Lab1 Special Tasks

mimte666 - Mim Kemal Tekin

25/11/2018

Special Task 1

Use the training and test data from assignment 1. The existing packages for nearest neighbor classification do not allow for using a distance measure which is often used to compare two text documents. This measure is based on a so called cosine similarity which for two vectors ***X*** *and* ***Y*** *is defined as:*

$$c(X,Y) = \frac{X^T Y}{\sqrt{\sum_i X_i^2}\sqrt{\sum_i Y_i^2}}$$

The corresponding distance is then defined as $d(X,Y) = 1 - c(X,Y)$

Task 1.1

Implement from scratch the K-nearest neighbors method which is based on distance function $d(X,Y)$ ***(use only basic R functions****). Your code should be presented as a function* ***knearest(data, K, newdata)*** *that uses data as trainin data and then returns the predicted class probabilities for newdata by using K-nearest neighbor approach.*

Task 1.2

Computer training and test misclassification errors for K=30 and compare these with errors obtained in assignment 1, step 4.

Table 1: Test with Train Data

	Predict (-)	Predict (+)
Actual (-)	845	100
Actual (+)	260	165

Table 2: Test with Test Data

	Predict (-)	Predict (+)
Actual (-)	795	142
Actual (+)	282	151

Table 3: Misclassification Rates

	rate of train data kknn	rate of test data kknn
Misclassification	0.2627737	0.3094891
Accuracy	0.7372263	0.6905109

Tables from Assignment 1.4

Table 7: Test with Train Data

	Predict (-)	Predict (+)
Actual (-)	807	138
Actual (+)	98	327

Table 8: Test with Test Data

	Predict (-)	Predict (+)
Actual (-)	672	265
Actual (+)	187	246

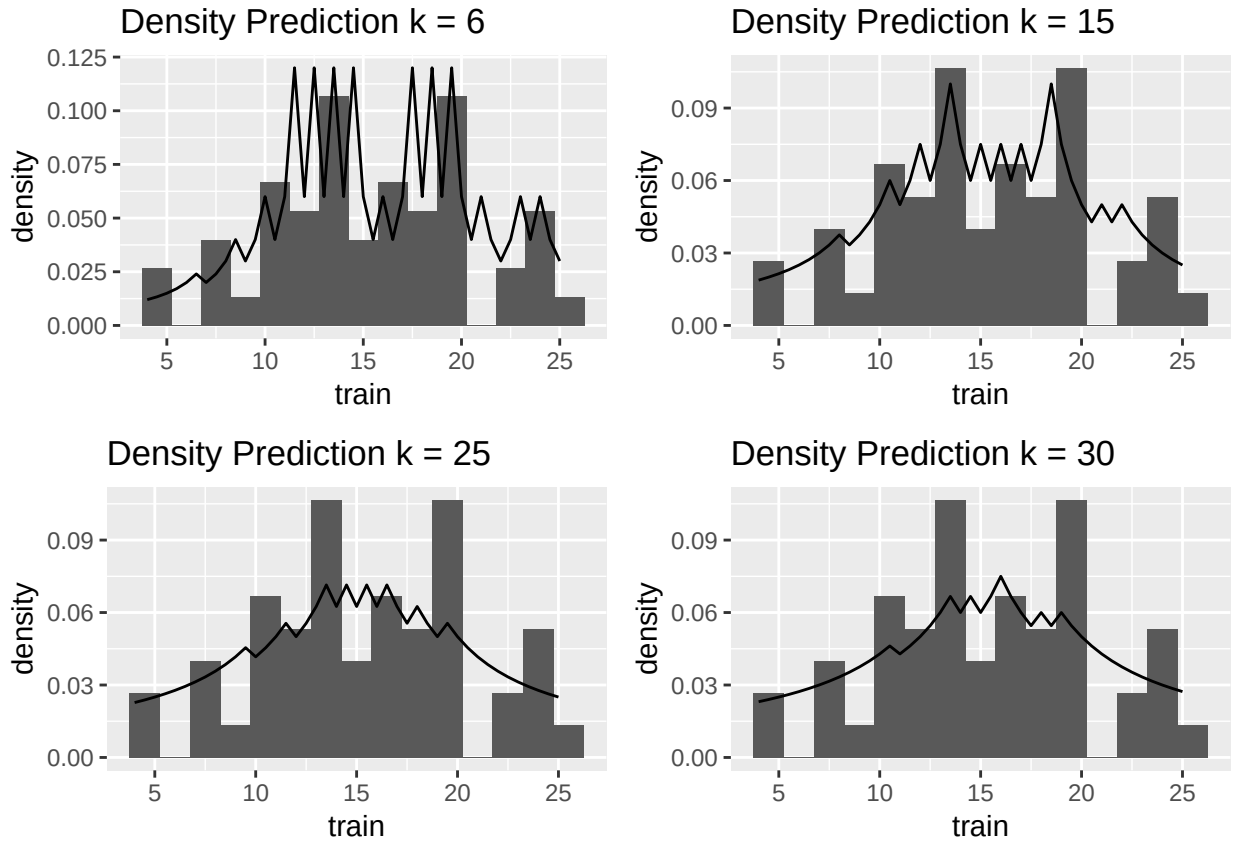
Table 9: Misclassification Rates

	Train Data	Test Data
Misclassification	0.1722628	0.329927
Accuracy	0.8277372	0.670073

We create a k-nearest neighbor algoritm in this lab, and we use cosine similarity to find similarities (distance) between observations. As we mentioned before, in the mandatory assignment, knn algorithm is a type of lazy learning and it stores training data. After prediction process starts, it calculates similarities of training data and newdata (the data which is predicted from). The knearest function can be found in appendix.
We test our data by predicting with train data and test data, separately. We can see from results by using cosine similarity, misclassification values are 0.2627 with train data and 0.3094 with test data. We can see there is a increase in misclassification after switching our newdata from train data to test data. Even accuracy rate is low with test data, this values are not at unacceptable level. Because these 2 predictions have quite same value, there is only 4,67% increasing.
If we remember the results of Assignment 1 Step 4 (Table 7, Table 8, Table 9), we had 0.1722 with train data and 0.3299 with test data as misclassification rates. If we compare these results, we can see they have almost double values in Assg1-Step1 which mean this model is successfull only testing with training data, because misclassification rate increases significantly. It is more logical to choose the model which is created by using cosine similarity, despite the fact that the misclassification rate is more than the model whic is in assignment 1. Because model with cosine similarity is more stable than the other one. We can say the other one is biased to the training data.

Special Task 2

Data set "cars" describes speeds of some cars in 1920. Type ?cars in R for more information. Implement k-nearest neighbor density estimation (use only basic R functions, no specialized packages) for the variable "speed" from cars data set. Use k=6 and present the histogram of the "speed" overlaid by the density curve computed by your density estimation algorithm. Conclusions?



In this task we use k-nearest neighbors' distances to make a prediction of densities. We can make this predictions by using this formula: $P(x|K) = \frac{K}{N.\Delta}$. In this formula K is number of neighbors, N is length of data and Δ is length of the interval containing K neighbors. In this formula Δ provides us to make weighted predictions by using the shortest length which contains k values. In this task we divide the range of the original data ([4,25]) by 0.5 and we predict density for each value in this vector.
We can see in the plot which shows k = 6, when the data density is intensive, our predicted density also peaks. Predicted densities also fills some missing bins and fits quite good to data. Additionally, we can observe when we increase k value, Predicted value is more smoothed and count of peaks decreases. Because when we increase k, the range of δ also increases and it makes smaller and closer predictions.

Appendix

```
knitr::opts_chunk$set(echo = F, warning = F, error = F)
##### TASK 1.1 #####

library(readxl)

conf_matrix = function(real_data, predicted_data){
  ct = table(real_data, predicted_data)
  rownames(ct) = c("Actual (-)", "Actual (+)")
  colnames(ct) = c("Predict (-)", "Predict (+)")
  return(ct)
}

calculate_mc_rate = function(confision_matrix, n){
  return(1-sum(diag(confision_matrix))/n)
}

knearest = function(data, K, newdata){
  # function that calculates distance matrix.
  dist_function = function(X, Y){
    # X = train data
    # Y = test data
    # output = cosine similarity matrix between X and Y, rows:X, cols:Y.
    x = as.matrix(X)
    y = as.matrix(Y)
    # calculate hats
    x_hat = x / sqrt( rowSums(x^2) )
    y_hat = y / sqrt( rowSums(y^2) )
    # calculate matrix C
    C = x_hat %*% t(y_hat)
    # calculate distance matrix
    D = 1 - C
    return(D)
  }

  # function that predict
  predict = function(Y_train, dis_matrix, k){
    if(k > length(Y_train))
      stop("k cannot be greater than data!")
    predict_Y = c()
    # iterate for all columns ( test data )
    for(col in 1:ncol(dis_matrix)){
      indexes = c()
      # k times find min value
      for(j in 1:k){
        # find min distances
        curr_index = which(dis_matrix[,col]==min(dis_matrix[,col]))[1]
        # store indexes
        indexes = c(indexes, curr_index)
        # change the value as biggest value for us
        # we do not want to take same min again
        dis_matrix[curr_index, col] = 1
```

```
      }
      # calculate probability
      prob = mean(Y_train[indexes])
      # store prob
      predict_Y = c(predict_Y, prob)
    }
    return(predict_Y)
  }

  X_train = data[,-ncol(data)]
  Y_train = data$Spam
  X_test = newdata[,-ncol(newdata)]
  Y_test = newdata$Spam

  dis_mat = dist_function(X_train, X_test)
  probs = predict(Y_train = Y_train, dis_matrix = dis_mat, k = K)

  return(probs)
}
##### TASK 1.2 #####

# import data
df_spambase = read_xlsx("../spambase.xlsx")

# divide data into training and test sets
n = dim(df_spambase)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
# training data
train = df_spambase[id,]
# test data
test = df_spambase[-id,]

# get predicted probabilities
prob_train = knearest(data = train, newdata = train, K = 30)
prob_test = knearest(data = train, newdata = test, K = 30)

# apply threshold
predicts_train = ifelse(prob_train > 0.5, 1, 0)
predicts_test = ifelse(prob_test > 0.5, 1, 0)

# create confusion matrix
cm_train = conf_matrix(real_data = train$Spam, predicted_data = predicts_train)
cm_test = conf_matrix(real_data = test$Spam, predicted_data = predicts_test)

# calculate misclassification rate
mc_rate_train = calculate_mc_rate(cm_train, length(predicts_train))
mc_rate_test = calculate_mc_rate(cm_test, length(predicts_test))
accuracy_train = 1 - mc_rate_train
accuracy_test = 1 - mc_rate_test

mc_rates = data.frame(train = c(mc_rate_train, accuracy_train),
           test = c(mc_rate_test, accuracy_test))
```

```r
colnames(mc_rates) = c("rate of train data kknn", "rate of test data kknn")
rownames(mc_rates) = c("Misclassification", "Accuracy")

knitr::kable(cm_train, caption = "Test with Train Data")
knitr::kable(cm_test, caption = "Test with Test Data")

knitr::kable(mc_rates, caption = "Misclassification Rates")

prob_test = knearest(data = train[13:18,], newdata = test[13:18,], K = 3)
##### TASK 2 #####

library(gridExtra)
library(ggplot2)

# data: original data which is we calculate distances
# newdata: the interval values that we will calculate densities
# k: k value
knn_density = function(data, newdata, k){
  # store length of original data
  N = length(data)
  predicts = c()
  # loop for interval values.
  # We will calculate densities for all values in interval
  for(i in 1:length(newdata)){
    # calculate and sort distances for the value in the interval
    dist = sort(abs(data - newdata[i]))
    # calculate data to prediction
    delta = 2 * dist[k]
    # prediction of densitiy
    p = k/(delta*N)
    # store prediction
    predicts = c(predicts, p)
  }
  # create a response dataframe
  df = data.frame(x = test, density = predicts)
  return(df)
}
# get train data
train = cars$speed
# create a interval from min to max and increase it by 0.5
test = seq(from = min(train), to = max(train), by=0.5)

# calculations by using differen k values to compare how k values effect
# differences in predictions of density
results_6 = knn_density(train, test, 6)
results_15 = knn_density(train, test, 15)
results_25 = knn_density(train, test, 25)
results_30 = knn_density(train, test, 30)

# plots
plot1 = ggplot() +
  geom_histogram(aes(x = train, y = ..density..), bins = 15) +
  geom_line(aes(x = results_6$x, y = results_6$density)) +
```

```
  scale_x_continuous(breaks = seq(from = 0, to = 25, by = 5)) +
  labs(title="Density Prediction k = 6")

plot2 = ggplot() +
  geom_histogram(aes(x = train, y = ..density..), bins = 15) +
  geom_line(aes(x = results_15$x, y = results_15$density))+
  scale_x_continuous(breaks = seq(from = 0, to = 25, by = 5)) +
  labs(title="Density Prediction k = 15")

plot3 = ggplot() +
  geom_histogram(aes(x = train, y = ..density..), bins = 15) +
  geom_line(aes(x = results_25$x, y = results_25$density)) +
  scale_x_continuous(breaks = seq(from = 0, to = 25, by = 5)) +
  labs(title="Density Prediction k = 25")

plot4 = ggplot() +
  geom_histogram(aes(x = train, y = ..density..), bins = 15) +
  geom_line(aes(x = results_30$x, y = results_30$density))+
  scale_x_continuous(breaks = seq(from = 0, to = 25, by = 5)) +
  labs(title="Density Prediction k = 30")

# arrange plots as a grid
grid.arrange(grobs = list(plot1, plot2, plot3, plot4), ncol=2)
```