STATS503 HW1

Ningyuan Wang

1/29/2020

Question 1

a. Better. Becuase we have large sample size but small number of predictors. Using flexible method and increasing model complexity may better fit large number of data.

b. Worse. Since we have small number of observations but large number of predictors, using flexible method may overfit the data which is not an optimal choice. Also, for small data size, complex model is hard to interpret and explain.

c. Better. Since the relationship between predictors and outcome is non-linear, incereasing complexity of model and using flexible method will better fit the data.

d. Worse, since the variance of error is high, consider to the trade off between bias and variance, we probably need simplify model to decrease the variance.

Question 2

Data Exploration and Cleaning

Based on numerical summary and data plot, we noticed some possible data entry errors in variables, such as BMI has many 0 values. Also, some variables did not have obvious difference between two groups of outcome, sich as skin thickness. Therefore, we decided to clean data to prepare for further analysis.

```
# load the datasets
train = read.csv("diabetes_train.csv") %>% mutate(Outcome = as.factor(Outcome))
test = read.csv("diabetes_test.csv") %>% mutate(Outcome = as.factor(Outcome))

summary(train) # check missing values and data pattern
```

```
##   Pregnancies        Glucose      BloodPressure    SkinThickness
##  Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00
##  1st Qu.: 1.000   1st Qu.:103.0   1st Qu.: 64.00   1st Qu.: 0.00
##  Median : 3.000   Median :123.0   Median : 72.00   Median :22.50
##  Mean   : 4.054   Mean   :124.8   Mean   : 69.67   Mean   :20.07
##  3rd Qu.: 7.000   3rd Qu.:145.0   3rd Qu.: 80.00   3rd Qu.:32.00
##  Max.   :17.000   Max.   :199.0   Max.   :114.00   Max.   :99.00
##     Insulin           BMI        DiabetesPedigreeFunction      Age
##  Min.   :  0.00   Min.   : 0.00   Min.   :0.0780           Min.   :21.00
##  1st Qu.:  0.00   1st Qu.:27.88   1st Qu.:0.2537           1st Qu.:25.00
##  Median :  0.00   Median :32.50   Median :0.4025           Median :31.00
##  Mean   : 84.07   Mean   :32.55   Mean   :0.5023           Mean   :34.33
##  3rd Qu.:130.00   3rd Qu.:36.80   3rd Qu.:0.6750           3rd Qu.:41.25
##  Max.   :846.00   Max.   :59.40   Max.   :2.4200           Max.   :81.00
##  Outcome
##  0:223
##  1:205
```

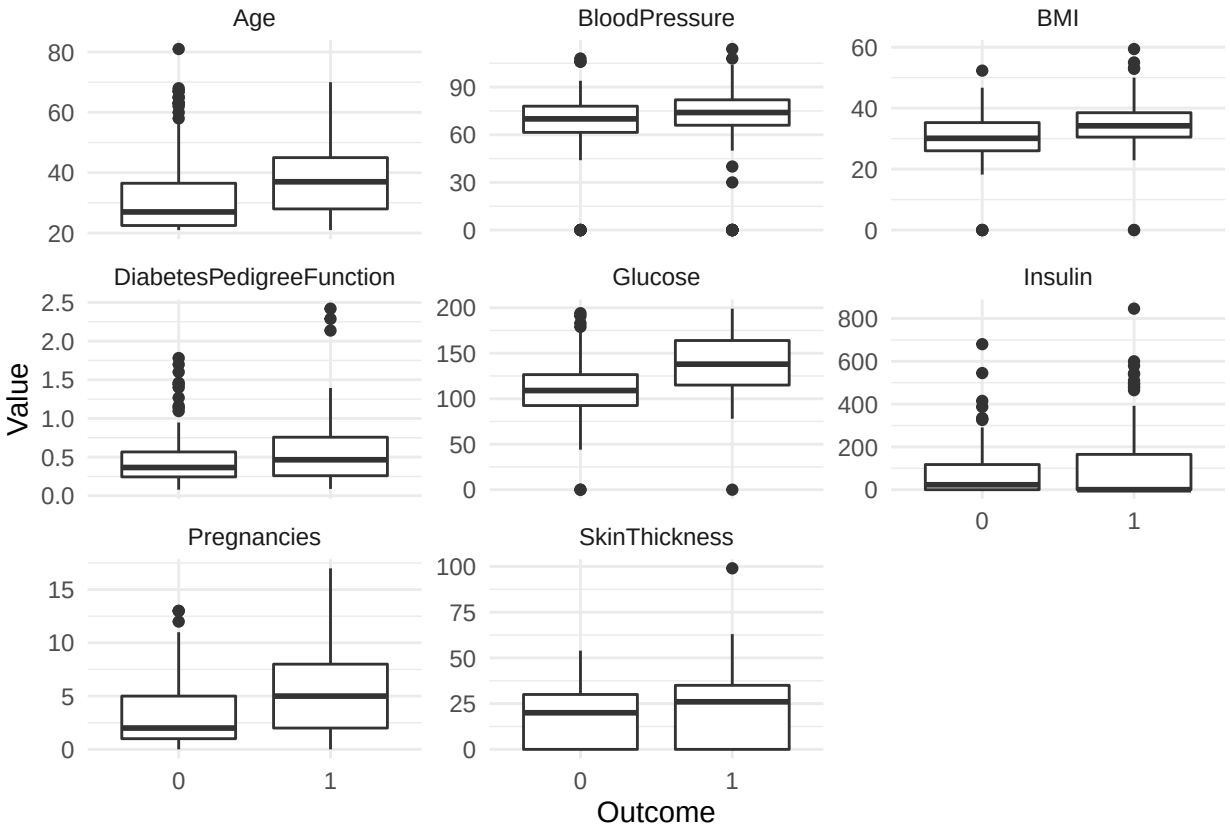
```
##
##
##
##
```

```
dim(train)
```

```
## [1] 428   9
```

```
dat.explore = gather(train, key = "Variable", value = "Value", -c("Outcome"))

ggplot(dat.explore) + geom_boxplot(aes(x = Outcome, y = Value)) + facet_wrap(.~Variable, scales = "free_
```



First, we tried deleting observations with any missing value, and it resulted that observations decreased from 428 to 209. In that case, we didn't think those 209 samples were representative for the whole dataset. In next analysis part, we also tried using the 209 samples fit KNN model, but the best choice for K is 1 which is not a useful model for us due to model complexity.

```
# delete observations with data entry error
train1 = train %>%
  filter(Glucose>0, BloodPressure>0, BMI>0, SkinThickness>0, Insulin>0)

test1 = test %>%
  filter(Glucose>0, BloodPressure>0, BMI>0, SkinThickness>0, Insulin>0)

dim(train1)
```

```
## [1] 209   9
```

Finally, we decided to remove two variables (i.e.insurlin and skin thickness) rather than deleting observations with any missing value, because above two variables did not have obvious difference in two groups of outcomes. Also, we deleted four observations in the training dataset since those observations have at least two variables with missing values. We applied same method for testing data as well.

After cleaning data, training dataset ended with 6 (predictor) variables and 424 observations.

```
# delete variables without obvious difference in two groups of outcomes
train2 = train %>% select(-c("Insulin", "SkinThickness")) %>% filter(BloodPressure>0 | BMI>0)
test2 = test %>% select(-c("Insulin", "SkinThickness")) %>% filter(BloodPressure>0 | BMI>0)
dim(train2)
```

```
## [1] 424   7
```

KNN Classification

We used KNN to calsiffy the outcome with 6 variables. Since the predictors are of different units and scales, we standardized variables first and then considered K from 1 to 20 in order to find best classification. At last, with table and plot below, we found K= 8 is an optimal choice for KNN classificaton with lowerst test error rate.

```
# pull out the outcome variable and the designing matrix for the training and test data
train_label = train2 %>% .$Outcome
train_x = train2 %>% select(-"Outcome")

test_label = test2 %>% .$Outcome
test_x = test2 %>% select(-"Outcome")
```

```
# scale both training and test data in the same way
mean_train = colMeans(train_x)
std_train = sqrt(diag(var(train_x)))
# training data
train_x = scale(train_x, center = mean_train, scale = std_train)
# test data
test_x = scale(test_x, center = mean_train, scale = std_train)

set.seed(503)
```

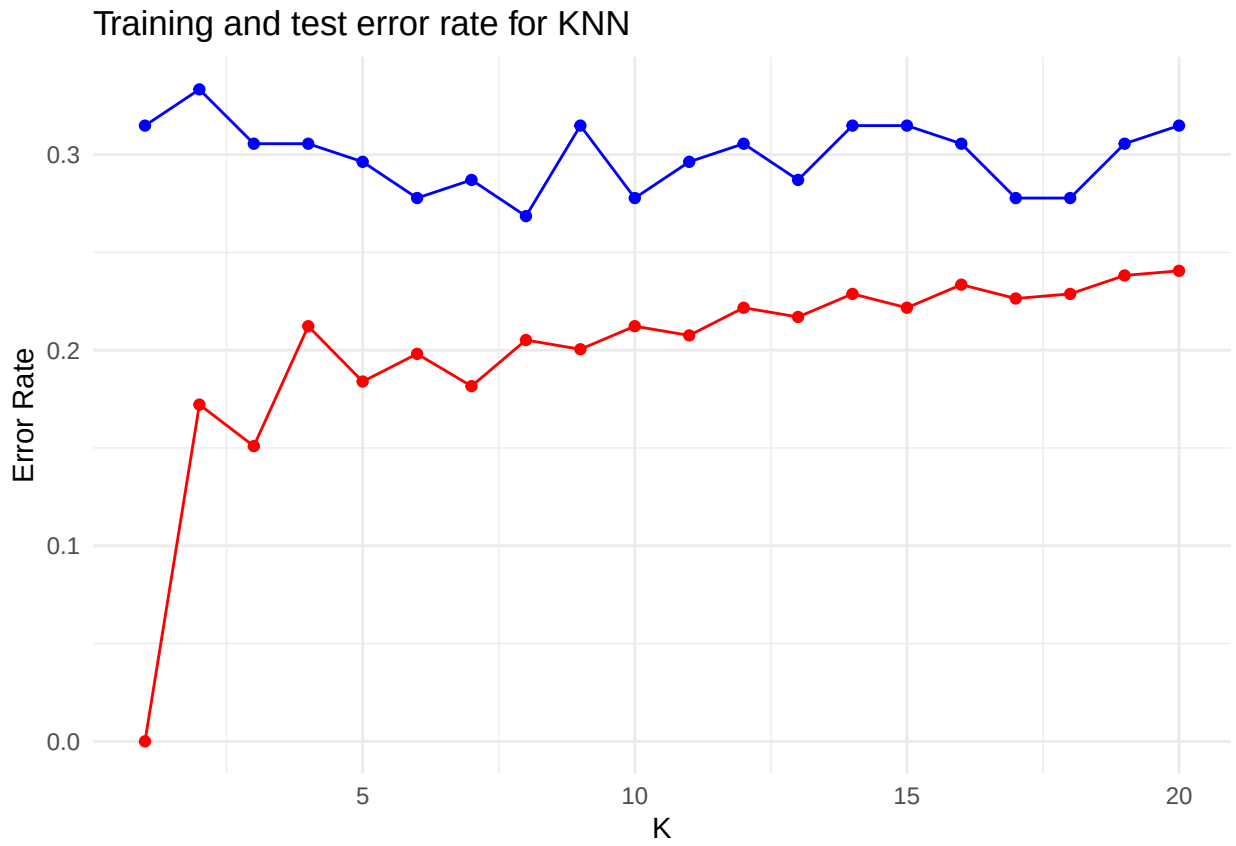
```
# look for a best k to minimize training error
k_range = 1:20
train_error = c()
test_error = c()

for(i in 1:length(k_range)){
  pred_train <- knn(train_x,
                    train_x,
                    train_label,
                    k = k_range[i])
  train_error[i] = mean(pred_train != train_label)
  pred_test = knn(train_x,
                  test_x,
```

```
                  train_label,
                  k = k_range[i])
  test_error[i] = mean(pred_test != test_label)
} # for each iteration in the for loop, we fit a KNN regression model using the same dataset,
# with different choice of k as k[i]. We compute the training and testing error rate each round.

errors = data.frame(train_error, test_error, k_range)

ggplot(errors, aes(x = k_range)) +
  geom_line(aes(y = train_error), col = "red") + geom_point(aes(y = train_error), col = "red") +
  geom_line(aes(y = test_error), col = "blue") + geom_point(aes(y = test_error), col = "blue") +
  ylab("Error Rate") + xlab("K") + ggtitle("Training and test error rate for KNN") + theme_minimal()
```




```
errors %>% arrange(test_error) # arrange error rate by decending test error rate
```

```
##    train_error test_error k_range
## 1    0.2051887  0.2685185       8
## 2    0.1981132  0.2777778       6
## 3    0.2122642  0.2777778      10
## 4    0.2264151  0.2777778      17
## 5    0.2287736  0.2777778      18
## 6    0.1816038  0.2870370       7
## 7    0.2169811  0.2870370      13
## 8    0.1839623  0.2962963       5
## 9    0.2075472  0.2962963      11
## 10   0.1509434  0.3055556       3
```

```
## 11   0.2122642  0.3055556       4
## 12   0.2216981  0.3055556      12
## 13   0.2334906  0.3055556      16
## 14   0.2382075  0.3055556      19
## 15   0.0000000  0.3148148       1
## 16   0.2004717  0.3148148       9
## 17   0.2287736  0.3148148      14
## 18   0.2216981  0.3148148      15
## 19   0.2405660  0.3148148      20
## 20   0.1721698  0.3333333       2
```