STAT 6340 Mini Project 1 Report

Randy Suarez Rodes

2/3/2021

1 Answers

Question 1.a

We have fitted KNN for both training and testing set for values of k ranging between 1, 5, ...,196 and additional values of 200, 225, 250,...,400.

Question 1.b

We present the training and testing error rates against the number of nearest neighbors on @ref(fig:graph1). We can appreciate that for small values of k we have a low error rate on the training set in contrast we obtained high error rates for the testing set. As the value of K increases we can appreciate how the error of the training set increases while the testing set start to decrease. Both training and testing sets stabilizes and then the testing set increases again. This is consistent with class discussion since the K-NN algorithm flexibility decreases as the value of K increases, leading to an increase in bias. For small values of K this algorithm tends to overfit due to high variance. As K increases, flexibility decreases, implying that bias increases and variance decreases, therefore the test error rate graph starts to show an U shape.

Question 1.c

The optimal K for our experiment is 116, dealing a training error rate of 0.1181 and a testing error rate of 0.1155.

Question 1.d

We present the training data set along with the decision boundary obtained from the optimal value K(116) on @ref(fig:graph2). We appreciate how the decision boundary separates the training set into two well defined groups and the majority of the observations are reasonably classified correctly. There is some misclassification close to the decision boundary as expected, since the error rate for this value of K is around the 12%. So we can affirm that the decision boundary tends to predict the correct class with an 88% accuracy. Therefore we conclude that the decision boundary it is not as much sensible to new data.

Question 2.a

We have fitted KNN for the testing set for values of K = 50, 100, 200, 300 and 400. The error rate for every value of K is presented on Table 1.

Table 1: Test Error Rate

kval	Error
50	0.59
100	0.61
200	0.61

kval	Error
300	0.60
400	0.63

Question 2.b

A quick look at the confusion matrix immediately shows that the misclassification is high. In particular we can quickly find the accuracy (add diagonal divided by total) yielding 41% which is as good as random guess.

Table 2: Confusion Matrix

	0	1	2	3	4	5	6	7	8	9
0	3	1	1	0	1	0	0	0	0	1
1	0	1	0	1	0	0	0	0	0	0
2	1	1	2	2	3	2	1	1	1	2
3	0	0	0	2	0	0	0	1	0	0
4	0	1	1	5	7	1	5	2	2	4
5	0	0	1	1	0	2	0	1	1	0
6	1	2	0	0	1	2	6	0	0	0
7	0	0	0	0	0	0	0	1	1	0
8	1	1	0	0	0	1	0	1	14	1
9	0	1	0	0	0	1	0	0	0	3

Question 2.c

It is not a good idea to use KNN for image classification

KNN algorithm is that it uses all the features equally in computing for similarities

Accuracy depends on the quality of the data. With large data, the prediction stage might be slow. Sensitive to the scale of the data and irrelevant features. Require high memory – need to store all of the training data. Given that it stores all of the training, it can be computationally expensive.

The Nearest Neighbor Classifier may sometimes be a good choice in some settings (especially if the data is low-dimensional), but it is rarely appropriate for use in practical image classification settings. One problem is that images are high-dimensional objects (i.e. they often contain many pixels), and distances over high-dimensional spaces can be very counter-intuitive.

In particular, note that images that are nearby each other are much more a function of the general color distribution of the images, or the type of background rather than their semantic identity. For example, a dog can be seen very near a frog since both happen to be on white background. Ideally we would like images of all of the 10 classes to form their own clusters, so that images of the same class are nearby to each other regardless of irrelevant characteristics and variations (such as the background). However, to get this property we will have to go beyond raw pixels.

The advantages of the KNN algorithm are its simplicity and the ability to deal with multi classes. Nevertheless, a major disadvantage of the KNN algorithm is using all the features equally for similarity computing. This can lead to classification errors, especially when there is only a small subset of features that are useful for classification

Nearest neighbors algorithms are “instance-based,” which means that that save each training observation. They then make predictions for new observations by searching for the most similar training observations and pooling their values.

These algorithms are memory-intensive, perform poorly for high-dimensional data, and require a meaningful

distance function to calculate similarity. In practice, training regularized regression or tree ensembles are almost always better uses of your time.

2 Code

```
#Packages
library(ggplot2) #Used for graphics an visual representations
library(class) #Used for KNN models
library(data.table)
library(dplyr) #Data manipulation

rdseed=8467 #Seed to replicate results in case of a tie on KNN

#Helper function to calculate the classification error rate
classification_error_rate=function(ypred,ytrue)
{
  mean(ypred!=ytrue)
}


#Setting graphic options.
error_colors=c("blue","orange")
graph_legend=c("Training Set","Testing Set")


############################


#Experiment 1

#Values of K for experiment 1
kvals=c(seq(1,200,5),seq(200,400,25))


#Reading training and testing data sets
trn=read.csv("1-training_data.csv", stringsAsFactors = TRUE)
tst=read.csv("1-test_data.csv", stringsAsFactors = TRUE)

#Exploring the data set
str(trn)
summary(trn)

str(tst)
summary(tst)
#We can appreciate that the class labels are equally distributed
#in both training and testing sets

#Saving training and testing labels
trn_y=trn$y
tst_y=tst$y

#Dropping the classes to use only the predictors on the knn function.
trn$y=NULL
tst$y=NULL

#Helper function to find KNN predictions and find the classification error rate
#This function is declared for dplyr manipulation purposes.
```

```r
fitKNN_Error=function(testing_set,true_labels,i)
{
  set.seed(rdseed)
  classification_error_rate(knn(trn,testing_set,cl=trn_y,k=i),true_labels)
}


#Dataframe to track the error rates
Error_df=data.frame(kval=kvals)

#Question 1.a

#Fitting KNN for different values of K (training data)
Error_df=Error_df%>%group_by(kval)%>%mutate(trn_Error=fitKNN_Error(trn,trn_y,kval))

#Fitting KNN for different values of K (testing data)
Error_df=Error_df%>%group_by(kval)%>%mutate(tst_Error=fitKNN_Error(tst,tst_y,kval))


#Question 1.b

#Generating plot of training and testing error rates against k.

g=ggplot(data=Error_df, aes(x=kval, y=trn_Error))+
  geom_line(aes(y=trn_Error, color=graph_legend[1]), size=1,alpha = .8)+
  geom_point(color=error_colors[1], shape=19,alpha = .8)+
  geom_line(aes(y=tst_Error, color=graph_legend[2]), size=1,alpha=.6)+
  geom_point(x=Error_df$kval, y=Error_df$tst_Error, color=error_colors[2], shape=15,alpha=.6)+
  scale_color_manual("Legend",values=c("Training Set"=error_colors[1], "Testing Set"=error_colors[2]))+
  labs(title="Classification Error Rate", x="K", y="Error Rate")+
  theme(plot.title=element_text(hjust=0.5))

print(g)


#Question 1.c

#Finding the index of the optimal K, this is the index of the least test error rate
ind_optimalK=which.min(Error_df$tst_Error)

Error_df[ind_optimalK,] #The row of the optimal k contains the associated errors for training and testi

optimalK=Error_df$kval[ind_optimalK]


#Question 1.d

#Creating grid
x1=seq(min(trn[,1]),max(trn[,1]),length.out=100)
x2=seq(min(trn[,2]),max(trn[,2]),length.out=100)
grid <- expand.grid(x=x1, y=x2)

#Classifying the grid
```

```
bestK=knn(trn,grid,cl=trn_y,k=optimalK,prob = TRUE ) #Fitting with optimal K
prob = attr(bestK, "prob")
prob = ifelse(bestK=="yes", prob, 1-prob)

#Data Frame to generate the surface for the contour
df_contour=data.frame(x=grid[,1],y=grid[,2],z=prob)

#Data Frame to plot the training set
df_trn=data.frame(x1=trn[,1],x2=trn[,2],classes=trn_y)

#Plotting the training set
g2=ggplot()+geom_point(aes(x=x1,y=x2,color=classes),data=df_trn)

#Plotting the decision boundary
g2=g2+geom_contour(aes(x=x,y=y,z=z),
                data=df_contour,size=2,colour="black",breaks = 0.5)

print(g2)

##########################

#Experiment 2

##Preprocessing
library(keras)
cifar <- dataset_cifar10()
str(cifar)

x.train <- cifar$train$x
y.train <- cifar$train$y
x.test <- cifar$test$x
y.test <- cifar$test$y

# reshape the images as vectors (column-wise)
# (aka flatten or convert into wide format)
# (for row-wise reshaping, see ?array_reshape)
dim(x.train) <- c(nrow(x.train), 32*32*3) # 50000 x 3072
dim(x.test) <- c(nrow(x.test), 32*32*3) # 50000 x 3072

# rescale the x to lie between 0 and 1
x.train <- x.train/255
x.test <- x.test/255

# categorize the response
y.train <- as.factor(y.train)
y.test <- as.factor(y.test)

# randomly sample 1/100 of test data to reduce computing time

set.seed(2021)
id.test <- sample(1:10000, 100)

x.test <- x.test[id.test,]
```

```r
y.test <- y.test[id.test]

#Helper function to find KNN predictions and the classification error rate
#This function is declared for dplyr manipulation purposes.
fitKNN_Error_cifar=function(i)
{
  set.seed(rdseed)
  classification_error_rate(knn(x.train,x.test,cl=y.train,k=i),y.test)
}

#Values of K for experiment 2
kvals2=c(50, 100, 200, 300, 400)

#Dataframe to track the test error rates
Cifar_Error=data.frame(kval=kvals2)

##Experiment 2 questions

#Question 2.a

#Fitting KNN for different values of K
Cifar_Error=Cifar_Error%>%group_by(kval)%>%mutate(tst_Error=fitKNN_Error_cifar(kval))


#Question 2.b

ind_optimalK=which.min(Cifar_Error$tst_Error) #Finding optimal K

#Fitting KNN with the optimal K
set.seed(rdseed)
cifar_pred=knn(x.train,x.test,cl=y.train,k=ind_optimalK)

#Displaying confusion matrix
table(cifar_pred ,y.test,dnn=c("predicted","actual"))
```