Week 10: K Nearest Neighbors (KNN)

DSC 365: Introduction to Data Science

2024-10-29

```
library(tidyverse)
library(Metrics)
```

"Lazy" learning

So far we've focused on building models that can predict outcomes on a new set of data. Another approach is to just be *lazy*!

Lazy learning: no assumptions necessary to classify data

- How does that work?

Example: Consider the plot below - describe the relationship between x and y.

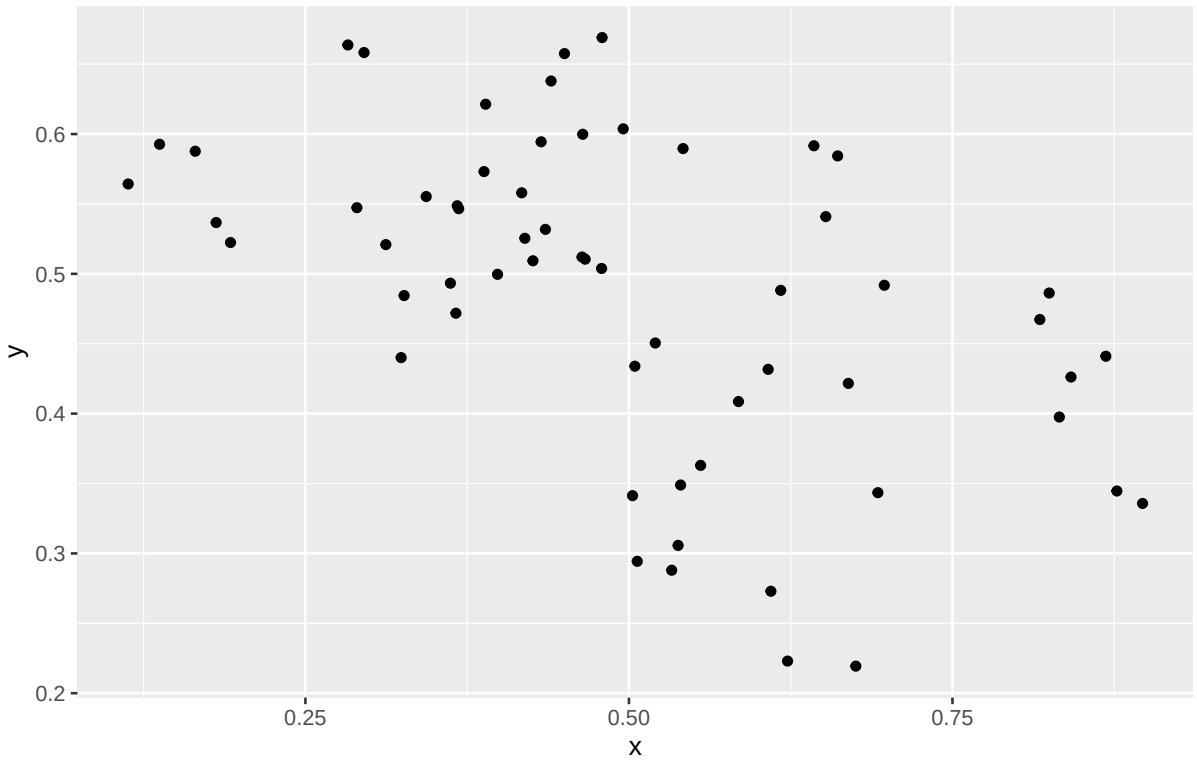
What if the data points belonged to three different groups, like this?

How should a new data point, $(0.2, 0.5)$ be classified? What about $(0.4, 0.2)$?

k-nearest neighbor (k-NN or KNN):

A non-parametric supervised learning method that can be used for both classification and regression

1. In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small).
2. In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors.



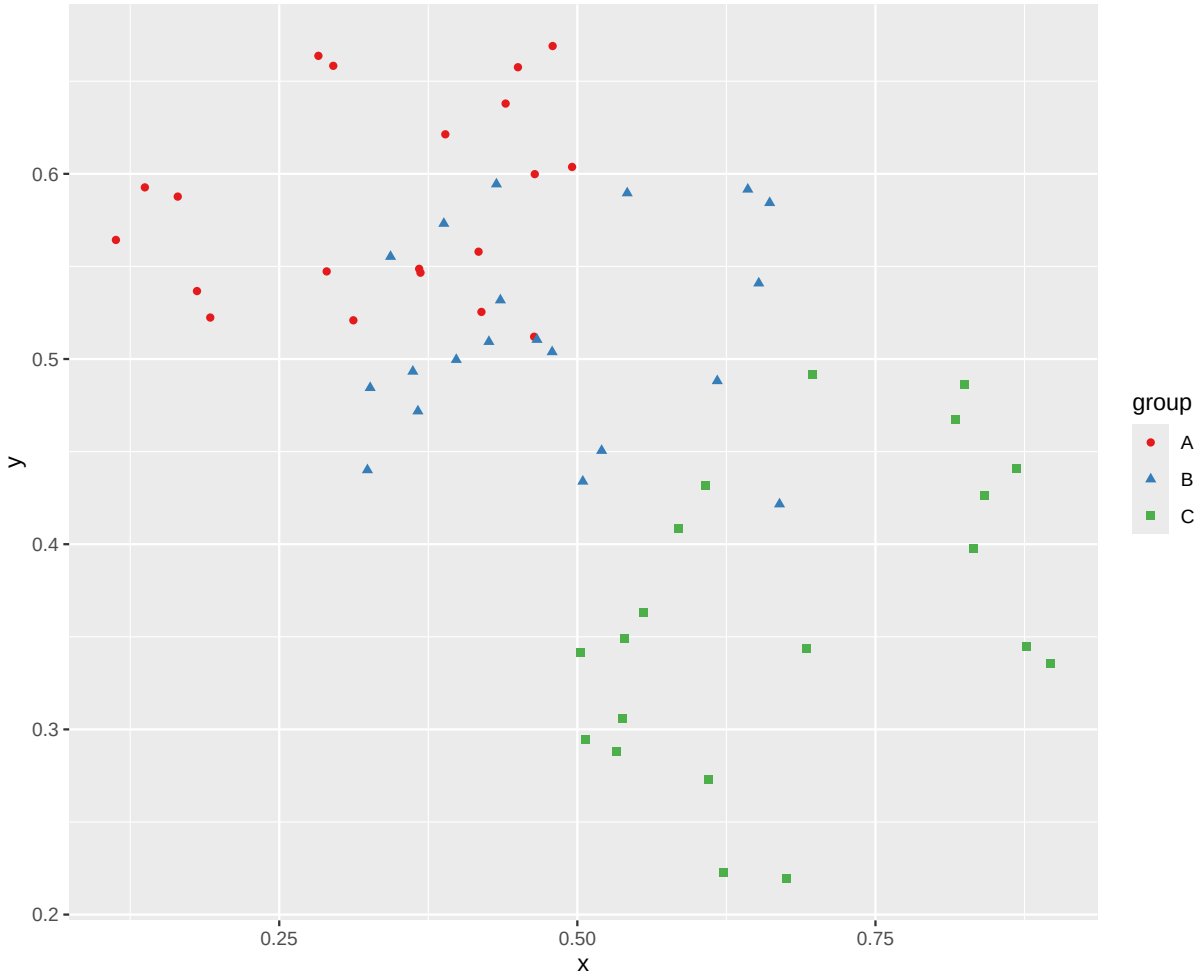
KNN for Classification Steps

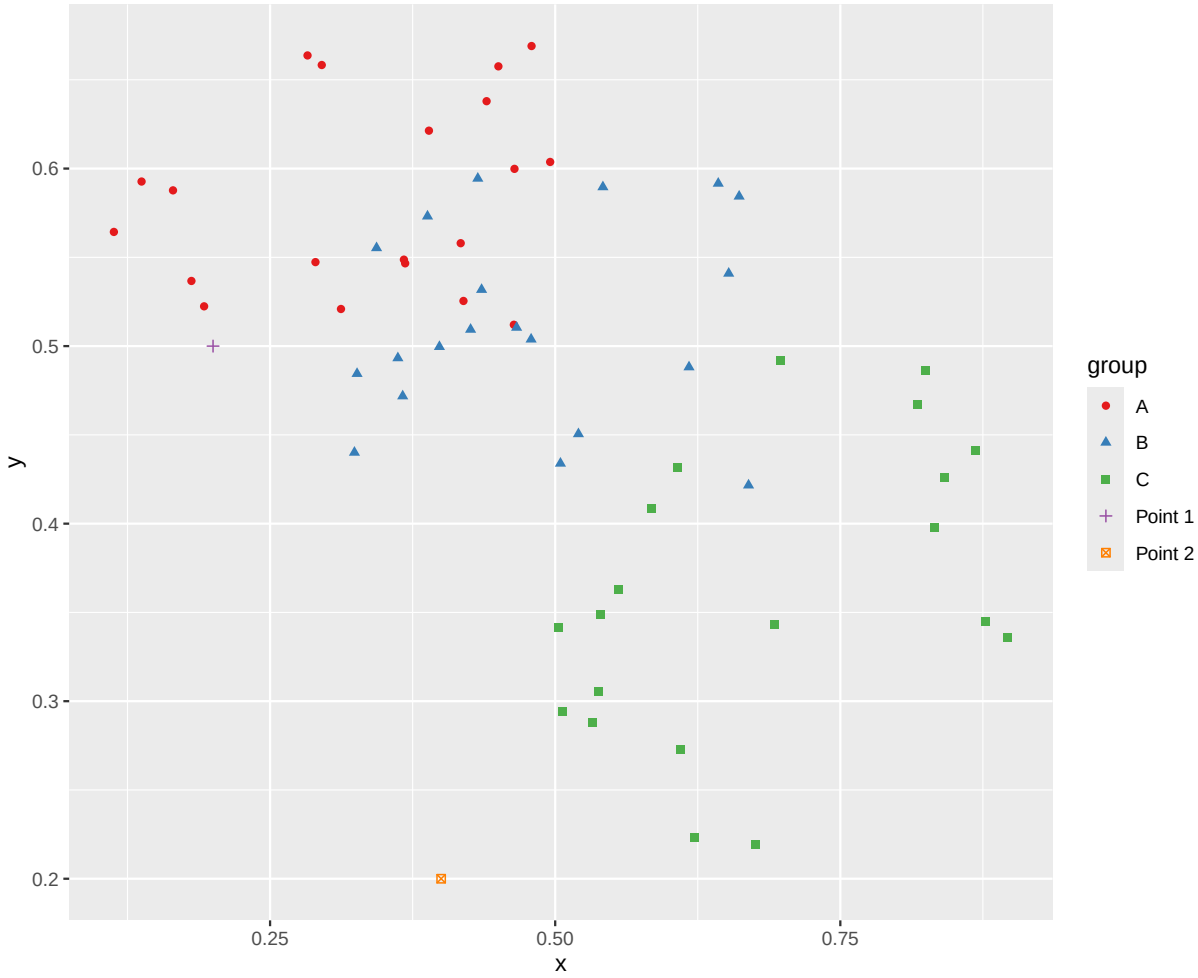
In the KNN algorithm, k specifies the number of neighbors and its algorithm is as follows:

- Choose the number k of neighbors
- Find the k Nearest Neighbor of an unknown data point using distance.
  - Euclidean Distance: $d(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$
- Among the k-neighbors, count the number of data points in each category (or find the conditional probability the new point (x_0) is in category j).
  - $P(Y = j|X = x_0) = \frac{1}{k}\sum_{i \in N_0} I(y_i = j)$
- Assign the new data point to a category, where you counted the most neighbors/highest probability from formula above.

`knn()`

Example: Let's classify our new points using $k = 2$.


0.6
0.5
0.4
0.3
0.2
y
0.25
0.50
0.75
x
group
A
B
C


0.6
0.5
0.4
0.3
0.2
y
0.25
0.50
0.75
x
group
A
B
C
Point 1
Point 2

```r
library(class)

knnModel = knn(train = data[,1:2],
               test = new.points[,1:2],
               cl = data$group,
               k = 2, prob = TRUE)
knnModel
```

```
[1] A C
attr(,"prob")
[1] 1 1
Levels: A B C
```

What if we use more points, $k = 10$?

```r
knnModel = knn(train = data[,1:2],
               test = new.points[,1:2],
               cl = data$group,
               k = 10, prob = TRUE)
knnModel
```

```
[1] A C
attr(,"prob")
[1] 0.7 0.8
Levels: A B C
```

Advantages and disadvantages of k-nearest neighbors

The algorithm is easy to implement and straight forward. New data can be added seamlessly which will not impact the accuracy of the algorithm. No training period, relatively fast.

However, sometimes it is hard to decide the k value. With more variables included, the accuracy will be affected. Sensitive to the outliers, you need to scale the data sometimes.

Further, to provide accurate classification, KNN requires a lot of observations relative to the number of predictors—that is, n much larger than p.

Choice of k

- $k = 1$: low bias, but high variance
- $k = 100$: low variance, but high bias

Example: Credit Utilization

Example: Can we use kNN to predict which utilization quantile a new customer falls into based on their application data?

```
library(ISLR)
data(Credit)
Credit <- Credit %>% mutate(Utilization = Balance/Limit) %>%
  mutate(Quartile = ifelse(Utilization<0.01851, 'Q1',
                           ifelse(Utilization<0.09873, 'Q2',
                           ifelse(Utilization<0.14325, 'Q3',
                                  'Q4'))))
```

We want to predict the utilization quantile based on two application characteristics: credit rating and age.

New applicants:

Name	Age	Credit Rating
Lacey	33	750
Zach	47	400
Ashlee	21	250

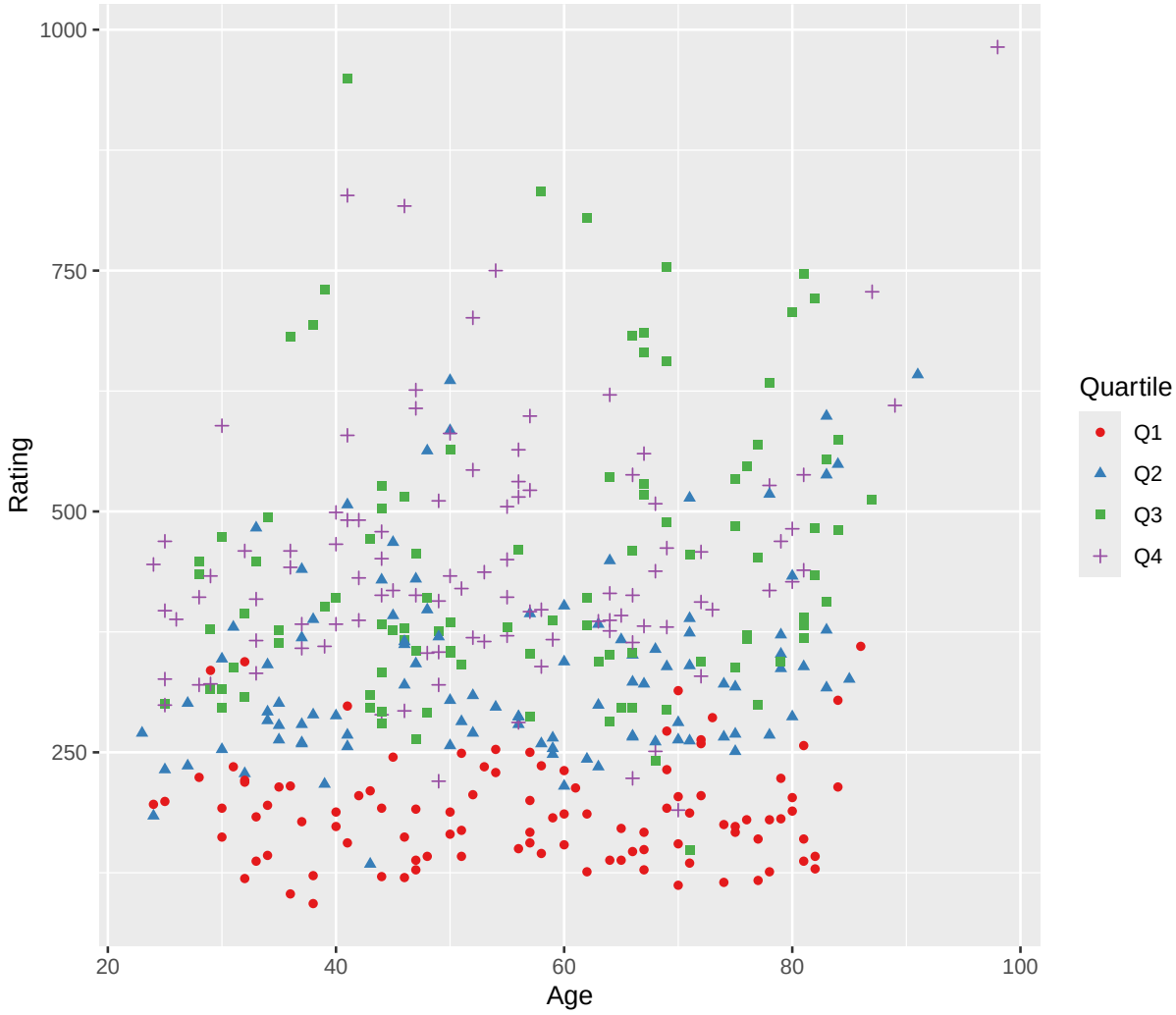
```
applicants <- data.frame(Age = c(33, 47, 21),
                         Rating = c(750, 400, 250),
                         Quartile=c('New', 'New', 'New'))
```

Plotting the new applicants. Use `fct_relevel()` to reorder the categories.

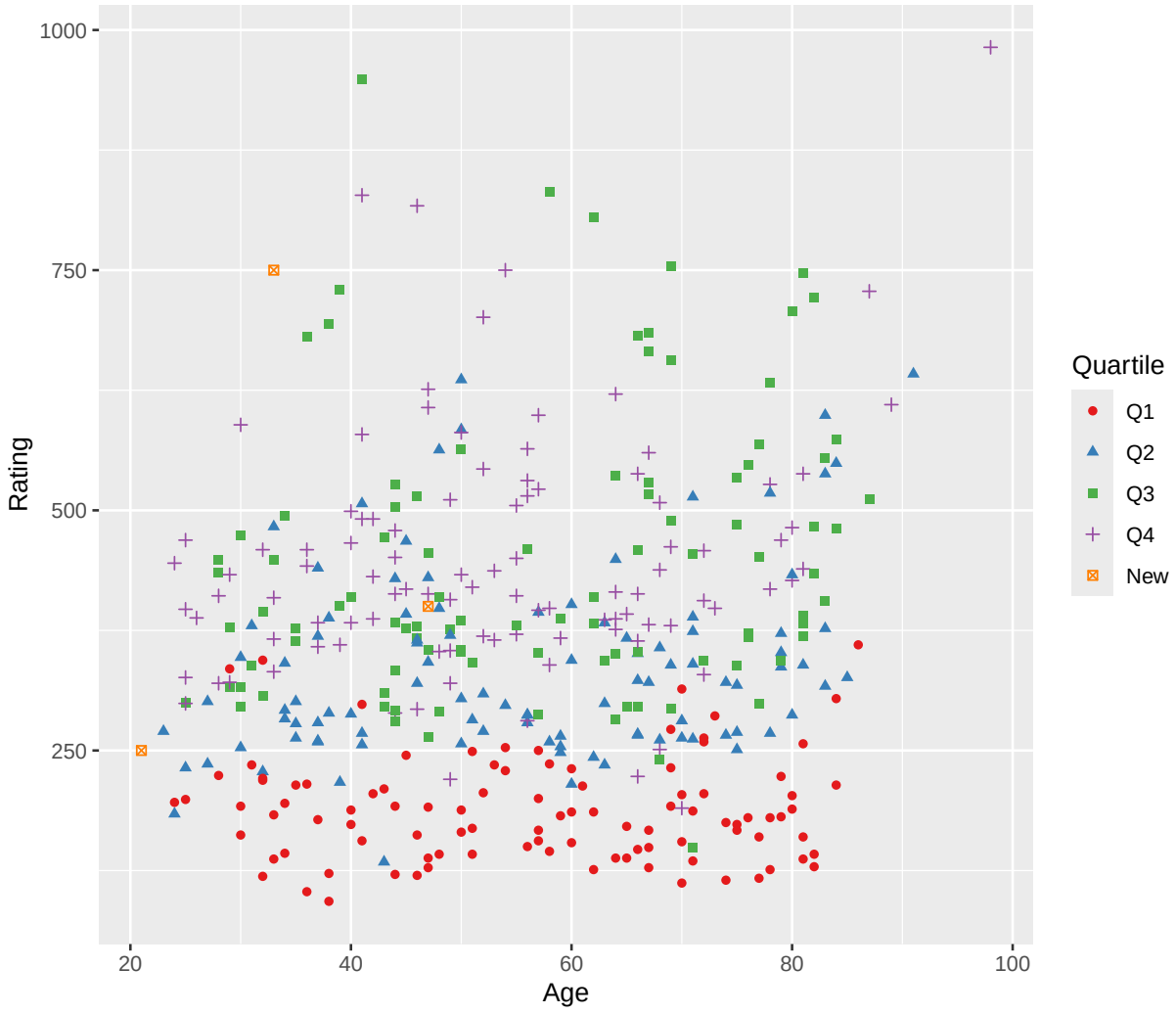
```
old = Credit %>% dplyr::select(Age, Rating, Quartile)
full = rbind(old, applicants)
full = full %>%
  mutate(Quartile = fct_relevel(Quartile,
                                "Q1", "Q2", "Q3", "Q4", "New"))
str(full$Quartile)
```

```
Factor w/ 5 levels "Q1","Q2","Q3",..: 2 3 2 3 2 3 2 3 2 4 ...
```

```
ggplot(full, aes(x=Age, y=Rating, group=Quartile)) +
  geom_point(aes(col=Quartile, pch=Quartile)) +
```


1000
750
500
250
Rating
20
40
60
80
100
Age
Quartile
Q1
Q2
Q3
Q4

```
  scale_color_brewer(palette='Set1')
```



Fit a first model with $k = 10$.

```
knn10 = knn(train = old[,1:2],
            test = applicants[,1:2],
            cl = old[,3],
            k = 10, prob = TRUE)
knn10
```

```
[1] Q3 Q4 Q2
```

```
attr(,"prob")
[1] 0.7 0.4 0.8
Levels: Q1 Q2 Q3 Q4
```

What if we use $k = 20$?

```
knn20 = knn(train = old[,1:2],
            test = applicants[,1:2],
            cl = old[,3],
            k = 20, prob = TRUE)
knn20
```

```
[1] Q3 Q4 Q2
attr(,"prob")
[1] 0.65 0.40 0.65
Levels: Q1 Q2 Q3 Q4
```

Going bigger: $k = 100$

```
knn100 = knn(train = old[,1:2],
            test = applicants[,1:2],
            cl = old[,3],
            k = 100, prob = TRUE)
knn100
```

```
[1] Q4 Q4 Q2
attr(,"prob")
[1] 0.44 0.46 0.45
Levels: Q1 Q2 Q3 Q4
```

Evaluate models with Cross-validation

Example: Let's add some more dimensions to the model. We want to know if k-nearest neighbor is effective at predicting quartile membership using an applicant's age, credit rating, income, number of existing credit cards, and education level. I'll randomly select 100 observations for testing, and assign the other 300 to my training data set.

```
set.seed(365)
test_ID = sample(1:nrow(Credit), size = 100)
TEST = Credit[test_ID,]
```

```
TRAIN = Credit[-test_ID, ]
```

Now, we'll set the testing data as "new data", and make predictions using the k-nearest neighbors from the training data.

```
knn_train = TRAIN %>% dplyr::select(Age, Rating, Income, Cards, Education)
knn_test = TEST %>% dplyr::select(Age, Rating, Income, Cards, Education)


knn50 = knn(train = knn_train,
            test = knn_test,
            cl = TRAIN$Quartile,
            k = 50, prob = TRUE)
knn50
```

```
 [1] Q4 Q1 Q4 Q1 Q4 Q3 Q4 Q4 Q1 Q4 Q2 Q2 Q4 Q4 Q3 Q1 Q1 Q3 Q4 Q1 Q4 Q2 Q3 Q2 Q2
[26] Q4 Q3 Q4 Q4 Q2 Q2 Q2 Q3 Q2 Q1 Q4 Q4 Q3 Q3 Q2 Q1 Q2 Q3 Q4 Q1 Q4 Q3 Q2 Q4 Q4
[51] Q4 Q2 Q4 Q1 Q4 Q1 Q1 Q2 Q1 Q1 Q1 Q4 Q1 Q4 Q4 Q1 Q4 Q4 Q4 Q4 Q2 Q1 Q4 Q2 Q3
[76] Q2 Q2 Q1 Q1 Q4 Q2 Q4 Q2 Q3 Q2 Q4 Q1 Q4 Q4 Q2 Q4 Q4 Q4 Q4 Q2 Q2 Q4 Q2 Q1 Q2
attr(,"prob")
 [1] 0.66 0.86 0.54 0.96 0.50 0.40 0.36 0.52 0.96 0.66 0.54 0.56 0.54 0.42 0.38
[16] 0.96 0.92 0.42 0.46 0.98 0.44 0.52 0.40 0.54 0.60 0.46 0.46 0.42 0.62 0.52
[31] 0.52 0.56 0.40 0.54 0.58 0.46 0.52 0.38 0.46 0.52 0.96 0.44 0.44 0.42 0.98
[46] 0.50 0.44 0.48 0.54 0.46 0.50 0.58 0.46 0.90 0.50 0.94 0.96 0.42 0.96 0.82
[61] 0.54 0.48 0.54 0.58 0.44 0.96 0.44 0.46 0.58 0.54 0.34 0.96 0.44 0.58 0.42
[76] 0.34 0.54 0.96 0.96 0.42 0.60 0.52 0.56 0.40 0.48 0.54 0.64 0.34 0.60 0.56
[91] 0.44 0.50 0.48 0.50 0.46 0.52 0.44 0.58 0.78 0.58
Levels: Q1 Q2 Q3 Q4
```

Now, we'll set the testing data as "new data", and make predictions using the k-nearest neighbors from the training data.

```
#Create Confusion Matrix
t = table(knn50, TEST$Quartile)
t
```

```
knn50 Q1 Q2 Q3 Q4
   Q1 18  2  1  1
   Q2  4 16  5  1
   Q3  0  3  7  2
```

```
   Q4  0  8 16 16
```

```
sum(diag(t))/nrow(TEST) #Classification Accuracy
```

```
[1] 0.57
```

Try by yourself

Let's see whether we can predict people's ethnicity based on their credit card information (Age, Rating, Income, Cards, Education, Balance). Try to fit a KNN model with K = 5, 10, 25, 50, 100. See how the prediction values change and think about why.

```
knn_train = TRAIN %>% dplyr::select(Age, Rating, Income, Cards, Education, Balance)
knn_test = TEST %>% dplyr::select(Age, Rating, Income, Cards, Education, Balance)

for(i in c(5, 10, 25, 50, 100)){
  knnModel = knn(train = knn_train,
            test = knn_test,
            cl = TRAIN$Quartile,
            k = i, prob = TRUE)
  t = table(knnModel, TEST$Quartile)
  print(sum(diag(t))/nrow(TEST))
}
```

```
[1] 0.9
[1] 0.87
[1] 0.82
[1] 0.76
[1] 0.72
```

KNN for Regression Steps

In the KNN algorithm, k specifies the number of neighbors and its algorithm is as follows:

- Choose the number k of neighbors
- Find the k Nearest Neighbor of an unknown data point using distance.
  - Euclidean Distance: $d(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$

- Estimates $f(x_0)$ (response value of the new observation) using the average of all the training responses of the k Nearest Neighbors
  - $\hat{f}(x_0) = \frac{1}{K} \sum_{x_i \in N_0} y_i$

Example: Credit Utilization

```
knn_train = TRAIN %>% dplyr::select(Age, Rating, Income, Cards,
                                    Education, )
knn_test = TEST %>% dplyr::select(Age, Rating, Income, Cards,
                                  Education)

knn50 = FNN::knn.reg(train = knn_train,
             test = knn_test,
             y = TRAIN$Utilization,
             k = 50)

head(knn50$pred, n = 5)
```

```
[1] 0.157343839 0.011478125 0.144051032 0.005360386 0.142495141
```

```
rmse(TEST$Utilization, knn50$pred)
```

```
[1] 0.0386645
```

Linear Regression vs KNN Regression

In General:

Parametric approaches will outperform nonparametric approaches if the parametric form that has been selected is close to the true form of the data

So in situations where the relationship is nonlinear, KNN regression may perform better than Linear Regression

- But, Linear Regression may actually perform better than KNN as the number of variables (dimensions) increase
  - curse of dimensionality

Try by yourself

Run a multiple linear regression model on the same data, and compare the RMSE with the KNN regression model. Which method is performing better? Will increasing/decreasing k lower the RMSE?

```
## MLR
mod.fit = lm(Utilization ~ Age + Rating + Income, Cards + Education,
          data = TRAIN)

predictions <- predict(mod.fit, TEST)

rmse(TEST$Utilization, predictions)
```

```
[1] 0.04203612
```

Summary

Linear model: The basic model for regression, easy for interpretation, but has strict assumption thus hard to get a better prediction

Quadratic model: If you see a quadratic trend (in the EDA plot or residual plots) when fitting the linear model, you may consider quadratic model

Decision tree: Can be applied to regression and classification. Has good data visualization but it has high variance.

Random Forest: a collection of tree models. Hard for interpretation but it can output variable importance. Can be useful if you have a lot of variable and what want to select the most useful ones. Also, if you have variables are not independent with each other, it performs better than the linear model.

KNN Can be applied to regression and classification. relatively fast, no assumption need and add data anytime. May affect the accuracy if we have too many variables and need to use CV to decide the k value.