

Assignment 4 – CS 545

John Whipple

October 15, 2009

Contents

1	Introduction	1
2	Logistic regression	2
2.1	Implementation	2
2.2	Results	4
3	Kth Nearest Neighbor	6
3.1	Implementation	6
3.2	Results	7
4	Parkinson’s Data	8
4.1	Data and Implementation	8
4.2	Results	12
4.3	Further Analysis	14
5	Car Quality Data	15
5.1	Data and Implentation	15
5.2	Results	16
6	Conclusion	16

1 Introduction

The purpose of this assignment was to introduce two new classification techniques and compare them to Quadratic Discriminant Analysis (QDA) and Linear Discriminant Analysis (LDA). The first new technique, logistic regression was different from QDA and LDA in that it calculated the posterior probability ($P(C=k|x)$) directly rather than deriving it using Bayes’ Rule. The second technique, kth nearest neighbor (KNN) simply classified a sample by finding the training sample that was “nearest” to it using the standard distance formula. The algorithm could find the first nearest neighbor or the first k nearest neighbors, thus its name. The assignment began by looking at logistic regression,

moved on to KNN, looked at the Parkinson's data and finally looked at a data set having to do with car quality.

2 Logistic regression

2.1 Implementation

The first part of the assignment asked us to compare logistic regression to LDA and QDA. This was accomplished by apply logistic regression techniques to data generated by the professor's implementation of assignment three. Two functions were created to make logistic regression easier to use:

```
makeLogReg <- function(Xtrain, Ttrain, K) {
  standardize <- makeStandardizeF(Xtrain)
  Xtrains <- standardize(Xtrain)
  TtrainI <- makeIndicatorVars(Ttrain)
  Xtrains <- cbind(1, Xtrains)
  D <- ncol(Xtrain)+1
  ## Initial beta
  beta <- matrix(0, D, K-1)
  beta <- matrix(beta)
  ## CA make a column vector
  res <- scg(beta, logregNegMaxLogL, logregGrad, Xtrains, TtrainI,
    fPrecision=0.00001, nIterations=1000000, ftracep=TRUE)
  beta <- matrix(res$x, D, K-1)
  return(list(beta=beta, standardize=standardize))
}

useLogReg <- function(Xtest, logReg) {
  Xtests <- logReg$standardize(Xtest)
  Xtests <- cbind(1, Xtests)
  p <- g(Xtests, logReg$beta)
  preds <- apply(p, 1, which.max)
  return(list(p=p, preds=preds, testXs=Xtests))
}
```

The functions greatly reduced the amount of code required for both the 1D and 2D cases:

```
allData <- read.table("1D.data", sep=",")
Ttrain <- allData[, 2, drop=FALSE]
Ttrain <- data.matrix(Ttrain)

Xtrain <- allData[, 1, drop=FALSE]
Xtrain <- data.matrix(Xtrain)
```

```

par(mfrow=c(1,3))
matplot(Xtrain, Ttrain, pch=1)

Xtest <- matrix(seq(0,4, len=100))

logReg <- makeLogReg(Xtrain, Ttrain, 3)

logRegResults <- useLogReg(Xtest, logReg)

matplot(logRegResults$testXs[,2], logRegResults$p, xlab="
x", ylab="P(C=k|x)")

matplot(logRegResults$testXs[,2], logRegResults$preds,
xlab="x", ylab="Predicted_class", pch=1)

allData <- read.table("assign3Solution/2DComplex.data",
sep=",")
classes <- c(rep(1,20), rep(2,20), rep(3,20))
Ttrain <- allData[,3, drop=FALSE]
Ttrain <- data.matrix(Ttrain)

Xtrain <- allData[,c(1,2), drop=FALSE]
Xtrain <- data.matrix(Xtrain)
N<-100
Xtest <- matrix(seq(0,10, len=N))
e <- expand.grid(Xtest, Xtest)
Xtest <- data.matrix(e)

par(mfrow=c(1,1))
plot(allData[,1], allData[,2], col=classes, pch=classes,
xlab="X", ylab="Y")
title("Samples")

logReg <- makeLogReg(Xtrain, Ttrain, 3)

logRegResults <- useLogReg(Xtest, logReg)

library(rgl)

z <- matrix(logRegResults$p[,1, drop=FALSE], nrow=N, ncol=
N)
persp3d(, , z, add=FALSE, axes=FALSE, col="gray")

```

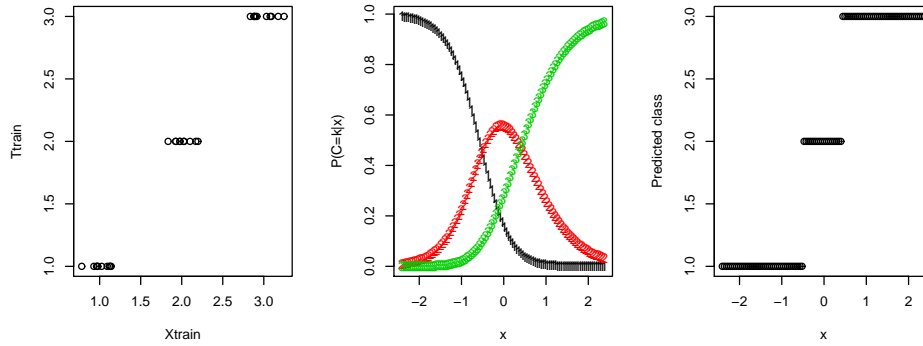


Figure 1: Logistic Regression 1D

```

z <- matrix(logRegResults$p[,2,drop=FALSE], nrow=N, ncol=
N)
material3d(col="red")
persp3d(,z,add=TRUE,axes=FALSE)

z <- matrix(logRegResults$p[,3,drop=FALSE], nrow=N, ncol=
N)
material3d(col="green")
persp3d(,z,add=TRUE,axes=FALSE)
snapshot3d(filename="part1.2D.complex.p.png", fmt="png")

z <- matrix(logRegResults$preds, nrow=N, ncol=N)
persp3d(,z,add=FALSE,axes=FALSE,col="purple")
snapshot3d(filename="part1.2D.complex.d.png", fmt="png")

```

Matplot was used to generate the graphs for the 1D section (Figure 1) while the RGL library and the persp3d function was used for the 2D section (Figures 2 and 3).

2.2 Results

Unlike QDA and LDA, logistic regression produced $P(C=k|x)$ directly. The 1D QDA and LDA $P(C=k|x)$ graphs tended to produce somewhat square functions that went from 1 to 0 almost instantaneously at the crossover points for each class. The logistic regression $P(C=k|x)$ graph was much smoother and was much easier to see where the probability for one class becomes greater than the next. This idea also extended into the 2D where the $P(C=k|x)$ graphs for QDA/LDA form plateaus with sharp cliffs from 1 to 0 at the class boundaries. Logistic regression on the other hand formed smoother transitions between classes. The

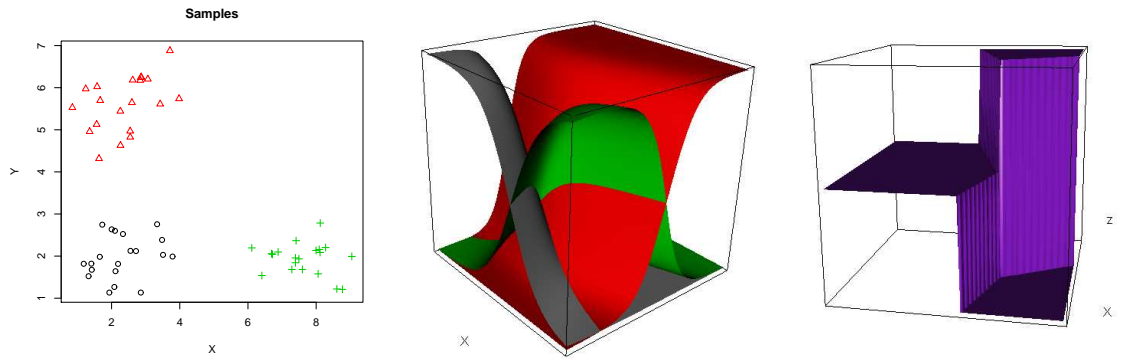


Figure 2: Logistic Regression 2D Simple

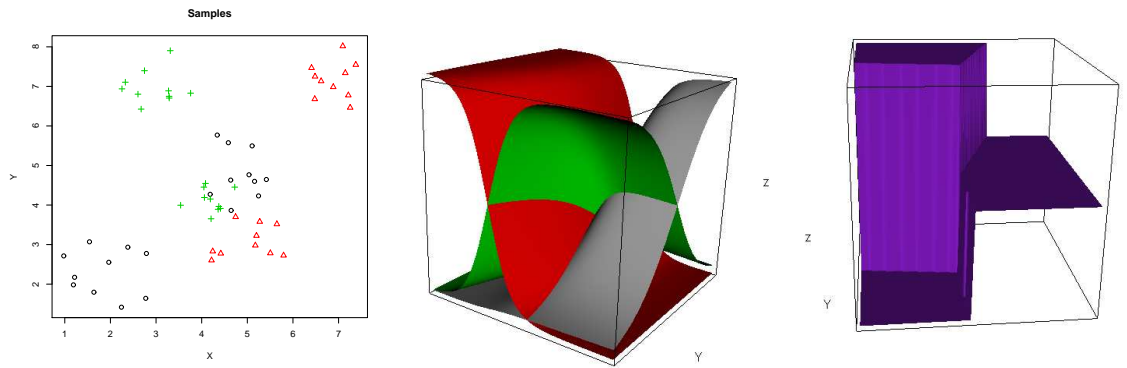


Figure 3: Logistic Regression 2D Complex

predicted class graphs look fairly similar for QDA, LDA and logistic regression.

3 Kth Nearest Neighbor

3.1 Implementation

An implementation of the KNN algorithm was given by the professor. This implementation used matrix operations to classify data based on the nearest neighbor. We were asked to implement the same algorithm, replacing matrix operations with for loops:

```
useForLoopKNN <- function(knn,Xtest,
  nNeighbors=1) {
  Xtest <- knn$standardizeF(Xtest)
  indices <- matrix(0, nrow(Xtest), nrow(knn$
    Xtrain))

  distanceSquared <- matrix(0, 1, nrow(knn$
    Xtrain))
  for(i in 1:nrow(Xtest)) {
    for(j in 1:nrow(knn$Xtrain)) {
      rowSum <- 0
      for(a in 1:ncol(Xtest)) {
        rowSum <- rowSum + (knn$Xtrain[j,a] -
          Xtest[i,a])^2
      }
      distanceSquared[1,j] <- rowSum
    }

    indices[i,] <- order(distanceSquared[1,])
  }

  predictions <- NULL
  for (k in nNeighbors)
    predictions <- cbind(predictions,
      mostCommon(matrix(knn
        $Ttrain[c(indices
          [,1:k])],nrow(
            indices),k), knn$
          classes))

  predictions
}

allData <- cbind(rnorm(1100, 3, .1), rnorm
  (1100, 5, .2), 1)
```

user	system	elapsed
0.86	0.01	0.87

Testing data 47.72727 percent correct using "useKNN"

user	system	elapsed
30.85	0.03	32.17

Testing data 47.72727 percent correct using "useForLoopKNN"

Table 1: KNN Timing Results

```

allData <- rbind(allData,
                 cbind(rnorm(1100, 3, .1), rnorm
                       (1100, 5, .2), 2)
                 )

trainingSplit <- .5

prepare trainXs and trainT

knn <- makeKNN(trainXs, trainT)

print(system.time(testTPredicted <- useKNN(knn
, testXs, nNeighbors=3)))
pCorrectTest <- sum(abs(testTPredicted - testT
) < 0.5) / length(testT) * 100.0
cat("Testing data",pCorrectTest,"percent
correct using \"useKNN\"\\n")

print(system.time(testTPredicted <-
useForLoopKNN(knn, testXs, nNeighbors=3)))
pCorrectTest <- sum(abs(testTPredicted - testT
) < 0.5) / length(testT) * 100.0
cat("Testing data",pCorrectTest,"percent
correct using \"useForLoopKNN\"\\n")

```

3.2 Results

The for loop algorithm running time on 2200 randomly generated samples took 37 times as long to complete compared to the version using matrix operations. Not only was the code easier to read, the performance advantage of matrix operations in R was obvious.

4 Parkinson's Data

4.1 Data and Implementation

The third part of the assignment was to compare the classification developed so far on the Parkinson's data used previously in class. The data consisted of 47 voice print samples of a healthy patient and 147 samples of a patient with Parkinson's.

```
originalData <- read.table("parkinsons.data",
  header=TRUE, sep=",")
accuracies<-c()
for(trainf in c(.5, .6, .7, .8, .95)) {
  for(i in 1:100) {
    data <- originalData
    data <- as.matrix(data[,-1]) ## remove
      name and make numeric
    data <- data[sample(nrow(data)),] ##
      randomly rearrange data

    status <- data["status"]
    data <- data[, -which(colnames(data)=="
      status")] ## remove status column
      from data

    ### ...data preparation taken from
      professor's code...

    ### Logistic Regression
    logReg <- makeLogReg(Xtrain, Ttrain, 2)

    logRegResults <- useLogReg(Xtrain, logReg)
    Xtrains <- logRegResults$testXs
    accsTrain <- sum(logRegResults$preds==
      Ttrain)/length(Ttrain)

    logRegResults <- useLogReg(Xtest, logReg)
    Xttests <- logRegResults$testXs
    accsTest <- sum(logRegResults$preds==Ttest
      )/length(Ttest)

    results[["scg"]] <- c(results[["scg"]],
      list(accuracies=c(accsTrain, accsTest))
    )
  }
}
```

```

resultsTable <- cbind(results$scg$
  accuracies)

colnames(resultsTable) <- c("SCG")
rownames(resultsTable) <- c("Train","Test"
  )

innerSum[1,1] <- innerSum[1,1] + accsTrain
  *100
innerSum[1,2] <- innerSum[1,2] + accsTest*
  100

Xtrains <- Xtrains[,-1]
Xtests <- Xtests[,-1]

### QDA
mean1 <- colMeans(Xtrains[Ttrain==1,])
cov1 <- cov(Xtrains[Ttrain==1,])
mean2 <- colMeans(Xtrains[Ttrain==2,])
cov2 <- cov(Xtrains[Ttrain==2,])

qdadisc1 <- makeQDADiscF(mean1,cov1,
  nHealthy/(nHealthy+nParks))
qdadisc2 <- makeQDADiscF(mean2,cov2,nParks
  /(nHealthy+nParks))

TtrainPredicted <- apply(cbind(qdadisc1(
  Xtrains),qdadisc2(Xtrains)),1,which.max
  )
TtestPredicted <- apply(cbind(qdadisc1(
  Xtests),qdadisc2(Xtests)),1,which.max)

pCorrectTrain <- sum(abs(TtrainPredicted -
  Ttrain) < 0.5) / length(Ttrain) *
  100.0
pCorrectTest <- sum(abs(TtestPredicted -
  Ttest) < 0.5) / length(Ttest) * 100.0

innerSum[2,1] <- innerSum[2,1] +
  pCorrectTrain
innerSum[2,2] <- innerSum[2,2] +
  pCorrectTest

```

```

### LDA
N <- nHealthy + nParks
covinnerSum <- cov1 * nHealthy/N + cov2 *
  nParks/N

ldadisc1 <- makeLDADiscF(mean1, covinnerSum
  , nHealthy/(nHealthy+nParks))
ldadisc2 <- makeLDADiscF(mean2, covinnerSum
  , nParks/(nHealthy+nParks))

TtrainPredicted <- apply(cbind(ldadisc1(
  Xtrains), ldadisc2(Xtrains)), 1, which.max
)
TtestPredicted <- apply(cbind(ldadisc1(
  Xtests), ldadisc2(Xtests)), 1, which.max)

pCorrectTrain <- sum(abs(TtrainPredicted -
  Ttrain) < 0.5) / length(Ttrain) *
  100.0
pCorrectTest <- sum(abs(TtestPredicted -
  Ttest) < 0.5) / length(Ttest) * 100.0
#cat("Training data", pCorrectTrain, "
  percent correct\n")
#cat("Testing data", pCorrectTest, "percent
  correct\n")

innerSum[3,1] <- innerSum[3,1] +
  pCorrectTrain
innerSum[3,2] <- innerSum[3,2] +
  pCorrectTest

### KNN 1
knn <- makeKNN(Xtrains, Ttrain)

testTPredicted <- useKNN(knn, Xtests,
  nNeighbors=1)
cbind(Ttest, TtestPredicted)
pCorrectTest <- sum(abs(TtestPredicted -
  Ttest) < 0.5) / length(Ttest) * 100.0
#cat("Testing data", pCorrectTest, "test
  percent correct using \"useKNN 1\"\n")

innerSum[4,2] <- innerSum[4,2] +
  pCorrectTest

```

```

trainTPredicted <- useKNN(knn, Xtrains,
  nNeighbors=1)
cbind(Ttrain, TtrainPredicted)
pCorrectTrain <- sum(abs(TtrainPredicted -
  Ttrain) < 0.5) / length(Ttrain) *
  100.0
#cat("Training data", pCorrectTrain, "train
  percent correct using \"useKNN 1\"\\n")

innerSum[4,1] <- innerSum[4,1] +
  pCorrectTrain

### KNN 10

testTPredicted <- useKNN(knn, Xtests,
  nNeighbors=10)
cbind(Ttest, TtestPredicted)
pCorrectTest <- sum(abs(TtestPredicted -
  Ttest) < 0.5) / length(Ttest) * 100.0
#cat("Testing data", pCorrectTest, "test
  percent correct using \"useKNN 10\"\\n")
innerSum[5,2] <- innerSum[5,2] +
  pCorrectTest

trainTPredicted <- useKNN(knn, Xtrains,
  nNeighbors=10)
cbind(Ttrain, TtrainPredicted)
pCorrectTest <- sum(abs(TtrainPredicted -
  Ttrain) < 0.5) / length(Ttrain) * 100.0
#cat("Training data", pCorrectTest, "train
  percent correct using \"useKNN 10\"\\n")
innerSum[5,1] <- innerSum[5,1] +
  pCorrectTrain
}
innerSum <- innerSum / 100
print(trainf)
innerSum <- t(innerSum)
colnames(innerSum) <- c("SCG", "QDA", "LDA",
  "KNN1", "KNN10")
rownames(innerSum) <- c("Train", "Test")

```

Train %		SCG	QDA	LDA	KNN1	KNN10
50	Train	0.8453608	1.0000000	0.8865979	1.0000000	1.0000000
50	Test	0.8061224	0.7755102	0.8367347	0.9387755	0.8877551
60	Train	0.8448276	1.0000000	0.9051724	1.0000000	1.0000000
60	Test	0.8734177	0.8607595	0.8354430	0.8734177	0.8227848
70	Train	0.8518519	1.0000000	0.9037037	1.0000000	1.0000000
70	Test	0.8333333	0.8166667	0.8833333	0.9166667	0.8166667
80	Train	0.8580645	0.9806452	0.8774194	1.000	1.000
80	Test	0.7750000	0.8000000	0.8250000	0.875	0.875
95	Train	0.8586957	0.9728261	0.9130435	1.0000000	1
95	Test	0.8181818	0.9090909	1.0000000	0.9090909	1

Table 2: Trainin Fraction Accuracy

```

print(innerSum)
  accuracies <- rbind(accuracies, innerSum)
}
print(accuracies)
trainAccs <- accuracies[rownames(accuracies)==
  "Train",]
testAccs <- accuracies[rownames(accuracies)==
  "Test",]

par(mfrow=c(1,2))
colors<-c(1,2,3,4,5)

matplot(c(.5, .6, .7, .8, .95), trainAccs,
  type="l", xlab="Training fraction", ylab="
  Prediction Accuracy", lty=1, col=colors)
title("Train Accuracy")

matplot(c(.5, .6, .7, .8, .95), testAccs, type
  ="l", xlab="Training fraction", ylab="
  Prediction Accuracy", lty=1, col=colors)
title("Test Accuracy")

legend("topleft", c("SCG", "QDA", "LDA", "KNN1
  ", "KNN10"), lty=1, col=colors);

```

4.2 Results

The average of 100 runs for training splits ranging from 50% to 95% for each classification technique was collected in a table and then graphed.

Figure 4 shows that the training fraction did not seem to have much effect

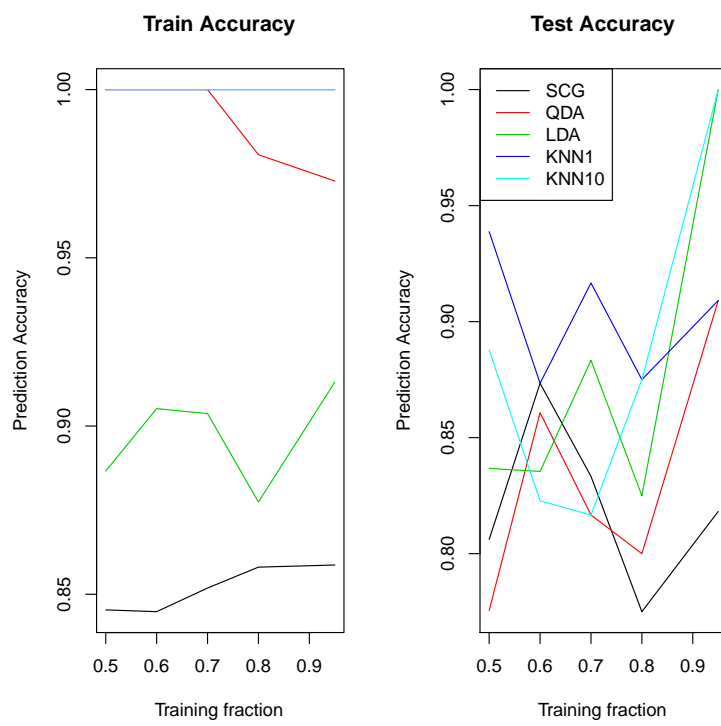


Figure 4: Training Fraction Accuracy

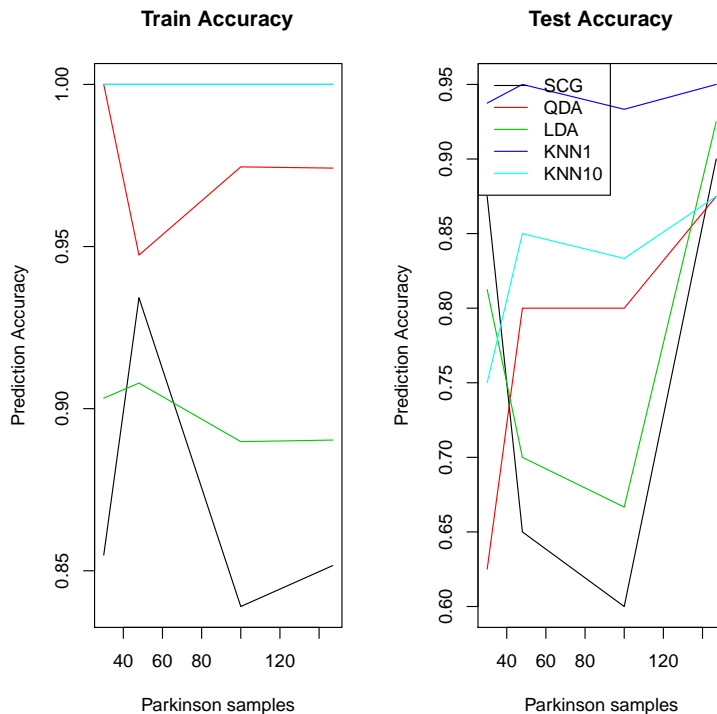


Figure 5: Number of Parkinson's Samples Accuracy

when the classifiers were used on the training data itself; the plots tended to be rather flat for the entire range. When the test data was classified however, more interesting things began to happen. The accuracies of the classifiers hovered around 85 to 90% until the training fraction reach 80%. KNN1 remained flat while KNN10 became very accurate. LDA became equally accurate while QDA became only mildly more accurate at high training fractions. Labeled “SCG” (aka scaled conjugate gradient), logistic regression actually became less accurate as the training fraction increased.

4.3 Further Analysis

One question I had was if the number of samples with Parkinson's relative to samples without Parkinson's had any effect on classification accuracy. I modified the code to vary the number of samples with Parkinson's from 30 to 147. The training fraction was set to 80% for all configurations.

Figure 5 shows the training data again seemed to produce relatively flat accuracy curves when classified. The testing data accuracy improved as the number of Parkinson's samples in the training set was increased. This contra-

dicted my hypothesis that the data was skewed because of the larger number of Parkinson's samples.

5 Car Quality Data

5.1 Data and Implementation

The final source of data consisted of 1,728 samples used to classify the “Overall Quality” of a car based on six attributes: buying price, maintenance price, number of doors, capacity and size of the trunk. This data was also used in the previous assignment so the task of preparing the data for the classification algorithms was already complete. I modified the code from part three to support this data and converted the text-based ranges for each attribute into numerical ranges.

```
allData <- read.table("car.data", sep=",")

data2 <- c()
for(i in 1:ncol(allData)) {
  data2 <- cbind(data2, apply(allData[,i,drop=
    FALSE], 1, function(x) {
      result <- x
      if(x=="low" || x=="small" ||
        x=="unacc") {
        result <- 1
      }
      if(x=="med" || x=="acc") {
        result <- 2
      }
      if(x=="high" || x=="big" ||
        x=="good") {
        result <- 3
      }
      if(x=="v-high" || x=="vhigh"
        || x=="vgood") {
        result <- 4
      }
      if(x=="5-more" || x=="5more"
        || x=="more") {
        result <- 5
      }
      result
    })
}
```

	SCG	QDA	LDA	KNN1	KNN10
Train	54.12373	70.20912	62.82200	76.38784	76.38784
Test	53.97110	69.63584	62.33237	60.58382	69.97977

Table 3: Car Quality Accuracy

```

mode(data2) <- "numeric"
allData <- data2

colnames(allData) <- c("buying", "maint", "
    doors", "persons", "lug_boot", "safety", "
    car quality")

```

5.2 Results

The results of running the five different classifiers produced some interesting results seen in table 3.

Logistic regression did the worst of all the classifiers while KNN10 did the best. I found it surprising that even though k nearest neighbors is such a simple idea, it is still quite powerful. QDA is the second best classifier followed by LDA and KNN1. Another interesting question would be if these results would extend to very large data sets. The simplicity of the idea does not necessarily mean the computations associated with the classifier are simple.

6 Conclusion

Comparing LDA/QDA with logistic regression the biggest advantage seems to be the way logistic regression handles outliers. While the discriminate functions of QDA and LDA might incorrectly include points with a high deviation from the rest of the data, the functions controlling logistic regression end up including these outliers. Kth nearest neighbor also proved to be a powerful classification technique. In the future I would be interested to discover the limits of KNN pertaining to computation time as well as the types of data it is able to classify.