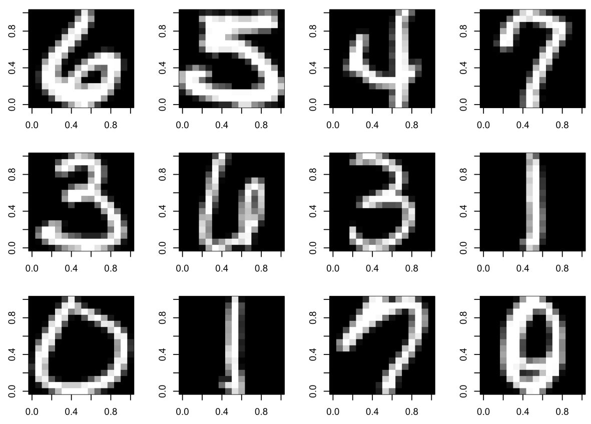
ZIP Practical Assignment

Krishna Kalyan

Introduction

We have normalized handwritten digits, automatically scanned from envelopes by the U.S. Postal Service in 16 x 16 grayscale images. The first columns has taget followed by the predictors. The purpose of this assignment is to build a classification function using the training data and evaluate its quality in the test data. Plotting first twelve samples from our dataset below:

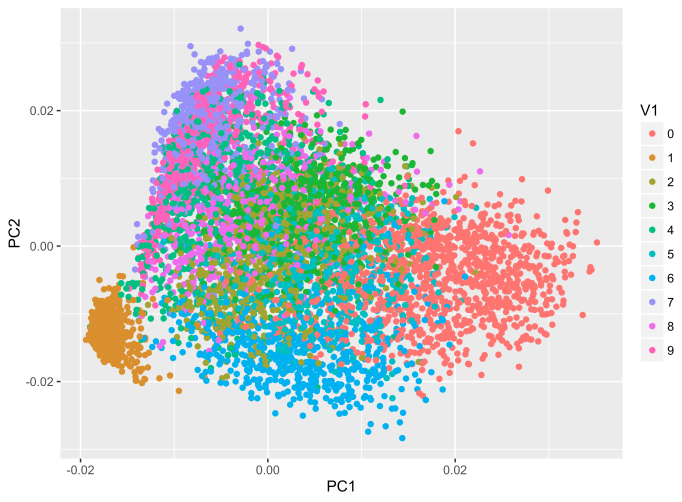


Data Analysis

Our dataset contains 7291 training and 2007 observation. We have 257 attributes or predictors. We center only the predictors followed by removing the intercept term from our linear model. It does not make sense to scale our targets as they are dummy variables. Scaling our data also does not make sense as our data is on the same scale.

PCA

A quick look at data when reduced to two dimensions using the Principal Component Analysis, We see that some of the digits, for example `0` are clustered together while other are dispersed across the axis. The first two components explain `23` percent of the variance in our data.



Experiemnts

For evaluating our experiments we use the following evaluation metics like `error` , `accuracy` to test our model. We will also evaluate goodness of fit by comparing `R-Square` values.

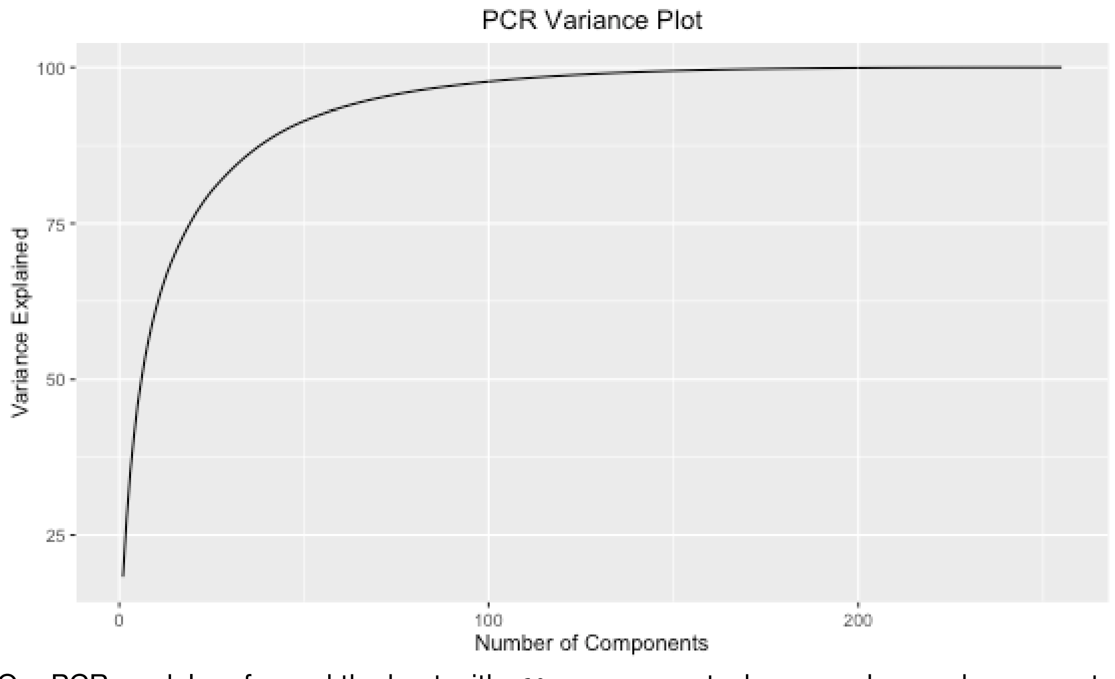
We use `5,30,70` percent of our training data set to train our `Multivariate Regression` model and predict on the whole testing set.

Sample Traning Percentage	R-Square	R-Square LOO	Error
0.05	0.79	-67.41	0.81
0.30	0.69	0.42	0.36
0.70	0.67	0.49	0.31

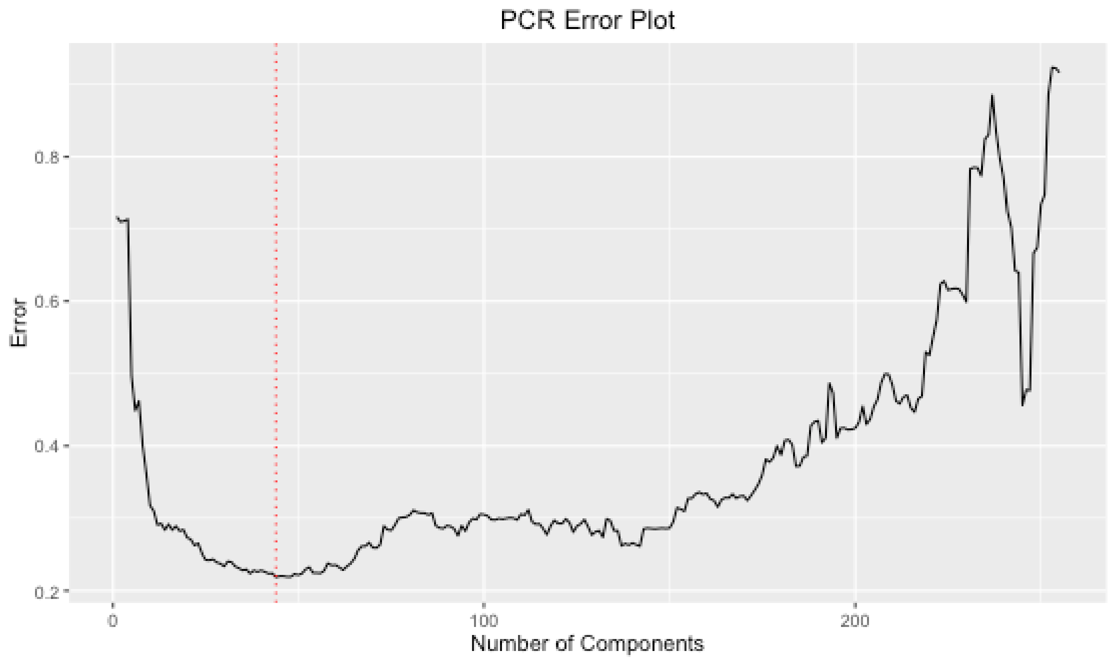
We see that our test error decreases as we increase the size of our training dataset, which is expected. Similar trend can be obtained when we train `Principal Components Regression Model` on the same training splits with all components.

Sample Traning Percentage	Accuracy	Error
0.05	0.79	0.23
0.30	0.80	0.19
0.70	0.81	0.18

We wanted to observe the variance in the dataset with respect to the number of components which can be seen from the plot below. We can conclude that most of the latent interactions in our data can be captured by first `100` components in our `PCR Model`.



Our PCR model performed the best with `60` compoment where we observed our error to be `23` percent. This can be observed in our plot below. The `60th` component explains `93` percent of our variation in data. We also observe that after `60th` component we start overfitting and our error starts increasing.



Conclusion

Multivariate regression did not do too well when we had a small sample. Its performance increase as we increased the number of samples. `PCR` had better performance when compared to multivariate regression. This was mostly because PCR captured latent interactions that multivariate regression could not capture. `PCR` performed quite well with small subset of data with `365` observation, giving us an accuracy of more than `70` percent. On the other hand neural network did quite badly with a small sample. It gave me an accuracy of only `56` percent. On Increasing the sample size, we could achieve an accuracy of more than `87` percent without tuning. After tuning, our accuracy went up to `93` percent (Tuning parameters : epochs, Introducing Convolutions and adding decay).

Code

```r
rm(list=ls())
require(caTools)
library(caret)
library(CatEncoders)
require(miscTools)
library(pls)
setwd('/Users/krishna/MIRI/MVA/Assignment1')
source('utils.R')
set.seed(123)

# 1 Read Data with 5 percent
train = read.delim('../ZIP data/zip_train.dat', header = F, sep=" ")
test = read.delim('../ZIP data/zip_test.dat', header = F, sep=" ")
remove_col = c('V258')
train = train[,!names(train) %in% remove_col]
test = test[,!names(test) %in% remove_col]

split = sample.split(train$V1, 0.05)
train_split = subset(train, split ==T)

# 2 Define X, Y and Scale
X = data.frame(train_split[,-1])
X = scale(X, center = TRUE, scale = FALSE)
y = data.frame(train_split[,1])
ohe_model = OneHotEncoder.fit(y)
y_ohe = as.matrix(transform(ohe_model,y))
y_ohe = as.data.frame(y_ohe)
names(y_ohe) = c("C0","C1","C2","C3","C4","C5","C6","C7","C8","C9")

# 3 Perform Multi, Compute Avg R sq
train_data = data.frame(X,y_ohe)
formula1 = cbind(C0,C1,C2,C3,C4,C5,C6,C7,C8,C9) ~ . -1
model1 = lm(formula1, data = train_data)
print(paste("R Squared Train",(summary(model1)[[1]])$r.squared))

# 4 Compute LOOCV
PRESS  <- colSums((model1$residuals/(1-ls.diag(model1)$hat))^2)
R2cv   <- 1 - PRESS/(diag(var(y_ohe))*(nrow(X)-1))
print(paste("Avg LOOCV R^2 Train",mean(R2cv)))

# 5 Predict with Centering the average R2 in the test data
Xp = data.frame(test[,-1])
Yp = data.frame(test[,1])
Xp = scale(Xp, center = TRUE, scale = FALSE)
test_scale = data.frame(Xp,Yp)
Yhat = predict(model1,test_scale)
RSS = colSums((Yp-Yhat)^2)
TSS = apply(Yp,2,function(x){sum((x-mean(x))^2)})
r2 = mean(1 - (RSS/TSS))
r2

# 6 Assign every test individual to the maximum response
Yp_hat = data.frame(unname(apply(Yhat, 1, which.max)) - 1)
eval = eval_func(unlist(Yp),unlist(Yp_hat), cm_show = T)
print(paste("Accuracy :",eval[1]))
print(paste("Error :",eval[2]))
```

```r
# 7 Perform a PCR without LOO
ncomp = 255
model2 = pcr(formula1, data = train_data, ncomp = ncomp)
Yhat = predict(model2,test)
Y = data.frame(test[,1])
pred_dim = dim(Yhat)
accuracy_metrics = c()
error_metric = c()
for(i in 1:pred_dim[3]){
  Yhat_class = data.frame(unname(apply(Yhat[,,i], 1, which.max)) - 1)
  eval = eval_func(unlist(Y),unlist(Yhat_class))
  accuracy_metrics = c(accuracy_metrics, eval[1])
  error_metric = c(error_metric, eval[2])
}

line_fix =  which.min(error_metric)
print(paste("Component ",line_fix))
print(paste("Error ",min(error_metric)))
qplot(1:pred_dim[3],error_metric,
      xlab="Number of Components", ylab="Error",
      main="PCR Error Plot", geom='line') +
  geom_vline(xintercept = line_fix,  color = "red", linetype="dotted")

# 8 PCR with LOOCV with 60 Components.
model3 = pcr(formula1, data = train_data, ncomp = 60, validation= "LOO")
Yhat = predict(model2,test)
Yhat = data.frame(unname(apply(Yhat[,,60], 1, which.max)) - 1)
Y = data.frame(test[,1])
eval_func(unlist(Y),unlist(Yhat))

# Experiment
# 5 percent
split = sample.split(train$V1, 0.05)
train_split = subset(train, split ==T)
X = data.frame(train_split[,-1])
X = scale(X, center = TRUE, scale = FALSE)
y = data.frame(train_split[,1])
ohe_model = OneHotEncoder.fit(y)
y_ohe = as.matrix(transform(ohe_model,y))
y_ohe = as.data.frame(y_ohe)
names(y_ohe) = c("C0","C1","C2","C3","C4","C5","C6","C7","C8","C9")
train_data = data.frame(X,y_ohe)
model4 = pcr(formula1, data = train_data, ncomp = 60)
Y = data.frame(test[,1])
Yhat = predict(model4,test)
Yhat = data.frame(unname(apply(Yhat[,,60], 1, which.max)) - 1)
eval = eval_func(unlist(Y),unlist(Yhat))
eval

# 30 percent
split = sample.split(train$V1, 0.30)
train_split = subset(train, split ==T)
X = data.frame(train_split[,-1])
X = scale(X, center = TRUE, scale = FALSE)
y = data.frame(train_split[,1])
ohe_model = OneHotEncoder.fit(y)
y_ohe = as.matrix(transform(ohe_model,y))
y_ohe = as.data.frame(y_ohe)
```

```r
names(y_ohe) = c("C0","C1","C2","C3","C4","C5","C6","C7","C8","C9")
train_data = data.frame(X,y_ohe)
model5 = pcr(formula1, data = train_data, ncomp = 60)
Y = data.frame(test[,1])
Yhat = predict(model5,test)
Yhat = data.frame(unname(apply(Yhat[,,60], 1, which.max)) - 1)
eval = eval_func(unlist(Y),unlist(Yhat))
eval

# 70 percent
split = sample.split(train$V1, 0.70)
train_split = subset(train, split ==T)
X = data.frame(train_split[,-1])
X = scale(X, center = TRUE, scale = FALSE)
y = data.frame(train_split[,1])
ohe_model = OneHotEncoder.fit(y)
y_ohe = as.matrix(transform(ohe_model,y))
y_ohe = as.data.frame(y_ohe)
names(y_ohe) = c("C0","C1","C2","C3","C4","C5","C6","C7","C8","C9")
train_data = data.frame(X,y_ohe)
model6 = pcr(formula1, data = train_data, ncomp = 60)
Y = data.frame(test[,1])
Yhat = predict(model6,test)
Yhat = data.frame(unname(apply(Yhat[,,60], 1, which.max)) - 1)
eval = eval_func(unlist(Y),unlist(Yhat))
eval


# Not In scope
# Plot Digits
par(mfrow=c(3,4))
par(mar=c(2,2,2,2))
for(i in 1:12){
  plot_img(train, i)
}

# Percentage of Variance
pvar = cumsum(explvar(model2))
qplot(1:length(pvar),unlist(pvar),
      xlab="Number of Components", ylab="Variance Explained",
      main="PCR Variance Plot", geom='line')

# Accuracy
qplot(1:pred_dim[3],accuracy_metrics,
      xlab="Number of Components", ylab="Accuracy",
      main="PCR Accuracy Plot", geom='line') +
  geom_vline(xintercept = line_fix,  color = "red", linetype="dotted")

# PCA
library(ggfortify)
train$V1 = as.factor(train$V1)
autoplot(prcomp(train[,-1]), data = train, colour = 'V1', label = F,  loadings = F)
```

```r
# Custom Utilities

# Evaluation
eval_func = function(y, yhat, cm_show = FALSE){
  metrics = c()
  cm = table(y,yhat)

  if(cm_show == TRUE){
    print(cm)
  }

  total = sum(cm)
  no_diag = cm[row(cm) != (col(cm))]
  acc = sum(diag(cm))/total
  error = sum(no_diag)/total
  metrics = c(acc,error)
  return(metrics)
}

# Plotting Image
plot_img = function(train_data,i,show_target=FALSE){
  CUSTOM_COLORS = colorRampPalette(colors = c("black", "white"))
  if(show_target==TRUE){
    print(train_data[i,1])
  }
  train_data = train_data[,-1]
  z = unname(unlist((train_data[i,])))
  k = matrix(z,nrow = 16,ncol = 16)
  rotate <- function(x) t(apply(x, 2, rev))
  image(rotate(t(k)), col = CUSTOM_COLORS(256))
}
```