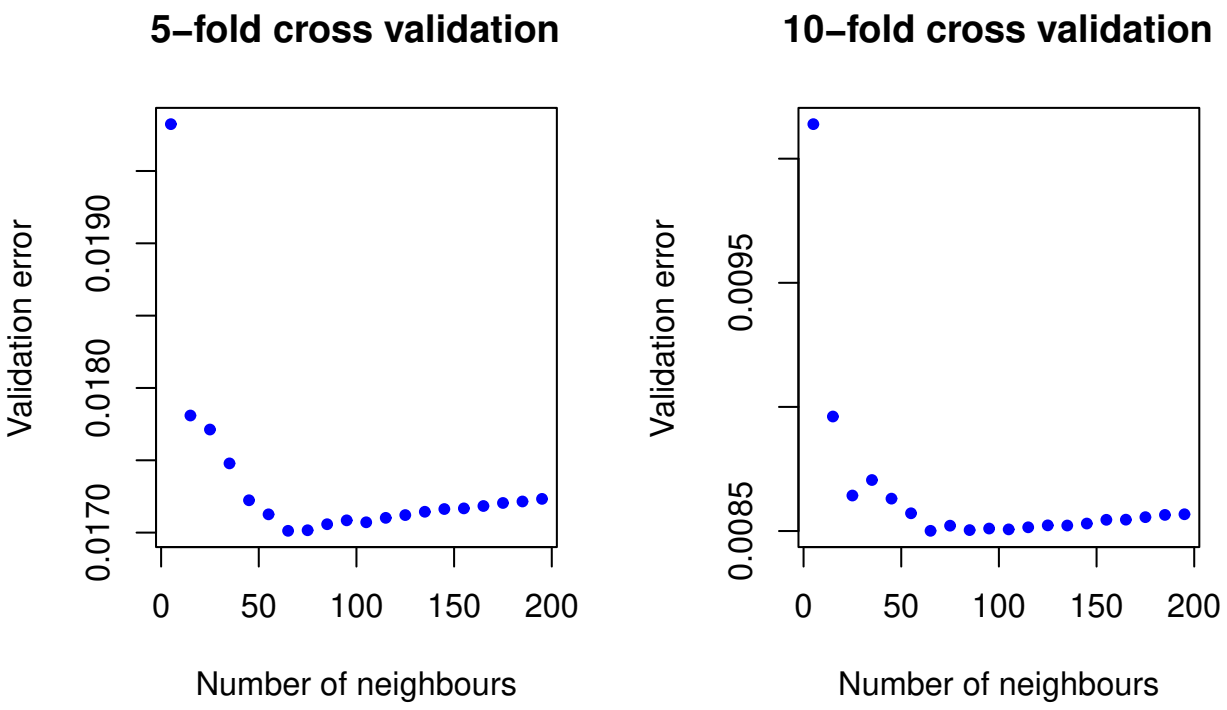


Figure 5: Prediction Error for $Y_t = r_t$ for different choices of K using Cross-Validation.

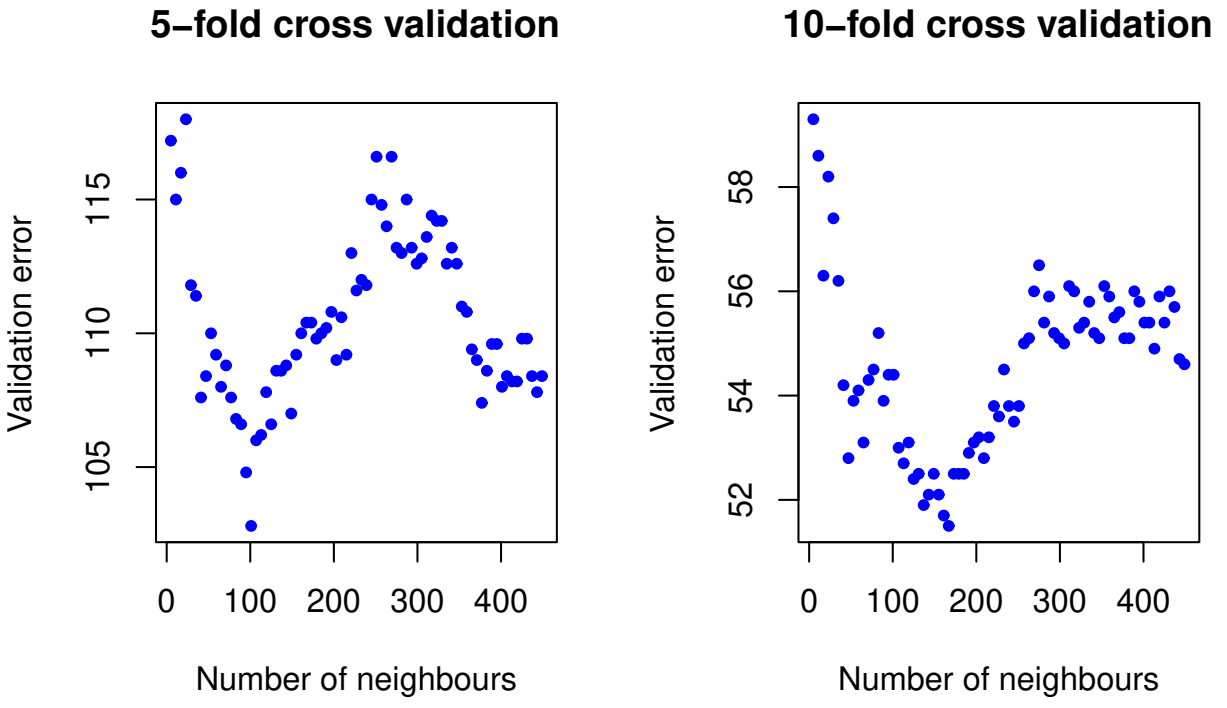


Figure 6: Prediction Error for $Y_t = \{0, 1\}$ for different choices of K using Cross-Validation.

Switching our target variable Y_t to a categorical variable, we implement an adjusted version of the KNN algorithm.[4]

Our choice of K in this context is investigated using K-Fold Cross-Validation for $K = \{5, 10\}$. The results are displayed in *Figure 6* on the previous page. The 5 and 10-fold datasets are minimised for $K = 101, 167$ respectively. Based on these results and the discussion found in *Technical Note 2*, our choice of K is 125. Implementing KNN with this value of K on the entire dataset, we have:

	0	1
0	610	106
1	411	130

where the vertical $\{0, 1\}$ represent the observed value of Y_t and the horizontal $\{0, 1\}$ represent the prediction of the model. Out of the 1257 data points, we have misclassifed $(106 + 411) \,/\, 1257 = 41\%$. However, we have only predicted $Y_t = 1$ (Up) and been wrong $106 \,/\, 1257 = 8\%$.

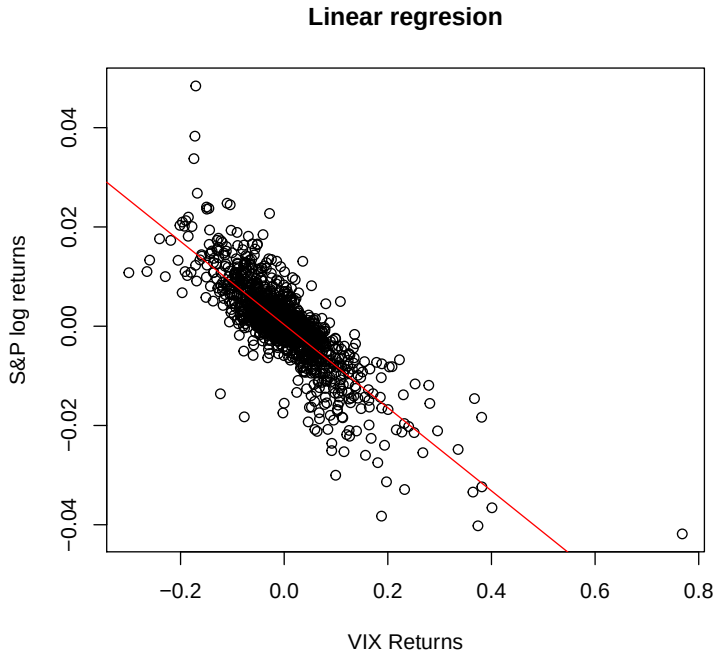
Statistical Inference

In general, the analysis can be divided into the K-nearest neighbor and linear regression parts. This section discusses the results and the according implication of both models. As a reminder, for technical details and a mathematical description of the models please refer to *Technical Notes 2* for K-nearest neighbors and *Technical Note 3* for linear regression.

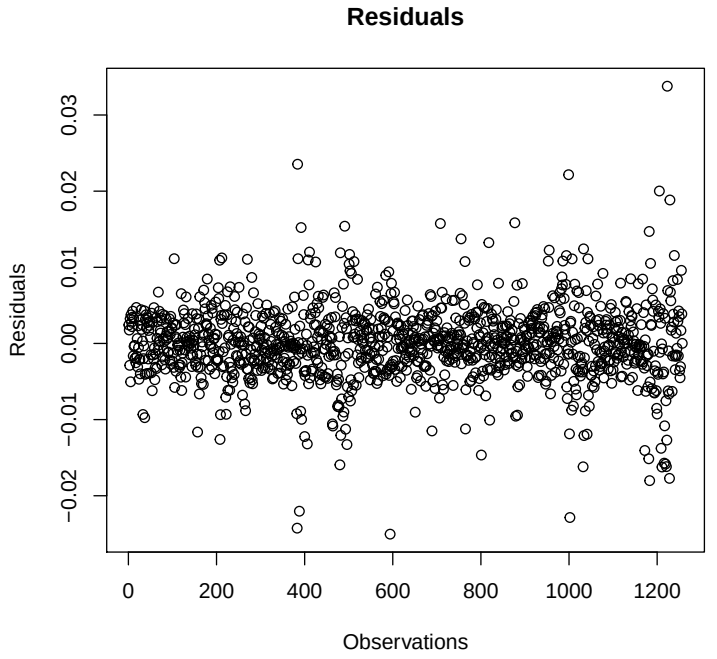
From many different resampling methods which are discussed in the technical notes, the optimal K can be assumed to lie around 60-85 or even around 100 depending on the underlying resampling method. The above plots from section 'Implementation and Sampling Methods' suggest that with a good choice of K the MSE of the validation set is quite low. This means, that the models approximate the existing data and the validation data relatively well. However, the choice of K is crucial for the analysis and from the validation approaches we see that we get a wide range for the optimal K which depends on the validation model that was used. Thus, no perfect value of K can be filtered out and some variability in the models will remain. This variability breaks the predictability of the model. In the implemented portfolio strategy that will follow in a later section, we can still see that we can use the KNN algorithm to create a strategy that seems to be better than guessing as it will be discussed.

Regarding the linear regression it can be concluded that the simplest model is the best fit and seems to be the best model to predict the log return of the S&P. The model considered the daily return of the VIX and it could be concluded that the regression coefficient β_1 was -0.0837. This means that a 1% increase in the VIX return results in a predicted log return for the S&P of -0.0837 %. Given a VIX return one can thus, use this regression to estimate the log return of the S&P. The following graph helps to visualise the regression and serves as an indication for the goodness of fit.

[4] See the adjusted function **KNN.classify** in *Technical Note 4*



One can see that the regression line seems to fit the data well but there still are significant outliers. The R^2 value which indicates how much of the variability is explained by the regression model is around 0.663. For a model with just one explanatory variable this value is quite high but overall there is still much variability that cannot be explained.



Furthermore, the plot of the residuals confirms the above observations since it indicates white noise but again with significant outliers.

Results and Recommendations

The linear regression with just one variable seemed as it is able to make predictions on the S&P given the VIX, however one variable is clearly not enough as the model simply takes averages and thus is too simple. The negative coefficient value indicates that on average, increased volatility implies a decrease in the S&P but this is not sophisticated enough since we still see many days with high volatility but also high returns. Thus, the linear regression model cannot be assumed to have significant prediction power on the S&P and more sophisticated regression models would be necessary.

The preceding analysis and *Figure 8 & Figure 9* on the next page indicate that a KNN classification model does show promise in predicting market movements given the VIX.

Having said that, our dataset will need to be tested over a larger training set, as well as a training set that includes significant shocks (such as the *Covid19* pandemic) to see how the strategy reacts over periods of extended high volatility.

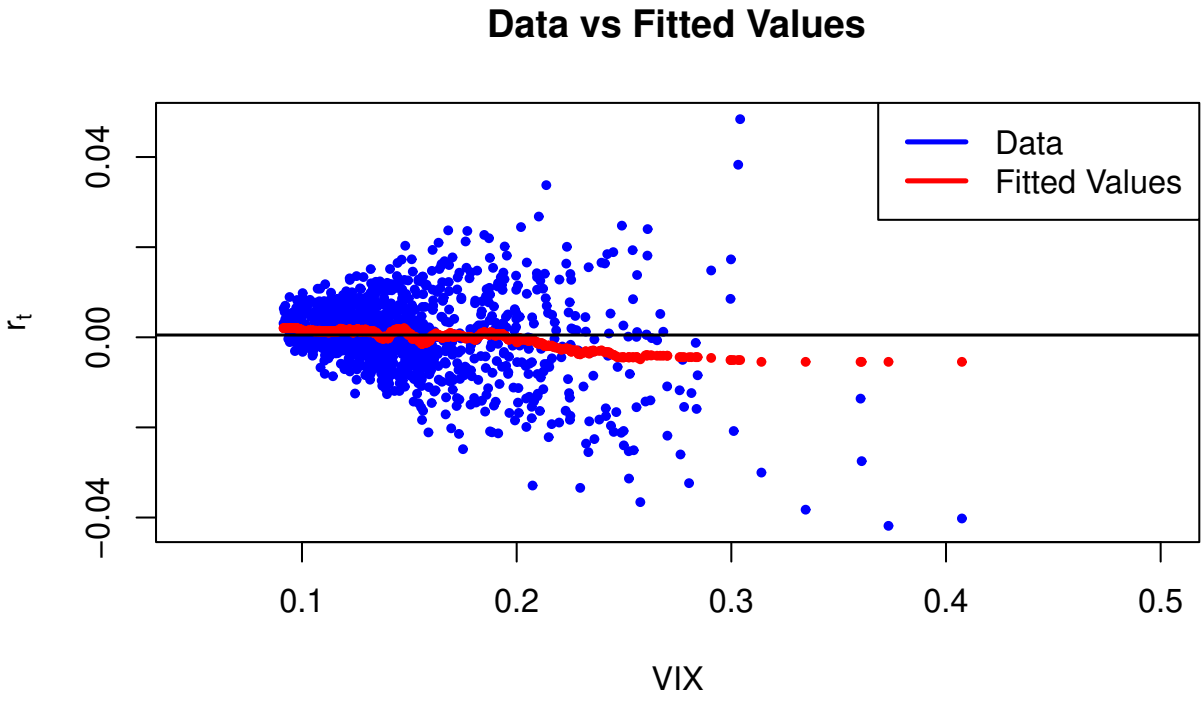


Figure 7: The observed log-return vs. KNN fitted values, given the VIX.

In contrast, the KNN model used to predict log returns (*Figure 7*) is not particularly useful, given the large prediction errors when implied volatility is high. The model is not able to detect the direction of large swings in times of high volatility, and its default is to price high volatility as a negative target (log-return). While this might be true on aggregate, we are clearly missing out on large positive moves.

Therefore, our recommendation would be to develop the KNN classification model further, and perhaps augment it with other classification algorithms, such as logistic regression.

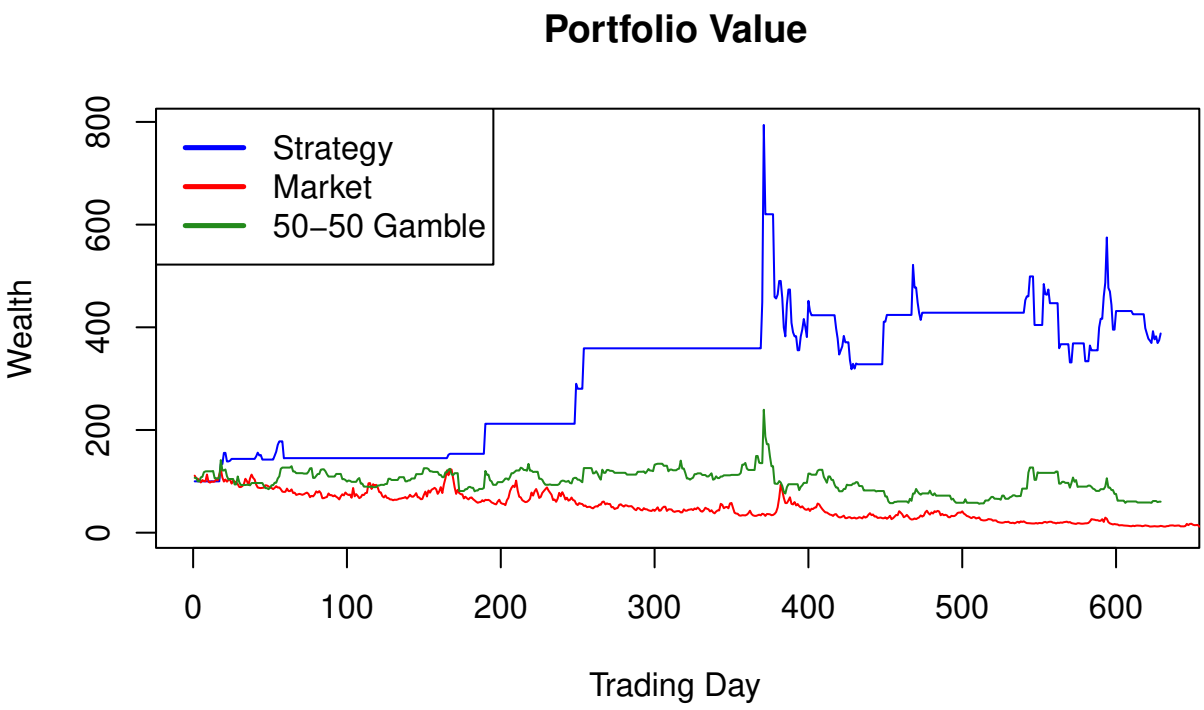


Figure 8: Value of $100 invested at the mid-point of the dataset for three different strategies.

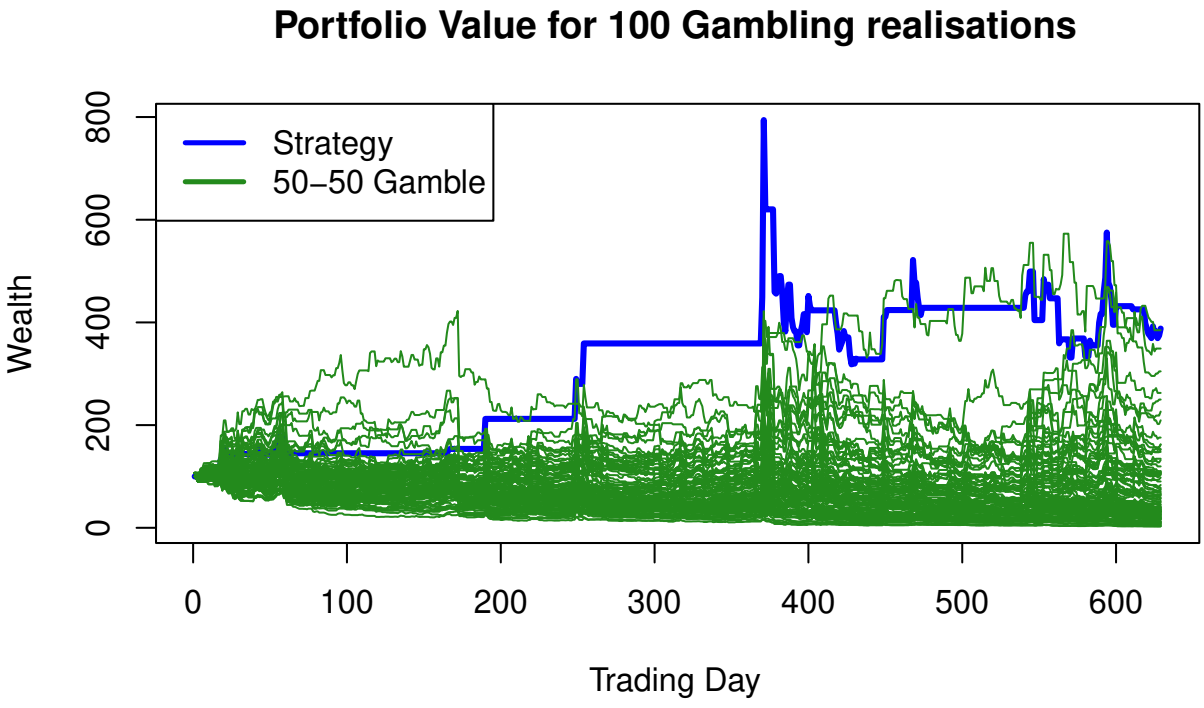


Figure 9: Value of $100 invested at the mid-point of the dataset for 100 random strategies.

Limitations and Extensions

Several modelling assumptions have been made throughout the report, and these assumptions have directly influenced the results and subsequent recommendations made.

Firstly, the cost function we have used for each method has relied on a distance represented by square-loss. This implies that an error that is equidistant of an observation, irrespective of the direction in which it errs, should be treated equally. This symmetry eases the computation of distance, but in financial applications, losses may be asymmetric *e.g.* it is better to allocate surplus capital than insufficient capital and be forced to close a position. Losses need to account for absorbing states, and asymmetries for the fund should be identified.

A major limitation in our analysis is the large degree of choice we have in the value of K. As the analysis has shown, we have an idea of a reasonable interval, although no one value dominates, hence we have had to make a judgement call. Expert opinion may differ, and we recommend technical readers to implement the code themselves and perform additional tests using different values of K.

A related point is that a probability of 0.5 was used to classify points in the KNN algorithm. This is again our choice, and the value used can be tailored for different risk preferences. As we have seen on the full dataset, the model predicted an up-move erroneously only 8% of the time, so we would not expect the value of our portfolio to decrease often. However, one may want to analyse the severity of these misclassifications and investigate methods for understanding why they were misclassified.

In addition, we note that the observations are ordered by time, but we have used validation methods that disregard ordering. In this case, the time component is nullified by viewing pairs of points in isolation, but this would be not be the case when building models that rely on multiple previous values of the VIX. In that case, time-series cross-validation would have to be used.

Finally, there is not necessarily a reason to use the VIX in isolation, and other useful predictors might be identified by experts and included in the model. Similarly, one could investigate the use of the VIX over an extended interval of days when making predictions, and compare that to the predictive power discussed in this report.

Technical Notes

1. Importing the Data

```
## Import the data
Data <- read.table(file="VIXSP500.csv", header = TRUE, sep = ",")
Data$Date <- as.Date(Data$Date, origin = "1899-12-31")
Data$VIX <- Data$VIX / 100

## exploratory plot (cover graphic)
par(mfrow=c(1,1))
plot(Data$Date, Data$Returns, type="l", col="red",
     xlab = "", ylab = "", xaxt='n', yaxt='n')
lines(Data$Date, Data$VIX, type = "l", col = "blue")

## include legends
legend("topleft", c("VIX","S&P 500 Return"),
       lty=c(1,1), lwd=c(2.5,2.5), col=c("blue" ,"red"))

## description of dataframe object
str(Data)
```

2. Model Representation: KNN

The loss function we used for the KNN estimation is the square loss function. Thus, we try to minimise

$$L(\mathcal{Y}, \hat{f}(X)) = (\mathcal{Y} - \hat{f}(X))^2$$

which leads to the result

$$\hat{f}(X) = \mathbb{E}[\mathcal{Y}|X = x].$$

To estimate this conditional expectation, the average of the k nearest neighbors is taken. Since we only have a one dimensional data variable the definition of 'nearest' is straightforward and is computed as

$$d(x, x_i) = \sqrt{(x - x_i)^2}$$

for the value of X that we condition upon and the i'th observation. Note that the square and the square root were added explicitly to emphasize the independence from the sign of the difference.

Since we still need to specify the hyper-parameter k we run the estimation for multiple different values of k and look at the output. In fact, three of the main resampling methods were implemented to find the best suitable k. First, the validation set approach, then the Leave-one-out-cross-validation (LOOCV) and finally the k-fold cross validation. The implications of those approaches are discussed under *Statistical Inference*.

In the validation set approach, the data has been randomly split into a training set and a validation set, where 80% of the data were used as training set and 20% as validation set. The hyper parameter k was then chosen in such a way that

$$k^* = argmin_k MSE_k$$

where

$$MSE_k = \frac{1}{0.2 * \#datapoints} \sum (\hat{y}_i - y_i)^2$$

for the sum over all i's within the validation set. Thus, the validation set approach tries to choose k such that the mean squared error over the validation set is minimised.

Since the validation set approach may be wasting too much data to create the validation set, the LOOCV approach was implemented as well.
In that approach we take the validation set to consist of just one element. We start by taking the first element as the first validation set and walk our way through the whole data set such that in the end all data points have been used as validation set. Thus, the training set is

$$\mathcal{D} \setminus \{(x_j, y_j)\}$$

where j walks from 1 to the number of points in our data set. As in the validation set approach we then take the k that minimises the average over the MSE's.

Finally, we considered the k-fold cross validation with 5 and 10 folds. In that approach the data is divided into k folds, where one part serves as the validation set and the other ones as the training set. As in the previous methods, the k will be chosen such that it minimises the average MSE over all different folds.

As mentioned above, the implications of these methods are discussed under *Statistical Inference* however, the numerical results and short technical implications will also be given here.

With the validation set approach we see a steep decrease of the MSE until a value for k of around 10. Afterwards, there the decrease in MSE is less steep but nonetheless existing. This continues until a value for k of around 50 after which the MSE seems to almost stay constant and after a while slowly starts increasing again. This is obvious since with a k value that is too large, the neighbors will be 'too far away' to precisely predict the relation. Note however, that the outcomes of the validation set approach are not deterministic and will thus have fluctuations. By plotting a histogram, it is still visible that the minimum value for the MSE is reached for a k of around 55 to 65 but also has significant high values around 100.

Since the LOOCV method is deterministic we find a specific value for k which is in our case 85. The pattern we see in the MSE/k plot is very similar to the pattern under the validation set approach.

For the 5-fold and 10-fold cross validation, we again have a stochastic algorithm since the k folds are divided randomly. However, the results from the 5-fold and 10-fold cross validation indicates that the optimal value of k in that model is slightly higher, maybe around 100. Thus, one can see that the optimal value of k seems to be in the area of around 60-85 or even around 100, depending on which model we use to do the validation.

3. Model Representation: Linear Regression

For the linear regression the model

$$Y = \beta_0 + \beta_1 X_1 + ... + \beta_n X_n + \epsilon$$

with $\epsilon \sim \mathrm{N}(0, \sigma^2)$ was assumed. Since we want to predict the log return of the S&P using the value of VIX we define Y as the daily log return and X_1 as the daily VIX return. β_0 denotes as usual the intercept and the β_j's denote the ceteris paribus effects of the according feature variable on the target variable.

More precisely, following models were estimated and compared:

$$\begin{aligned}(I): &\quad Y = \beta_0 + \beta_1 X_1,\\ (II): &\quad Y = \beta_0 + \beta_1 X_2,\\ (III): &\quad Y = \beta_0 + \beta_1 X_3,\end{aligned}$$

as well as the combination of the feature variables:

$$\begin{aligned}(IV): &\quad Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2,\\ (V): &\quad Y = \beta_0 + \beta_1 X_1 + \beta_2 X_3,\\ (VI): &\quad Y = \beta_0 + \beta_1 X_2 + \beta_2 X_3,\end{aligned}$$

as well as

$$(VII): \quad Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3,$$

where Y and X_1 are defined as above and $X_2 := X_1^2,\ X_3 := X_1^3$.

The resulting R^2 and adjusted R^2 values are displayed in the table below.

	R^2	$adj.R^2$
(I)	0.663	0.662
(II)	0.071	0.07
(III)	0.091	0.09
(IV)	0.667	0.667
(V)	0.671	0.67
(VI)	0.091	0.09
(VII)	0.671	0.67

We can see that the values for models $(I), (IV), (V)$ and (VII) are high, namely around 0.6 - 0.7. This are all the models in which X_1 is included. Thus, since the difference between the different R^2 and adjusted R^2 values is marginal, the simplest model can be taken which is model (I). The coefficients of this model are

$$\begin{aligned}\beta_0 &= 0.0003,\\ \beta_1 &= -0.0837.\end{aligned}$$

This means that with 1% increase in the VIX return, we assume the log return of the S&P to decrease by 0.0837%. The according plots and further discussion is conducted in the *Statistical Inference* section.

4. KNN Implementation

```
## Part I: Daily Log-return, given VIX (same day)

## create matrix of target
lr.sp500 <- na.omit(Data$Log.returns)
lr.sp500 <- matrix(lr.sp500, length(lr.sp500), 1)

## create matrix of predictor
vix <- Data$VIX[-1]
vix <- matrix(vix, length(vix), 1)

## knn function under square loss
KNN <- function(X,Y,k,X.hat){
  # X is the feature matrix. Each row contains one observation
  # Y are the target variables
  # X.hat matrix with observations for which we want to determine the yhat value
  n <- dim(X)[1]
  p <- dim(X)[2]

  n.hat <- dim(X.hat)[1]
  y.hat <- rep(0, n.hat)
  for (i in 1:n.hat){
    x <- X.hat[i,]
    DD <- sqrt(rowSums((X-matrix(1,n,1)%*%x)^2)) # distance
    S <- sort(DD, index.return=TRUE)
    I <- S$ix[1:k]
    y.hat[i] <- mean(Y[I])
  }
  return(y.hat)
}

## Run KNN for a variety of k
X <- vix; Y <- lr.sp500;
k.vector <- seq.int(1,150,5)
MSE.train <- rep(0,length(k.vector))
for (j in 1:length(k.vector)){
  Y.pred <- KNN(X,Y,k.vector[j],X)
  MSE.train[j] <- mean((Y-Y.pred)^2)
}

plot(k.vector, MSE.train, type="l", col="blue",
     main = "Training MSE for different choices of K",
     xlab = "K", ylab = "MSE")
```

```
## a) Validation set approach
##    randomly split the data (80 test - 20 validation)
X <- vix; Y <- lr.sp500; split <- 0.8;
I <- sample(1:dim(X)[1], size=floor(split*(dim(X)[1])));
X.train <- matrix(X[I], length(X[I]), dim(X)[2]) ;
Y.train <- matrix(Y[I], length(Y[I]), dim(Y)[2]);
X.val <- matrix(X[-I], length(X[-I]), dim(X)[2]);
Y.val <- matrix(Y[-I], length(Y[-I]), dim(Y)[2]);
k.vector <- seq.int(1, 150, 5)
MSE.val<- rep(0, length(k.vector))
for (j in 1:length(k.vector)){
  Y.pred <- KNN(X.train,Y.train,k.vector[j], X.val)
  MSE.val[j] <- mean((Y.val-Y.pred)^2)
}

plot(k.vector, MSE.val, type="l", col="forestgreen",
     main = "Validation MSE for different choices of K",
     xlab = "K", ylab = "MSE" )
idx <- which.min(MSE.val)
k.vector[idx]

## Choice of k
## BEWARE of running time above 100 simulations
n.sims <- 50
k.vector <- seq.int(1, 200, 10)
choice <- rep(0, n.sims)
for (i in 1:n.sims){
  I <- sample(1:dim(X)[1], size=floor(split*(dim(X)[1])));
  X.train <- matrix(X[I], length(X[I]), dim(X)[2]) ;
  Y.train <- matrix(Y[I], length((Y)[I]), dim(Y)[2]);
  X.val <- matrix(X[-I], length(X[-I]), dim(X)[2]);
  Y.val <- matrix(Y[-I], length(Y[-I]), dim(Y)[2]);
  MSE.val<- rep(0, length(k.vector))
  for (j in 1:length(k.vector)){
    Y.pred <- KNN(X.train,Y.train,k.vector[j], X.val)
    MSE.val[j] <- mean((Y.val-Y.pred)^2)
  }
  idx <- which.min(MSE.val)
  choice[i] <- k.vector[idx]
}

hist(choice, xlab = "Choice of K",
     main = "Choice of K for 1000 Training and Validation Splits")
```

```
## b) LOOCV
k.vector <- seq.int(1, 200, 2)
MSE.vec <- rep(0, length(k.vector))
LOOCV.mse <- rep(0, dim(Y)[1])
for (j in 1:length(k.vector)){
  k <- k.vector[j]
  for (i in 1:dim(Y)[1]){
    X.train <- matrix(X[-i], length(X[-i]), dim(X)[2]) ;
    Y.train <- matrix(Y[-i], length((Y)[-i]), dim(Y)[2]);
    X.val <- matrix(X[i], length(X[i]), dim(X)[2]);
    Y.val <- matrix(Y[i], length(Y[i]), dim(Y)[2]);
    Y.hat <- KNN(X.train,Y.train, k, X.val)
    LOOCV.mse[i] <- (Y.hat-Y.val)^2
  }
  MSE.vec[j] <- mean(LOOCV.mse)
}

plot(k.vector, MSE.vec, type ="l", col="blue", xlab = "K", ylab = "MSE",
     main = "LOOCV MSE for different choices of K")
idx <- which.min(MSE.vec)
k.vector[idx]

## c) 5 & 10 fold CV

## recall
X <- vix; Y <- lr.sp500;

## k-fold cross validation
kfCV=function(X,Y,a){
  # a is the number of folds
  # X, Y matrices
  # Creates matrix Folds' (each column represents 1 fold)
  n <- dim(Y)[1]
  n.fold <- floor(n/a) # truncate to avoid remainder
  Folds <- matrix(1,n.fold,a)
  A <- seq.int(1,n,1)
  A0 <- sample(A, n.fold)
  Folds[,1] <- A0
  for(i in 2:a){
    A1 <- sample(A[-A0], n.fold)
    Folds[,i] <- A1
    A0 <- c(A0,A1)
  }

  KK <- seq(5,200,10) # number of neighbors of the KNN

  ## function definition continued on next page
```

```
  MSE <- rep(0, length(KK)) # MSE for each k
  for(l in 1:length(KK)){
    k <- KK[l]
    MSE.Fold <- rep(0,a) # MSE for each fold as validation set
    for(j in 1:a){
      Train.X <- as.matrix(X[-Folds[,j],]) # not fold j is training set
      Train.Y <- as.matrix(Y[-Folds[,j]])
      Test.X <- as.matrix(X[Folds[,j],])  # Fold j is the validation set
      Test.Y <- as.matrix(Y[Folds[,j]])
      y.hat <- KNN(Train.X, Train.Y, k, Test.X)
      MSE.Fold[j] <- sum((y.hat-Test.Y)^2)
    }
    MSE[l] <- mean(MSE.Fold)
  }
  opt.k <- sort(MSE, index.return=TRUE)$ix[1]
  opt.MSE <- sort(MSE, index.return=TRUE)$x[1]
  return(data.frame(K=KK, MSE=MSE))
}

## fit the models
Model.5fold <- kfCV(X,Y,5)
Model.10fold <- kfCV(X,Y,10)

## plot the results
par(mfrow=c(1,2))

plot(Model.5fold$K, Model.5fold$MSE, xlab="Number of neighbours",
     ylab="Validation error", main = "5-fold cross validation",
     pch=20, col="blue")

plot(Model.10fold$K, Model.10fold$MSE, xlab="Number of neighbours",
     ylab="Validation error", main = "10-fold cross validation",
     pch=20, col="blue")
```

```
## Part II: Up-Down Classification, given VIX (same day)

## clear working environment
rm(list=ls())

## reimport the data
Data <- read.table(file="VIXSP500.csv", header = TRUE, sep = ",")
Data$Date <- as.Date(Data$Date, origin = "1899-12-31")
Data$VIX <- Data$VIX / 100

## create vector of target
returns <- na.omit(Data$Returns)
class.returns <- as.numeric(returns>0.005)

## create vector of predictor
vix <- Data$VIX[-1]

## KNN written for matrix arguments
X <- as.matrix(vix); Y <- as.matrix(class.returns)

## write a new function for KNN
KNN.classify <- function(Training, Y, k, Test){
  min.x=function(y){
    return((y-x)^2)
  }
  y_hat <- rep(0, dim(Test)[1])
  for (i in 1:dim(Test)[1]){
    x <- as.numeric(Test[i,])
    if(dim(Test)[2]==1){DD=sqrt((as.numeric(Training)-x)^2)}
    else{
      DD=sqrt(rowSums(t(apply(Training,1,min.x)))) # distance
    }
    S=sort(DD, index.return=TRUE)
    I=S$ix[1:k]
    y_hat[i]=1*(mean(Y[I])>0.5)
  }
  return(y_hat)
}

Y.hat <- KNN.classify(X,Y,125,X)

## misclassification error
MisClassification <- length(Y)-sum((Y-Y.hat)==0)
MisClassification
MisClassification/length(Y)
table(Y,Y.hat)
```

```
## k-fold cross validation
kfCV=function(X,Y,a){
  # a is the number of folds
  # X, Y matrices
  # Creates matrix Folds' (each column represents 1 fold)
  min.x=function(y){
    return((y-x)^2)
  }
  n=length(Y)
  n.fold=floor(n/a)
  Folds=matrix(1,n.fold,a)
  A=seq(1,n,1)
  A0=sample(A, n.fold)
  Folds[,1]=A0
  for(i in 2:a){
    A1=sample(A[-A0], n.fold)
    Folds[,i]=A1
    A0=c(A0,A1)
  }

  KK=seq.int(5,450,6) # number of neighbors of the KNN

  MSE <- rep(0,length(KK)) # MSE for each k
  for(l in 1:length(KK)){
    k=KK[l]
    MSE.Fold <- rep(0,a)  #MSE for each fold as validation set
    for(j in 1:a){
      Train.X <- as.matrix(X[-Folds[,j],]) # everything except fold j
      Train.Y <- as.matrix(Y[-Folds[,j]])
      Test.X <- as.matrix(X[Folds[,j],])  # Fold j is the validation set
      Test.Y <- as.matrix(Y[Folds[,j]])
      y.hat <- KNN.classify(Train.X, Train.Y, k, Test.X)
      MSE.Fold[j] <- sum(y.hat != as.numeric(Test.Y))
    }
    MSE[l]=mean(MSE.Fold)
  }
  opt.k=sort(MSE, index.return=TRUE)$ix[1]
  opt.MSE=sort(MSE, index.return=TRUE)$x[1]
  return(data.frame(K=KK, MSE=MSE))
}

## fitting the models
Model.5fold <- kfCV(X,Y,5)
Model.10fold <- kfCV(X,Y,10)
```

```
## plotting the results
par(mfrow=c(1,2))

plot(Model.5fold$K, Model.5fold$MSE, xlab="Number of neighbours",
     ylab="Validation error", main = "5-fold cross validation",
     pch=20, col="blue")

plot(Model.10fold$K, Model.10fold$MSE, xlab="Number of neighbours",
     ylab="Validation error", main = "10-fold cross validation",
     pch=20, col="blue")

## identify optimal k
idx <- which.min(Model.5fold$MSE)
Model.5fold$K[idx]

idx <- which.min(Model.10fold$MSE)
Model.10fold$K[idx]
```

5. Linear Model Implementation

```
## Create squared and to the power of three values
Data$Squared = Data$Returns**2
Data$Three = Data$Returns**3

#Estimate regression models
lin_reg1 <- lm(Data$Log.returns~Data$Returns)
lin_reg2 <- lm(Data$Log.returns~Data$Squared)
lin_reg3 <- lm(Data$Log.returns~Data$Three)
lin_reg4 <- lm(Data$Log.returns~Data$Returns + Data$Squared)
lin_reg5 <- lm(Data$Log.returns~Data$Returns + Data$Three)
lin_reg6 <- lm(Data$Log.returns~Data$Squared + Data$Three)
lin_reg7 <- lm(Data$Log.returns~Data$Returns + Data$Squared + Data$Three)

cat("Model 1: R-squared:", summary(lin_reg1)$r.squared, "and adjusted R-squared:",
summary(lin_reg1)$adj.r.squared)
cat("Model 2: R-squared:", summary(lin_reg2)$r.squared, "and adjusted R-squared:",
summary(lin_reg2)$adj.r.squared)
cat("Model 3: R-squared:", summary(lin_reg3)$r.squared, "and adjusted R-squared:",
summary(lin_reg3)$adj.r.squared)
cat("Model 4: R-squared:", summary(lin_reg4)$r.squared, "and adjusted R-squared:",
summary(lin_reg4)$adj.r.squared)
cat("Model 5: R-squared:", summary(lin_reg5)$r.squared, "and adjusted R-squared:",
summary(lin_reg5)$adj.r.squared)
cat("Model 6: R-squared:", summary(lin_reg6)$r.squared, "and adjusted R-squared:",
summary(lin_reg6)$adj.r.squared)
cat("Model 7: R-squared:", summary(lin_reg7)$r.squared, "and adjusted R-squared:",
summary(lin_reg7)$adj.r.squared)
```

```
## Plot the Data with the regression
plot(Data$Returns, Data$Log.returns, main="Linear regresion", xlab="VIX Returns",
ylab="S&P log returns")
abline(lin_reg1, col='red')

## Plot residuals
plot(resid(lin_reg1), main="Residuals", xlab="Observations", ylab="Residuals")
```

6. Trading Strategy

```
## Movement Classifier
k <- 120
n.obs <- dim(Y)[1]

f.split <- floor(0.5*n.obs)
c.split <- ceiling(0.5*n.obs)

Wealth <- rep(0, c.split)

for (i in 1:c.split){
  X.train <- as.matrix(X[1:(f.split+i-1),])
  Y.train <- as.matrix(Y[1:(f.split+i-1),])
  y.hat <- KNN.classify(X.train, Y.train, 120, as.matrix(X[(f.split+i),]))
  if (i==1){
    if (y.hat==1){
      Wealth[i] <- 100*(1+returns[c.split])
    } else {
      Wealth[i] <- 100
    }
  } else {
      if (y.hat==1){
        Wealth[i] <- Wealth[i-1]*(1+returns[f.split+i])
      } else {
        Wealth[i] <- Wealth[i-1]
      }
  }
}
```

```
## market portfolio
Market <- rep(0, dim(X)[1])
Market[1] <- 100*(1+returns[1])
for (i in 2:length(Market)){
  Market[i] <- Market[i-1]*(1+returns[i])
}

## plot strategy vs. market
par(mfrow=c(1,1))
plot(Wealth, type ="l", col="blue", xlab ="Trading Day",
     ylim=c(min(Market), max(Wealth)), main = "Portfolio Value")
lines(Market, col="red")
lines(Gamble,col="forestgreen")

## include legends
legend("topleft", c("Strategy","Market", "50-50 Gamble"),
       lty=c(1,1), lwd=c(2.5,2.5), col=c("blue" ,"red", "forestgreen"))


## one gamble
Gamble <- rep(0, c.split)
n <- length(Gamble)
guess <- sample(c(0,1), n, replace = TRUE, prob = c(.5, .5))

for (i in 1:n){
  y.hat <- guess[i]
  if (i==1){
    if (y.hat==1){
      Gamble[i] <- 100*(1+returns[c.split])
    } else {
      Gamble[i] <- 100
    }
  } else {
    if (y.hat==1){
      Gamble[i] <- Gamble[i-1]*(1+returns[f.split+i])
    } else {
      Gamble[i] <- Gamble[i-1]
    }
  }
}
```

```
## m gambles
m <- 100
Paths <- matrix(0, nrow = length(Gamble), ncol = m)
for (s in 1:m){
  Gamble <- rep(0, c.split)
  n <- length(Gamble)
  guess <- sample(c(0,1), n, replace = TRUE, prob = c(.5, .5))
  for (i in 1:n){
    y.hat <- guess[i]
    if (i==1){
      if (y.hat==1){
        Gamble[i] <- 100*(1+returns[c.split])
      } else {
        Gamble[i] <- 100
      }
    } else {
      if (y.hat==1){
        Gamble[i] <- Gamble[i-1]*(1+returns[f.split+i])
      } else {
        Gamble[i] <- Gamble[i-1]
      }
    }
  }
  Paths[,s] <- Gamble
}

## plot the m sample paths
## BEWARE running time for plotting m > 100
plot(Wealth, type ="l", col="blue", xlab ="Trading Day", lwd=3,
     ylim=c(min(Market), max(Paths, Wealth)),
     main="Portfolio Value for 100 Gambling realisations")
for (i in 1:dim(Paths)[2]){
  lines(Paths[,i], col="forestgreen", lwd=1)
}

## include legends
legend("topleft", c("Strategy", "50-50 Gamble"),
       lty=c(1,1), lwd=c(2.5,2.5), col=c("blue" ,"forestgreen"))
```