Machine Learning

Winter 2020

HW #1

Jiyoon Chung

Arman Bhuiyan

Hikaru Sugimori

Deepak Putchakayala

Question 1

For each one the ten statements below say whether they are true or not and explain why. 1. As one increases k, the number of nearest neighbor, in a kNN classifier,

1

(a) the bias of the classifier will increase;

True, the bias of the classifier will increase because the kNN classifier will be a simple looking function with low model complexity.

(b) the variance of the classifier will increase;

False, the larger the k is, the less flexible model is (has less degrees of freedom). These estimators have low variance.

(c) the misclassification rate on the training dataset will increase; (d) the misclassification rate on a test dataset will increase.

C = True, the misclassification rate on the training dataset will increase because lower k has a better fit within the training dataset, but the misclassification rate on a test dataset will decrease because higher k results in a formula that is not chasing as much noise in the training dataset, so it does better out of sample. D = False.

2. Consider the three line regression fits to the gray points plotted below.

(a) The estimate in (2) has a higher variance than the estimate in (1).

True. 2 has a higher k than 1. Higher k implies higher bias and lower variance.

(b) The estimate in (2) has a higher bias than the estimate in (3).

True. 2 has a higher k than 3 and thus the former has more flexibility/variance.

(c) The estimate in (3) has the smallest training error.

True. 3 with low k is overfitting. Typical overfitting means that error on the training data is very low, but error on new instances is high.

(d) The estimate in (1) has the smallest test error.

False. The estimate in 2 will have the smallest test error because it has the best fit without chasing the noise of the test dataset. 2 is the ideal function as it is not too complex but not too simple, this cannot be said of 1.

Misclassification rate of a classifier evaluated on a validation set will never be smaller than the one evaluated on the training set that is used to build the classifier.

False, most likely this will be true but it is incorrect to say NEVER. Training error will almost always **underestimate** your validation error. However it is possible for the validation error to be less than the training. This can happen if your training set had many 'hard' cases to learn or your validation set had mostly 'easy' cases to predict

k-fold cross-validation provides an unbiased estimate of the predictive error of the models.

False, there exists no universal (valid under all distributions) unbiased estimator of the variance of K-fold cross-validation. It is true that k-fold cross-validation provides a less biased or less optimistic estimate of the model skill than other methods, but bias cannot be completely eliminated. k = 5 or k = 10 are the industry standard, as these values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance.

Q2:

Q2 – 1:

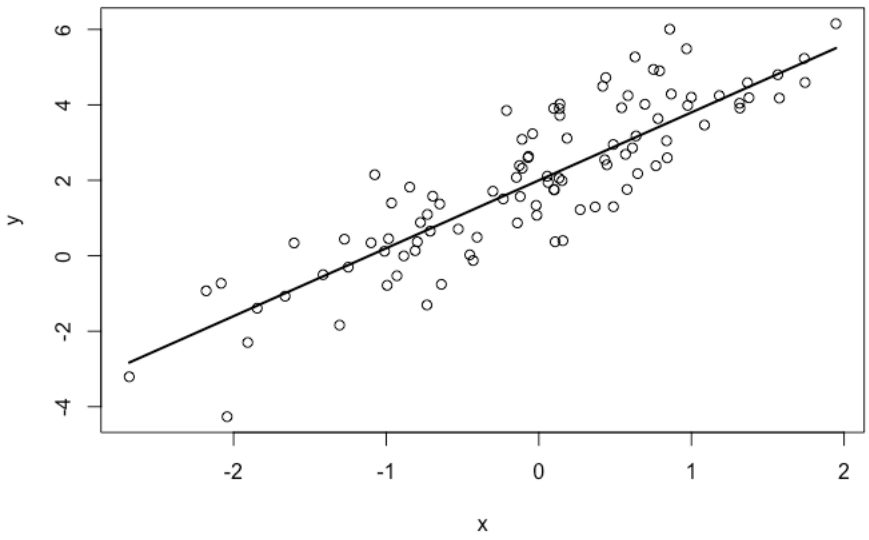
```
PackageList =c('MASS','data.table','tree','kknn','rpart','rpart.plot')
NewPackages=PackageList[!(PackageList %in%
                                        installed.packages()[,"Package"])]
if(length(NewPackages)) install.packages(NewPackages)
lapply(PackageList,require,character.only=TRUE)
```

```
set.seed(202001)
x <- rnorm(100, mean=0, sd=1)
epsilon <- rnorm(100, mean=0, sd=1)
```

```
y <- 1.8*x + 2 + epsilon
trainingdata <- data.frame(y,x)
trainingdata <- trainingdata[order(trainingdata$x),]

x <- rnorm(10000, mean=0, sd=1)
epsilon <- rnorm(10000, mean=0, sd=1)
y <- 1.8*x + 2 + epsilon
testdata <- data.frame(y, x)
testdata <- testdata[order(testdata$x),]
rm(x,y)
```

Q2 – 2:

```
plot(trainingdata$x, trainingdata$y, xlab = "x", ylab = "y")+
   lines(trainingdata$x, 1.8*trainingdata$x + 2, col="black", lwd = 2)
```

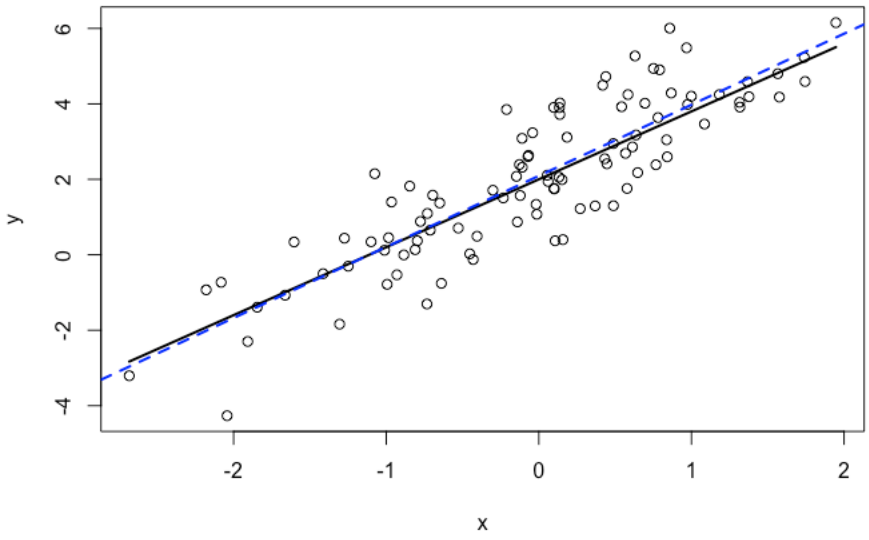


Q2 – 3:

```
ls <- lm(trainingdata$y~trainingdata$x, trainingdata)
print(summary(ls))
plot(trainingdata$x, trainingdata$y, xlab = "x", ylab = "y")+
```

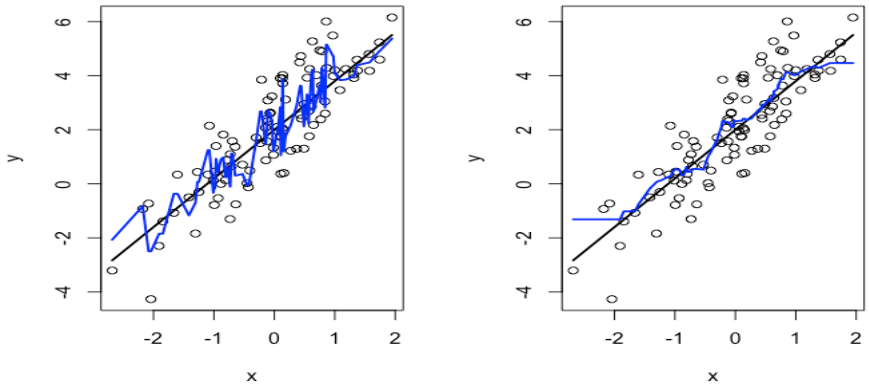
```
  lines(trainingdata$x, 1.8*trainingdata$x + 2, col="black", lwd = 2)+
  abline(ls$coef, col="blue", lwd = 2, lty = "dashed")
```

```
Call:
lm(formula = trainingdata$y ~ trainingdata$x, data = trainingdata)

Residuals:
      Min        1Q    Median        3Q       Max
-2.51013 -0.66251 -0.08314  0.67281  2.29903

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)       2.0920     0.1045   20.02   <2e-16 ***
trainingdata$x    1.8839     0.1089   17.30   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.044 on 98 degrees of freedom
Multiple R-squared:  0.7532,    Adjusted R-squared:  0.7507
F-statistic: 299.1 on 1 and 98 DF,  p-value: < 2.2e-16
```



Q2 – 4:

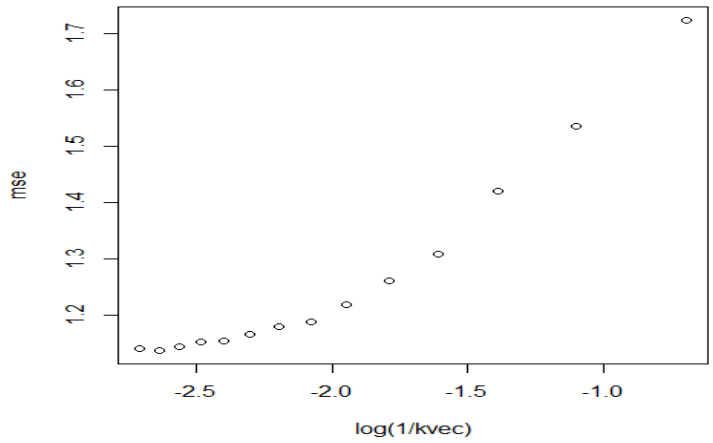
```
library(kknn)
kvec <- c(2:15)
kknn.train <- list()
fitted.train <- data.frame(matrix(NA, nrow = 100, ncol = length(kvec)))
for (i in 1:length(kvec)) {
   kknn.train[[i]] <- kknn(y~x, train, train, k = kvec[i], kernel = "rectangular")
   fitted.train[, i] <- kknn.train[[i]]$fitted
}

par(mfrow=c(1,2))
plot(trainingdata$x, trainingdata$y, xlab = "x", ylab = "y")+
   lines(trainingdata$x, 1.8*trainingdata$x + 2, col="black", lwd = 2)+
   lines(train$x, fitted.train[, 1], col="blue", lwd = 2)

plot(trainingdata$x, trainingdata$y, xlab = "x", ylab = "y")+
   lines(trainingdata$x, 1.8*trainingdata$x + 2, col="black", lwd = 2)+
   lines(train$x, fitted.train[, 11], col="blue", lwd = 2)
```



Q5

```
kvec <- c(2:15)
kknn.test <- list()
fitted.test <- data.frame(matrix(NA, nrow = 10000, ncol = length(kvec)))
mse <- as.numeric()
for(i in 1:length(kvec)) {
```

```
    kknn.test[[i]] <- kknn(y~x, train, test, k = kvec[i], kernel = "rectangular")
    fitted.test[, i] <- kknn.test[[i]]$fitted
    mse[i] = mean((test$y-fitted.test[,i])^2)
}
mse.ls <- mean((test$y-ls$coefficients[1]-ls$coefficients[2]*test$x)^2)
plot(log(1/kvec), mse)+
    abline(h = mse.ls, lwd=2, col = "black", lty = 3)
```



Minimum MSE

1. KNN – 1.261612
2. Linear Regression – 1.0248

Linear regression produces better model that produces lower MSE than KNN does

Q6

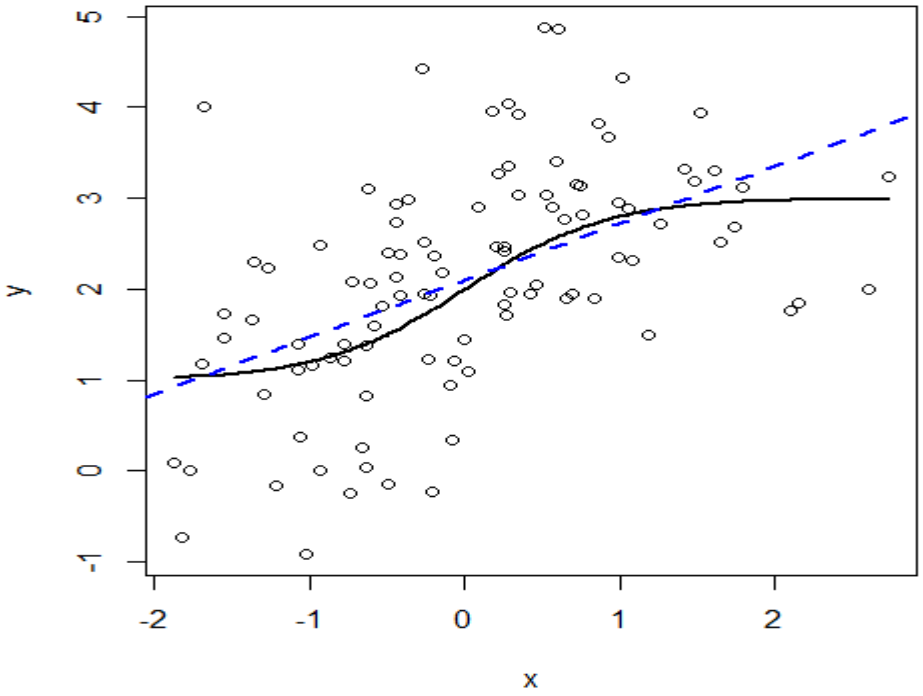
lm(formula = trainData$y ~ trainData$x, data = trainData)

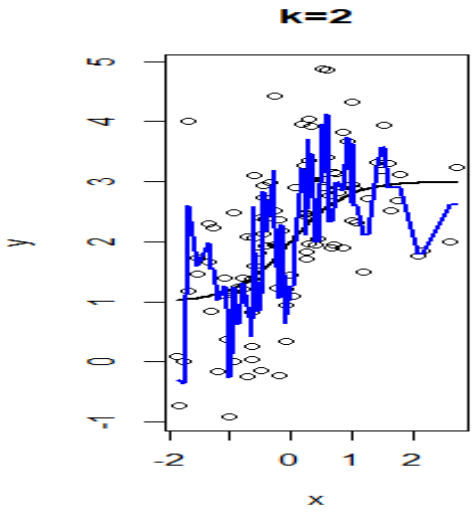
Residuals:

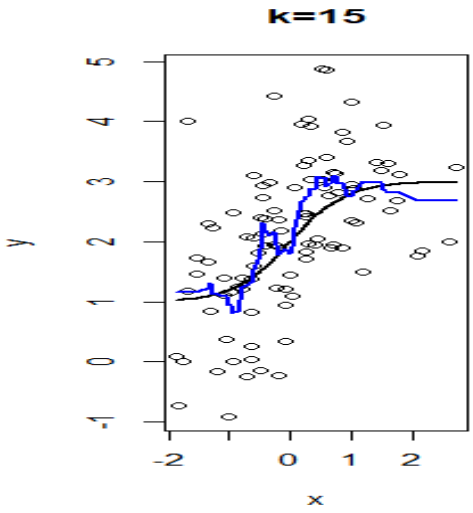
Min	1Q	Median	3Q	Max
-2.36223	-0.60754	0.07714	0.60121	2.95953

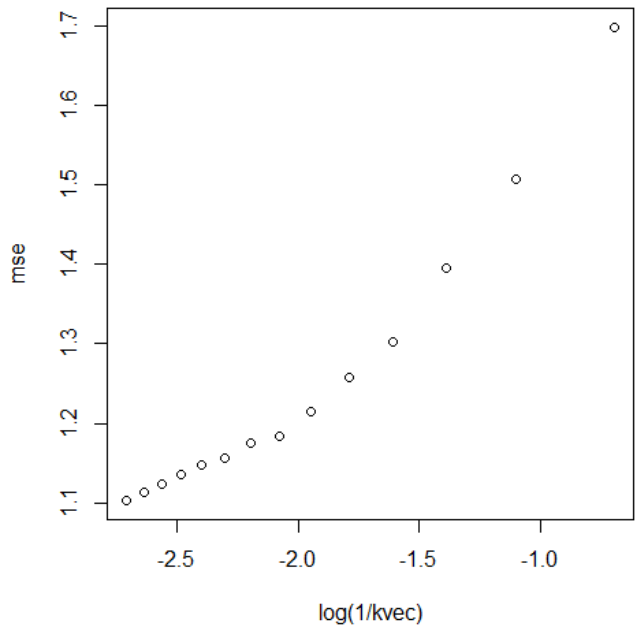
Coefficients:

```
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     2.0999     0.1070   19.63   < 2e-16 ***
trainData$x     0.6266     0.1060    5.91 4.98e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.069 on 98 degrees of freedom
Multiple R-squared:  0.2628,  Adjusted R-squared:  0.2552
F-statistic: 34.93 on 1 and 98 DF,  p-value: 4.979e-08
```









Minimum MSE

1. KNN – 1.10342
2. Linear regression – 1.058055

Linear regression produces better model that produces lower MSE than KNN does

Q7

lm(formula = trainData$y ~ trainData$x, data = trainData)

Residuals:

Min	1Q	Median	3Q	Max
-2.36223	-0.60754	0.07714	0.60121	2.95953

Coefficients:

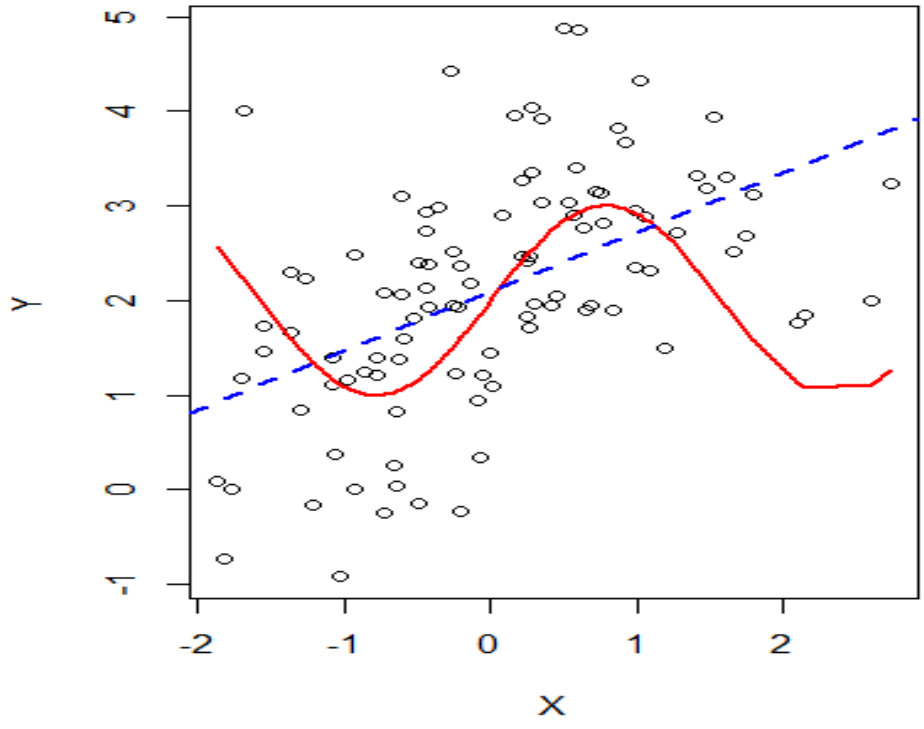
	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	2.0999	0.1070	19.63	< 2e-16	***
trainData$x	0.6266	0.1060	5.91	4.98e-08	***

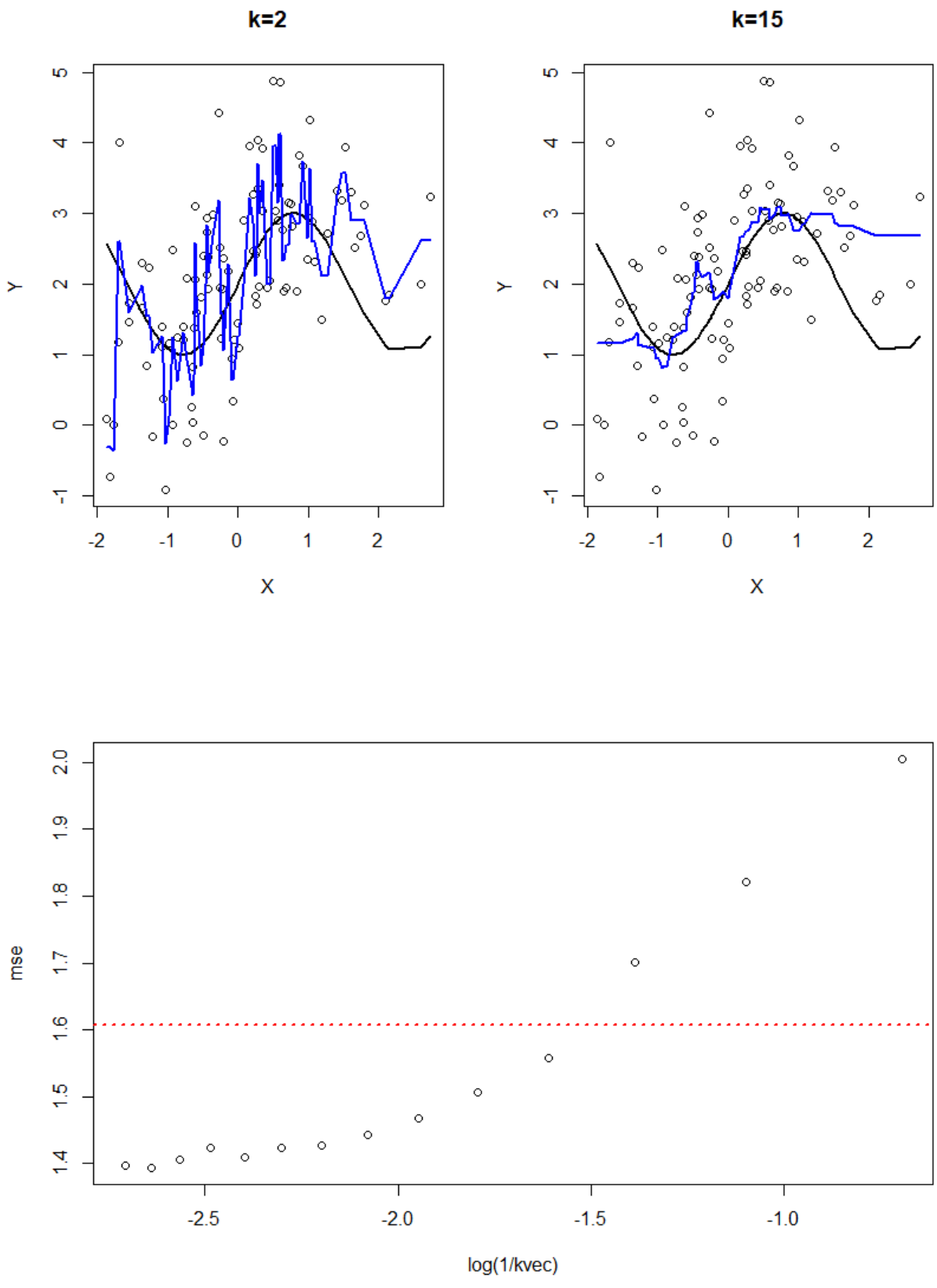
Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.069 on 98 degrees of freedom

Multiple R-squared: 0.2628, Adjusted R-squared: 0.2552

F-statistic: 34.93 on 1 and 98 DF, p-value: 4.979e-08



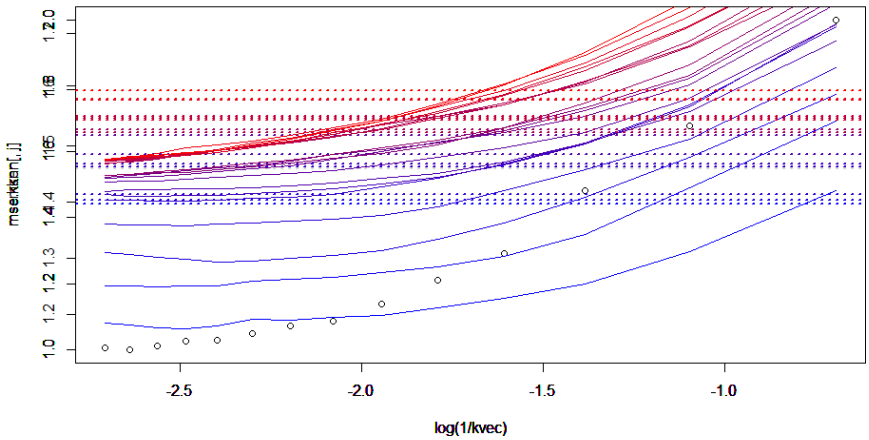

k=2
k=15
Y
X
mse
log(1/kvec)

Minimum MSE

1. KNN – 1.138171
2. Linear regression – 1.459013

KNN produces better model that produces lower MSE than linear regression does

Q8



Increasing number of the variable increases MSE in both knn and regression models. Knn produces lower MSE than linear model does

Q3

The most reviews were written by reviewer U584295664
The most reviews were written about product I760611623
The most similar user to the reviewer U141954350 is reviewer U887577623
The item recommended to reviewer U141954350 is product I279487841

For your reference, the Python code below was used to arrive at our answers.

```
import pandas as pd
import gzip
from scipy.sparse import csr_matrix
import sklearn
from sklearn.decomposition import TruncatedSVD
from sklearn.neighbors import NearestNeighbors
```

```
from scipy.spatial.distance import correlation

def parse(path):
    g = gzip.open(path, 'rb')
    for l in g:
        yield eval(l)

def getDF(path):
    i = 0
    df = {}
    for d in parse(path):
        df[i] = d
        i += 1
    df = pd.DataFrame.from_dict(df, orient='index')
    df_ratings = df[['reviewerID', 'itemID','rating']]
    df_ratings = df_ratings.groupby('reviewerID').filter(lambda x : len(x)>2)
    df_ratings = df_ratings.groupby('itemID').filter(lambda x : len(x)>3)
    return df_ratings

# Identify user who has rated the most video games

def identify_most_reviews_written(df):
        reviewer = df['reviewerID'].mode()
#         reviews = df.loc[df['reviewerID']==reviewer[0]]
        return reviewer[0]

# Which video games has been rated by the most amount of users?

def identify_product_with_most_reviews(df):
        product = df['itemID'].mode()
        return product[0]

# Find the user that is most similar to “U141954350”.

def get_adjusted_df(df):
        user_df = df.drop_duplicates(subset=['reviewerID','itemID'], keep='last')
```

```
    df_pivot = user_df.pivot(index='reviewerID',
columns='itemID',values='rating').fillna(0)
    return df_pivot

def calculate_knn(df,reviewer):
    matrix = csr_matrix(df.values)
    model_knn = NearestNeighbors(metric='cosine', algorithm = 'brute')
    model_knn.fit(matrix)
    distances, indices = model_knn.kneighbors(df.loc[reviewer, :].values.reshape(1,-1),
n_neighbors=20)
    return distances.flatten(), indices.flatten()

def most_similar(df_adjusted,indices_flat, distances,reviewer, rank):
    user = df_adjusted.index[indices_flat[rank]]
    return user

def reviewed_items(df_adjusted, user):
    reviewer_items =
df_adjusted.loc[user].iloc[df_adjusted.loc[user].to_numpy().nonzero()[0]]
    reviewer_items = reviewer_items.index.tolist()
    return reviewer_items

def make_recommendation(df_adjusted, reviewer, k, indices, distances):
    # Loop through similar users until you hit k or until no more users.
    valid_users = pd.DataFrame(columns=df_adjusted.columns)
    rank = 1
    reviewer_reviewed_items = reviewed_items(df_adjusted, reviewer)
    distance_rating = []
    max_range = max(k-1, len(distances)-1)
    while rank < max_range:
        similar = most_similar(df_adjusted, indices, distances, reviewer, rank)
        items = reviewed_items(df_adjusted, similar)
        result = all(elem in reviewer_reviewed_items for elem in items)
        if not result:
            temp = df_adjusted.loc[similar]/distances[rank]
            valid_users = valid_users.append(temp)
```

```
            rank = rank + 1
        mean = valid_users.mean(axis=0)
        recommendation = mean.idxmax(axis=0, skipna=True)
        return recommendation

def answers():
        # Recommend a video game to the user “U141954350”. For example, you may find a video game that the user “U141954350” has not bought yet and you expect that he would rate it high. Explain the recommendation strategy you used.
        df = getDF('assignments/week1/videoGames.json.gz')
        reviewer = 'U141954350'
        most_reviews = identify_most_reviews_written(df)
        print('The most reviews were written by reviewer '+most_reviews)
        most_reviewed_product = identify_product_with_most_reviews(df)
        print('The most reviews were written about product '+most_reviewed_product)
        df_adjusted = get_adjusted_df(df)
        distances, indices = calculate_knn(df_adjusted,reviewer)
        similar = most_similar(df_adjusted, indices, distances,reviewer, 1)
        print('The most similar user to the reviewer '+reviewer+' is reviewer '+similar)
        recommendation = make_recommendation(df_adjusted, reviewer, len(distances), indices, distances)
        print('The item recommended to reviewer '+reviewer+' is product '+recommendation)
```