Lab 4

Yuhao Wang and yw3204

October 23, 2018

Instructions

Make sure that you upload a knitted pdf or html file to the canvas page (this should have a .pdf or .html extension). Also upload the .Rmd file. Include output for each question in its own individual code chunk and don't print out any vector that has more than 20 elements.

Objectives: KNN Classification and Cross-Validation

Background

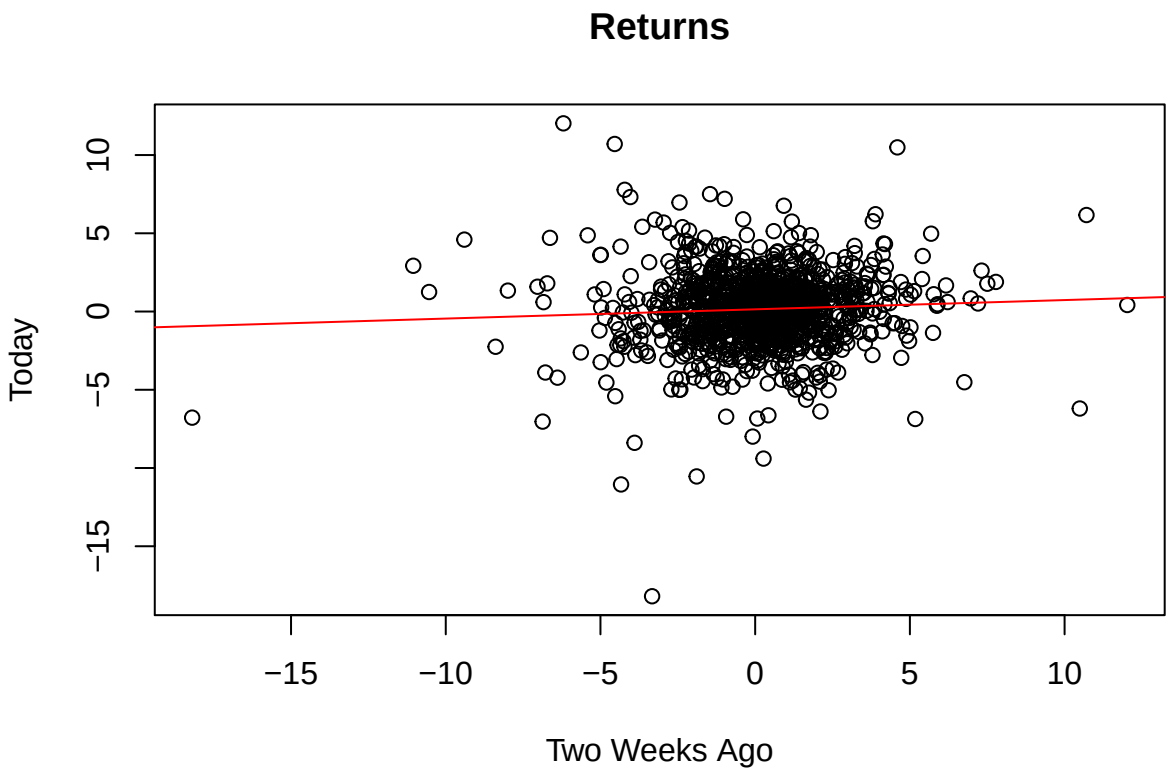
Today we'll be using the *Weekly* dataset from the *ISLR* package. This data is similar to the *Smarket* data from class. The dataset contains 1089 weekly returns from the beginning of 1990 to the end of 2010. Make sure that you have the *ISLR* package installed and loaded by running (without the code commented out) the following:

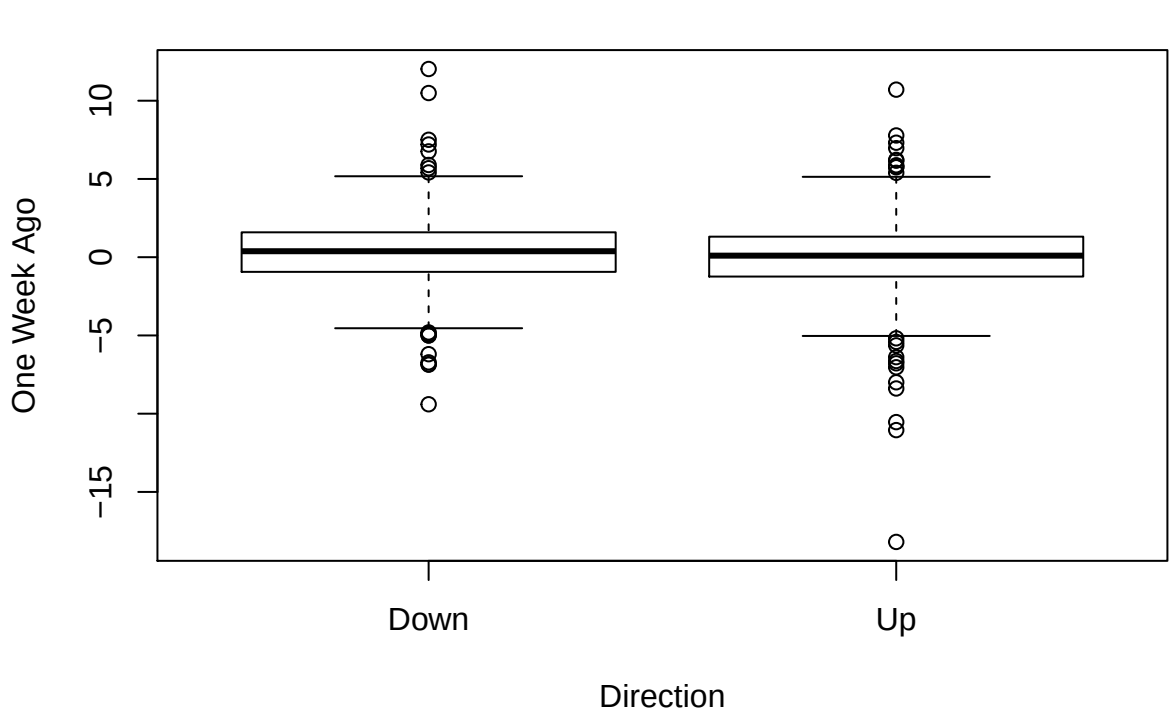
```
# install.packages("ISLR")
library(ISLR)
```

We'd like to see if we can accurately predict the direction of a week's return based on the returns over the last five weeks. *Today* gives the percentage return for the week considered and *Year* provides the year that the observation was recorded. *Lag1* - *Lag5* give the percentage return for 1 - 5 weeks previous and *Direction* is a factor variable indicating the direction ('UP' or 'DOWN') of the return for the week considered.

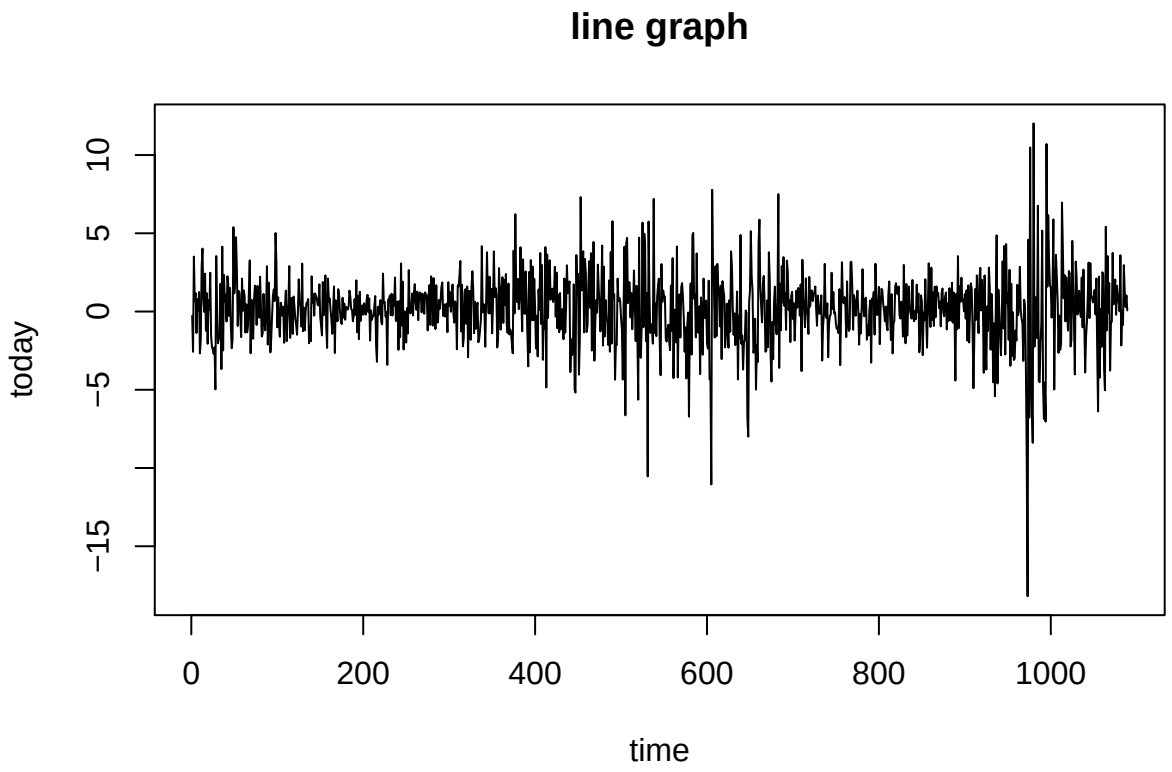
Part 1: Visualizing the relationship between this week's returns and the previous week's returns.

1. Explore the relationship between a week's return and the previous week's return. You should plot more graphs for yourself, but include in the lab write-up a scatterplot of the returns for the weeks considered (*Today*) vs the return from two weeks previous (*Lag2*), and side-by-side boxplots for the lag one week previous (*Lag1*) divided by the direction of this week's Reuther (*Direction*).






```
# EDA of weekly data
# time series scatter plot
#plot(Weekly$Today, xlab = "time", ylab = "today", main = "scatter plot")
# line graph
plot(Weekly$Today, xlab = "time", ylab = "today", type = "l", main = "line graph")
```




```
# correlation matrix among today and lag value
cor_mat <- cor(Weekly[, c(2, 3, 4, 5, 6, 8)])
#library(corrplot)
#corrplot(cor_mat, type = "upper", tl.col = "black", tl.srt = 45)
```

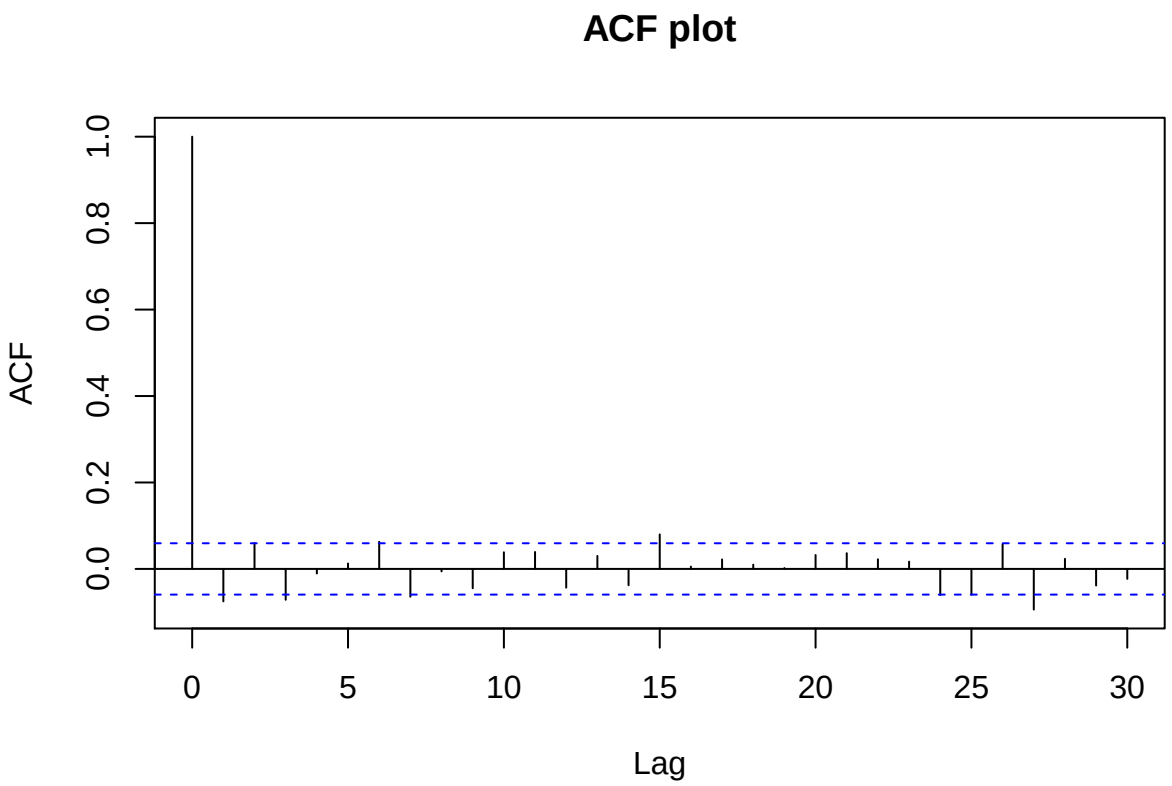
As we can see in the result above, there is little correlationship among today and lag value.

```
# time series analysis
library(tseries)
adf.test(Weekly$Today) #check the stationarity
```

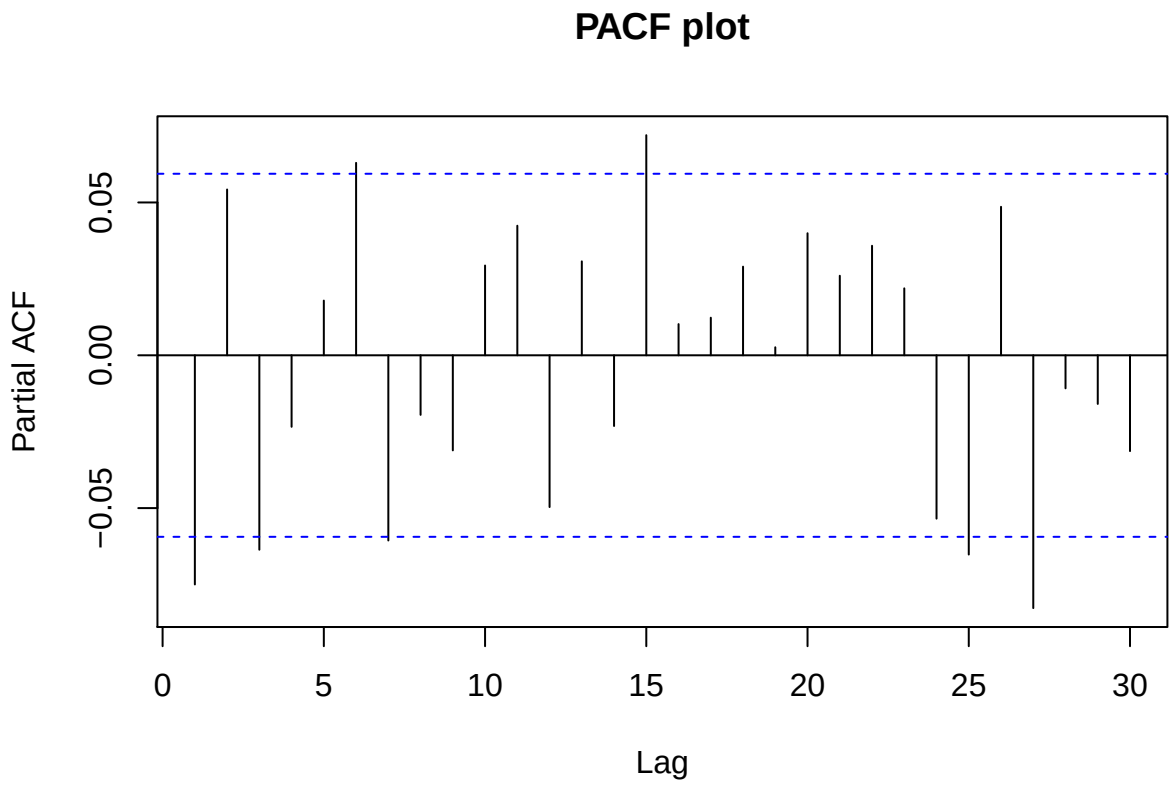
```
## Warning in adf.test(Weekly$Today): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  Weekly$Today
## Dickey-Fuller = -9.787, Lag order = 10, p-value = 0.01
## alternative hypothesis: stationary
```

```
acf(Weekly$Today, main = "ACF plot")
```




```
pacf(Weekly$Today, main = "PACF plot")
```



Part 2: Building a classifier

Recall the KNN procedure. We classify a new point with the following steps:

– Calculate the Euclidean distance between the new point and all other points.

– Create the set $\mathcal{N}_{new}$ containing the K closest points (or, nearest neighbors) to the new point.

– Determine the number of 'UPs' and 'DOWNs' in $\mathcal{N}_{new}$ and classify the new point according to the most frequent.

2. We'd like to perform KNN on the *Weekly* data, as we did with the *Smarket* data in class. In class we wrote the following function which takes as input a new point $(Lag1_{new}, Lag2_{new})$ and provides the KNN decision using as defaults $K = 5$, Lag1 data given in *Smarket$Lag1*, and Lag2 data given in *Smarket$Lag2*. Update the function to calculate the KNN decision for weekly market direction using the *Weekly* dataset with *Lag1* - *Lag5* as predictors. Your function should have only three input values: (1) a new point which should be a vector of length 5, (2) a value for K, and (3) the Lag data which should be a data frame with five columns (and n rows).

```
# update function
KNN.decision <- function(new_data, K, df_lag) {

   stopifnot(length(new_data) == 5, K <= nrow(df_lag))

   #these have really bad performance
   #dists <- apply(df_lag, 1, '-', as.numeric(new_data))
   #dists <- apply(df_lag, 1, function(x) x-as.numeric(new_data))
   #dists <- sapply(dists, function(x) sqrt(sum(x^2)))

   dists <- sweep(df_lag, 2, as.numeric(new_data))
   dists <- apply(dists, 1, function(x) sqrt(sum(x^2)))
   neighbors  <- order(dists)[1:K]
   neighb.dir <- Weekly$Direction[neighbors]
   choice <- names(which.max(table(neighb.dir)))
   return(choice)
}
```

3. Now train your model using data from 1990 - 2008 and use the data from 2009-2010 as test data. To do this, divide the data into two data frames, *test* and *train*. Then write a loop that iterates over the test points in the test dataset calculating a prediction for each based on the training data with $K = 5$. Save these predictions in a vector. Finally, calculate your test error, which you should store as a variable named *test.error*. The test error calculates the proportion of your predictions which are incorrect (don't match the actual directions).

```
train <- Weekly[Weekly$Year <= 2008, ]
test <- Weekly[!Weekly$Year <= 2008, ]

n_test <- nrow(test)

pred <- c()
for(i in c(1:n_test)) {
  pred <- c(pred, KNN.decision(test[i, 2:6], 5, train[, 2:6]))
}

test.error <- sum(pred != test$Direction) / n_test
test.error
```

```
## [1] 0.4519231
```

4. Do the same thing as in question 3, but instead use $K = 3$. Which has a lower test error?

```
pred1 <- c()

for(i in c(1:n_test)) {
  pred1 <- c(pred1, KNN.decision(test[i, 2:6], 3, train[, 2:6]))
}

test.error1 <- sum(pred1 != test$Direction) / n_test
test.error1
```

```
## [1] 0.4423077
```

The model with K = 3 has a lower test error.

Part 3: Cross-validation

Ideally we'd like to use our model to predict future returns, but how do we know which value of K to choose? We could choose the best value of K by training with data from 1990 - 2008, testing with the 2009 - 2010 data, and selecting the model with the lowest test error as in the previous section. However, in order to build the best model, we'd like to use ALL the data we have to train the model. In this case, we could use all of the *Weekly* data and choose the best model by comparing the training error, but unfortunately this isn't usually a good predictor of the test error.

In this section, we instead consider a class of methods that estimate the test error rate by holding out a (random) subset of the data to use as a test set, which is called k-fold cross validation. (Note this lower case k is different than the upper case K in KNN. They have nothing to do with each other, it just happens that the standard is to use the same letter in both.) This approach involves randomly dividing the set of observations into k groups, or folds, of equal size. The first fold is treated as a test set, and the model is fit on the remaining $k - 1$ folds. The error rate, ERR1, is then computed on the observations in the held-out fold. This procedure is repeated k times; each time, a different group of observations is treated as a test set. This process results in k estimates of the test error: ERR1, ERR2, ..., ERRk. The k-fold CV estimate of the test error is computed by averaging these values,

$$CV_{(k)} = \frac{1}{k} \sum_{i=1}^{k} ERR_k.$$

We'll run a 9-fold cross-validation in the following. Note that we have 1089 rows in the dataset, so each fold will have exactly 121 members.

5. Create a vector *fold* which has n elements, where n is the number of rows in *Weekly*. We'd like for the *fold* vector to take values in 1-9 which assign each corresponding row of the *Weekly* dataset to a fold. Do this in two steps: (1) create a vector using *rep()* with the values 1-9 each repeated 121 times (note $1089 = 121 \cdot 9$), and (2) use *sample()* to randomly reorder the vector you created in (1).

```
fold <- rep(1:9, 121)
fold <- sample(fold, nrow(Weekly))
```

6. Iterate over the 9 folds, treating a different fold as the test set and all others the training set in each iteration. Using a KNN classifier with $K = 5$ calculate the test error for each fold. Then calculate the cross-validation approximation to the test error which is the average of ERR1, ERR2, ..., ERR9.

```
err <- c()
for(i in c(1:9)) {
```

```
  train_i <- Weekly[fold != i, ]
  test_i <- Weekly[fold == i, ]

  pred_tmp <- c()
  for(j in c(1:nrow(test_i))) {
    pred_tmp <- c(pred_tmp, KNN.decision(test_i[j, 2:6], 5, train_i[, 2:6]))
  }

  err <- c(err, sum(pred_tmp != test_i$Direction) / nrow(test_i))
}

cv_5 <- mean(err)
cv_5
```

```
## [1] 0.4701561
```

7. Repeat step (6) for $K = 1$, $K = 3$, and $K = 7$. For which set is the cross-validation approximation to the test error the lowest?

```
cv <- c()

for(k in c(1, 3, 7)) {
  err <- c()

  for(i in c(1:9)) {
    train_i <- Weekly[fold != i, ]
    test_i <- Weekly[fold == i, ]

    pred_tmp <- c()
    for(j in c(1:nrow(test_i))) {
      pred_tmp <- c(pred_tmp, KNN.decision(test_i[j, 2:6], k, train_i[, 2:6]))
    }

    err <- c(err, sum(pred_tmp != test_i$Direction) / nrow(test_i))
  }

  cv <- c(cv, mean(err))
}

cv
```

```
## [1] 0.4848485 0.4719927 0.4784206
```

Comparatively speaking, when $K = 5$, we get the smallest CV error.