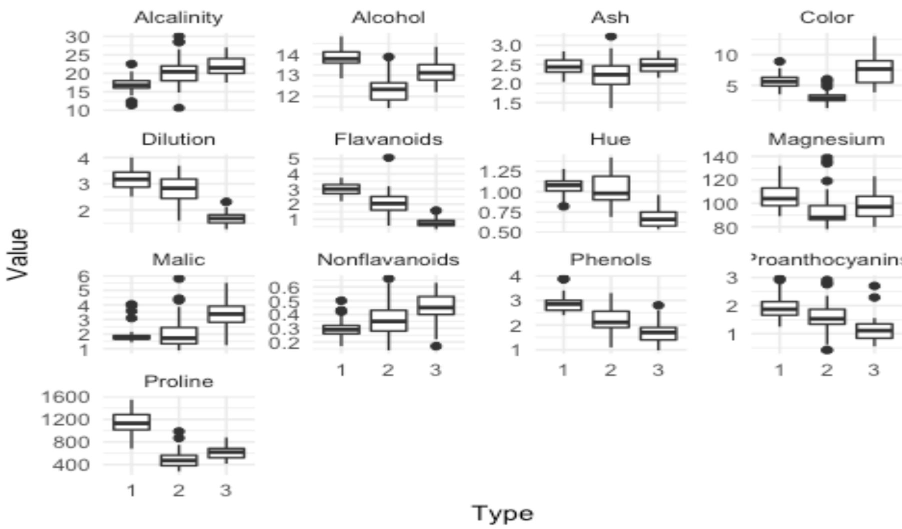
dehomework2

Qifan Wang

2/15/2020

Wine data-LDA, QDA and NB

```
# Data exploration: wine and other features
wine_train=read.csv("wine_train.csv")
wine_test=read.csv("wine_test.csv")

library(dplyr)

library(tidyr)
library(corrplot)

library(ggplot2)
head(wine_train)

##   Type Alcohol Malic  Ash Alcalinity Magnesium Phenols Flavanoids Nonflavanoids
## 1    1   13.77  1.90 2.68       17.1       115    3.00       2.79          0.39
## 2    1   13.94  1.73 2.27       17.4       108    2.88       3.54          0.32
## 3    1   13.75  1.73 2.41       16.0        89    2.60       2.76          0.29
## 4    1   12.85  1.60 2.52       17.8        95    2.48       2.37          0.26
## 5    1   13.63  1.81 2.70       17.2       112    2.85       2.91          0.30
## 6    1   13.58  1.66 2.36       19.1       106    2.86       3.19          0.22
##   Proanthocyanins Color  Hue Dilution Proline
## 1            1.68  6.30 1.13     2.93    1375
## 2            2.08  8.90 1.12     3.10    1260
## 3            1.81  5.60 1.15     2.90    1320
## 4            1.46  3.93 1.09     3.63    1015
## 5            1.46  7.30 1.28     2.88    1310
## 6            1.95  6.90 1.09     2.88    1515

# Wide data
wine_train$Type=factor(wine_train$Type)
wine_test$Type=factor(wine_test$Type)
dat_exp=gather(wine_train, key="Variable", value="Value",-c("Type"))
ggplot(dat_exp) + geom_boxplot(aes(x = Type, y = Value)) + facet_wrap(.~Variable, scale
s = "free_y") + theme_minimal()
```



Data Exploration

Based on the boxplots above, all features differ more or less across different wine types. Among them, Alchohol, Color, Dilution, Flavanoids, Alcalinity, Nonflavanoids, Phenois, Proline and Proanthocyanins are especially potential features that can help distinguish wine of different types.

LDA, QDA and Naive Bayes

```
# lda model
wine_lda=lda(Type~.,data=wine_train)
wine_lda_train_predict=predict(wine_lda, wine_train)$class
wine_lda_test_predict=predict(wine_lda, wine_test)$class
train_error_lda=mean(wine_lda_train_predict!=wine_train$Type)
test_error_lda=mean(wine_lda_test_predict!=wine_test$Type)

# test error and training error for lda
test_error_lda

## [1] 0.01818182
```

The testing error of LDA model is 0.018.

```
# qda model
wine_qda=qda(Type~.,data=wine_train)
wine_qda_train_predict=predict(wine_qda, wine_train)$class
wine_qda_test_predict=predict(wine_qda, wine_test)$class
train_error_qda=mean(wine_qda_train_predict!=wine_train$Type)
test_error_qda=mean(wine_qda_test_predict!=wine_test$Type)

# test error and training error for qda
test_error_qda

## [1] 0.03636364
```

The testing error for QDA model is 0.036.

```
library(e1071)
library(foreign)
```

```r
library(boot)
library(class)
library(ISLR)

# Nb classifier
wine_nb=naiveBayes(Type~.,data=wine_train)
predict_nb_train=predict(wine_nb, wine_train)
predict_nb_test=predict(wine_nb, wine_test)
train_error_nb=mean(predict_nb_train!=wine_train$Type)
test_error_nb=mean(predict_nb_test!=wine_test$Type)

# error for nb classifier
train_error_nb

## [1] 0.01626016

test_error_nb

## [1] 0.03636364
```

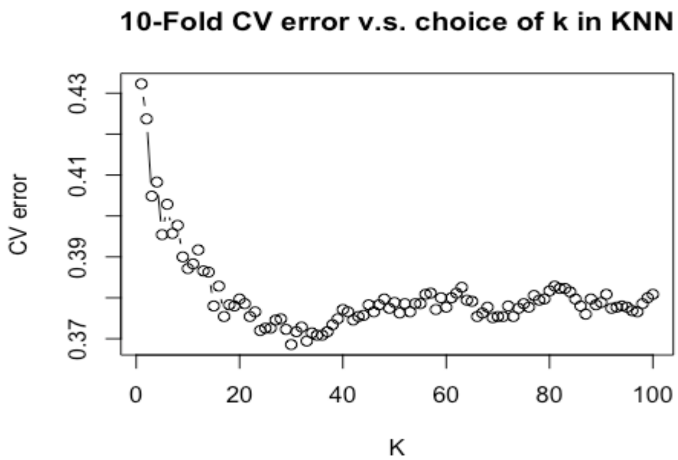
The testing error for NbClassifier is 0.036.

KNN and Cross Validation

Use cross-validation to select the best k and use the test data to evaluate the performance of the selected model. Show the training, cross-validation and test errors for each choice of k and report your findings.

```r
theft_train=read.csv("theft_train.csv")
theft_test=read.csv("theft_test.csv")
theft_test$theft=factor(theft_test$theft)
theft_train$theft=factor(theft_train$theft)

# Leave one out CV function
Kfold_cv<-function(k_fold,knn,train,train_label){
  fold_size=floor(nrow(train)/k_fold)
  cv_error=rep(0,k_fold)
  for (i in 1:k_fold){
    if (i!=k_fold){
      id=((i-1)*fold_size+1):(i*fold_size)
    }
    else {id=((i-1)*fold_size+1):nrow(train)
      }
    cv_train=train[-id,]
    cv_test=train[id,]

    mean_cv_train=colMeans(cv_train)
    sd_cv_train=apply(cv_train,2,sd)

    cv_train=scale(cv_train, center=mean_cv_train, scale=sd_cv_train)
    cv_test=scale(cv_test, center=mean_cv_train, scale=sd_cv_train)

    pred_knn=knn(cv_train, cv_test, train_label[-id],k=knn)

    cv_error[i]=mean(pred_knn!=train_label[id])

  }
   return(mean(cv_error))
}
```

```
set.seed(2020)
theft_train_X=theft_train[,-3]
theft_train_label=theft_train$theft
theft_test_X=theft_test[,-3]
theft_test_label=theft_test$theft
# 10_fold
k_fold=10
knn=1:100
cv.errors=rep(1,100)
train.errors=rep(1,100)
test.errors=rep(1,100)

train_mean=colMeans(theft_train_X)
train_sd=apply(theft_train_X,2,sd)

theft_train_scaled=scale(theft_train_X,center=train_mean, scale=train_sd)
theft_test_scaled=scale(theft_test_X,center =train_mean, scale=train_sd)

set.seed(2020)
for (i in 1:100){

  cv.errors[i]=Kfold_cv(k_fold, knn=knn[i], train=theft_train_X, train_label = theft_tr
ain_label)
}

set.seed(2020)
for (i in 1:100){

  knn_model1=knn(theft_train_scaled, theft_train_scaled, theft_train_label, knn[i])
  train.errors[i]=mean(knn_model1!=theft_train_label)

  knn_model2=knn(theft_train_scaled, theft_test_scaled, theft_train_label, knn[i])
  test.errors[i]=mean(knn_model2!=theft_test[,3])

}

set.seed(2020)
# which minimizes the cv error
min(cv.errors)

## [1] 0.3685714

which(cv.errors==min(cv.errors))

## [1] 30

# which minimized the test error
min(test.errors)

## [1] 0.368

which(test.errors==min(test.errors))

## [1] 26

plot(cv.errors~knn,type='b',main = '10-Fold CV error v.s. choice of k in KNN',xlab = 'K
',ylab = 'CV error')
```

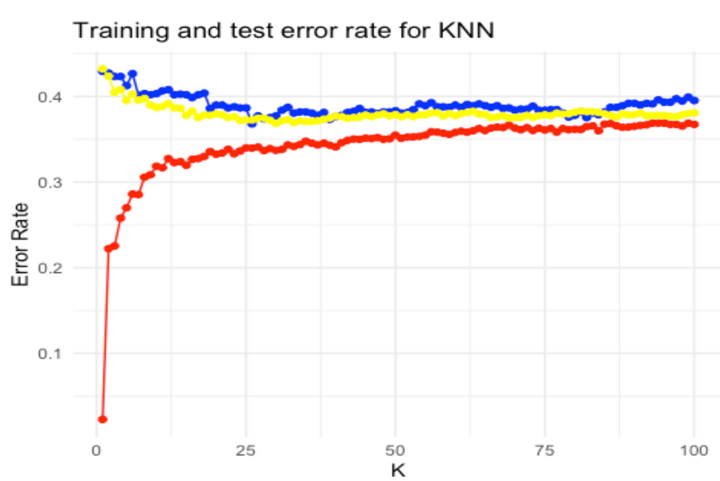



```
#test error at k=30
results=knn(theft_train_scaled, theft_test_scaled, theft_train_label, 30)
mean(results!=theft_test_label)

## [1] 0.378
```

The plot shows the CV error based on the 10-fold cross validation and lowest CV error (0.368) is obtained at k=30.

Fitting the best model selected by 10-fold cv on the test data, we get an error rate of 37.8%.

```
k=c(1:100)
errors2 = data.frame(train.errors, test.errors, cv.errors, k)
ggplot(errors2, aes(x = k)) +
  geom_line(aes(y = train.errors), col = "red") + geom_point(aes(y = train.errors), col
 = "red") +
  geom_line(aes(y = test.errors), col = "blue") + geom_point(aes(y = test.errors), col
= "blue") +
  geom_line(aes(y = cv.errors), col = "yellow") + geom_point(aes(y = cv.errors), col =
"yellow") +
  ylab("Error Rate") + xlab("K") + ggtitle("Training and test error rate for KNN") + th
eme_minimal()
```



The red line is the training error line, and it is always the lowest for whichever k we chose and this is expected because we are applying model to the in-sample data and thus the error rate is underestimated.

The test errors are mostly slightly higer than cv errors, but very close to each other. The k which minimizes cv error does not minimize test error. However, since we already used the cv estimation of error rate to select the model, it tends to overestimate the performance of the model. It would be therefore, use the test error (which is 0.378) at k=30 to report the performance.

CV for glm

```
weekly=Weekly
head(Weekly)

##   Year   Lag1   Lag2   Lag3   Lag4   Lag5    Volume  Today Direction
## 1 1990  0.816  1.572 -3.936 -0.229 -3.484 0.1549760 -0.270      Down
## 2 1990 -0.270  0.816  1.572 -3.936 -0.229 0.1485740 -2.576      Down
## 3 1990 -2.576 -0.270  0.816  1.572 -3.936 0.1598375  3.514        Up
## 4 1990  3.514 -2.576 -0.270  0.816  1.572 0.1616300  0.712        Up
## 5 1990  0.712  3.514 -2.576 -0.270  0.816 0.1537280  1.178        Up
## 6 1990  1.178  0.712  3.514 -2.576 -0.270 0.1544440 -1.372      Down

colnames(weekly)

## [1] "Year"      "Lag1"      "Lag2"      "Lag3"      "Lag4"      "Lag5"
## [7] "Volume"    "Today"     "Direction"

weekly$Direction=as.character(weekly$Direction)
weekly$Direction[weekly$Direction=="Up"]="1"
weekly$Direction[weekly$Direction=="Down"]="0"
head(weekly)

##   Year   Lag1   Lag2   Lag3   Lag4   Lag5    Volume  Today Direction
## 1 1990  0.816  1.572 -3.936 -0.229 -3.484 0.1549760 -0.270         0
## 2 1990 -0.270  0.816  1.572 -3.936 -0.229 0.1485740 -2.576         0
## 3 1990 -2.576 -0.270  0.816  1.572 -3.936 0.1598375  3.514         1
## 4 1990  3.514 -2.576 -0.270  0.816  1.572 0.1616300  0.712         1
## 5 1990  0.712  3.514 -2.576 -0.270  0.816 0.1537280  1.178         1
## 6 1990  1.178  0.712  3.514 -2.576 -0.270 0.1544440 -1.372         0

weekly$Direction=factor(weekly$Direction)
```

Logit regression: Dir~lag1+lag2

```
glm.a=glm(Direction~Lag1+Lag2, family = binomial,data=weekly)
summary(glm.a)

##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2, family = binomial, data = weekly)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.623  -1.261   1.001   1.083   1.506
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.22122    0.06147   3.599 0.000319 ***
```

```
## Lag1        -0.03872    0.02622  -1.477 0.139672
## Lag2         0.06025    0.02655   2.270 0.023232 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1496.2  on 1088  degrees of freedom
## Residual deviance: 1488.2  on 1086  degrees of freedom
## AIC: 1494.2
##
## Number of Fisher Scoring iterations: 4
```

The result shows that the second lag has a significant positive effect on the direction of the stock return, while the first lag does not: The higer the percentage return for 2 weeks previous, the more likely the return on this week is positive.

The result also shows the model is probably not a good choice. The residual deviance can be regarded as a measure of goodness-of-nit, but it is highly significant (statistic=1488, df=1086), indicating that the model does not fit well.

Logit Regression excluding the first observation

```
weekly.1=weekly[-1,]
glm.b=glm(Direction~Lag1+Lag2, family = binomial,data=weekly.1)
summary(glm.b)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2, family = binomial, data = weekly.1)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6258  -1.2617   0.9999   1.0819   1.5071
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.22324    0.06150   3.630 0.000283 ***
## Lag1        -0.03843    0.02622  -1.466 0.142683
## Lag2         0.06085    0.02656   2.291 0.021971 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1494.6  on 1087  degrees of freedom
## Residual deviance: 1486.5  on 1085  degrees of freedom
## AIC: 1492.5
##
## Number of Fisher Scoring iterations: 4
```

The result roughly remains the same. The lag 2 is still significant, but with estimated coefficient increased from 0.06025 to 0.06085. The lag 1 still remains insignificant.

The null deviance and residual deviance all underwent a tiny decrease (by 2) and AIC as well. Generally speaking, nothing changed significantly in this model that excluded the first observation.

Predict the first observation

```
inv.logit(predict(glm.b, weekly[1,]))

##         1
## 0.5713923

weekly$Direction[1]

## [1] 0
## Levels: 0 1
```

When the threhold is 0.5, the prediction on the first observation is wrong. (The prediction is "up" while the real label is "down")

```
set.seed(2020)
Loocv.errors=rep(0,nrow(weekly))
mse=rep(0,nrow(weekly))
for (i in (1:nrow(weekly))){
  train=weekly[-i,]
  test=weekly[i,]
  glm.cv=glm(Direction~Lag1+Lag2, data=train,family=binomial)
  predict=inv.logit(predict(glm.cv,test))
  mse[i]=(predict-as.numeric(as.character(test$Direction)))^2
  if (predict>0.5) {predict=1}
  else {predict=0}
  Loocv.errors[i]=(predict!=test$Direction)
}

mean(Loocv.errors)

## [1] 0.4499541

in_sample=inv.logit(predict(glm.a,weekly))
for (i in (1:length(in_sample))){
  if (in_sample[i]>0.5) {in_sample[i]=1}
  else {in_sample[i]=0}
}
mean(in_sample!=weekly$Direction)

## [1] 0.4444444
```

The cv estimation of error rate is about 0.45 while the estimated error rate whithout cv (directly apply the model to the in-sample data and do the prediction) is 0.44. We see that 0.44 slightly overestimates the performance of the model and cv error rate should provide a more accurate evaluation of the model performance.

The error rate is pretty high, close to 0.5, indicating the model is not an optimal one and it needs improvement, such as adding more lags, polynomial terms etc.