**1.1a Principal Component Analysis (PCA) – R code**

Principal Component Analysis is used to reduce the dimensionality of the input. In the code below 8 x 8 pixel of handwritten digits is reduced into its principal components. Then a scatter plot of the first 2 principal components give a very good visial representation of the data

library(dplyr)

library(ggplot2)

#Note: This example is adapted from an the example in the book Python Datascience handbook by

# Jake VanderPlas (https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html)

# Read the digits data (From sklearn datasets)

digits= read.csv("digits.csv")

# Create a digits classes target variable

digitClasses <- factor(digits$X0.000000000000000000e.00.29)

#Invoke the Principal Componsent analysis on columns 1-64

digitsPCA=prcomp(digits[,1:64])

# Create a dataframe of PCA

df <- data.frame(digitsPCA$x)

# Bind the digit classes

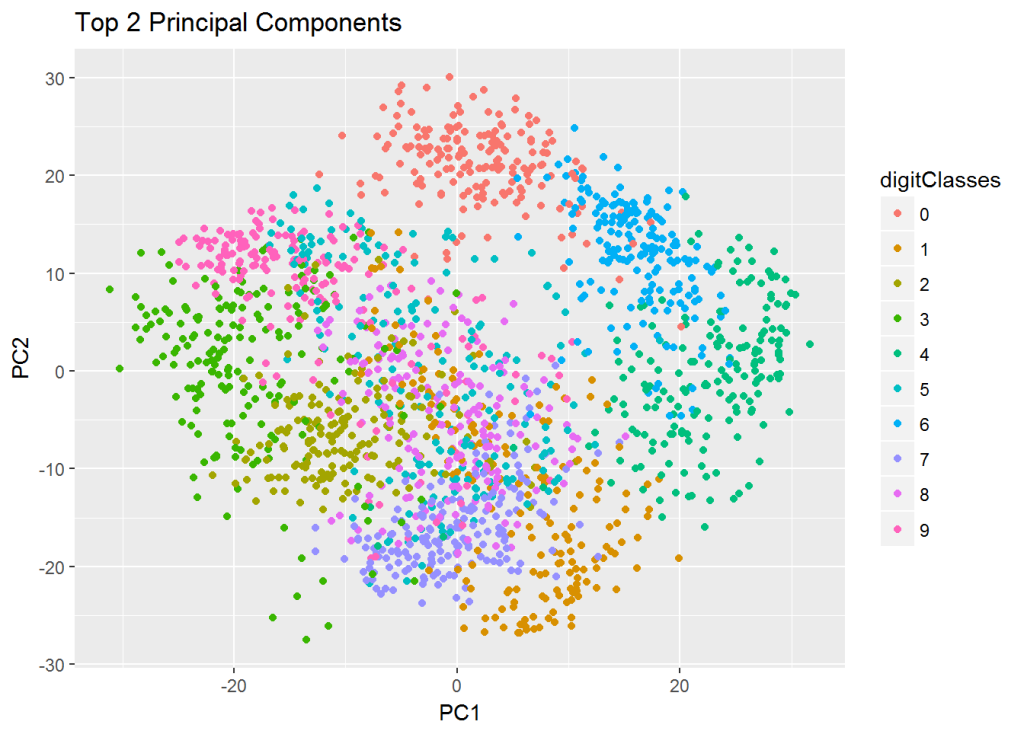
df1 <- cbind(df,digitClasses)

# Plot only the first 2 Principal components as a scatter plot. This plot uses only the

# first 2 principal components

ggplot(df1,aes(x=PC1,y=PC2,col=digitClasses)) + geom\_point() +

ggtitle("Top 2 Principal Components")

****

**1.1 b Variance explained vs no principal components – R code**

In the code below the variance explained vs the number of principal components is plotted. It can be seen that with 20 Principal components almost 90% of the variance is explained by this reduced dimensional model.

# Read the digits data (from sklearn datasets)

digits= read.csv("digits.csv")

# Digits target

digitClasses <- factor(digits$X0.000000000000000000e.00.29)

digitsPCA=prcomp(digits[,1:64])

# Get the Standard Deviation

sd=digitsPCA$sdev

# Compute the variance

digitsVar=digitsPCA$sdev^2

#Compute the percent variance explained

percentVarExp=digitsVar/sum(digitsVar)

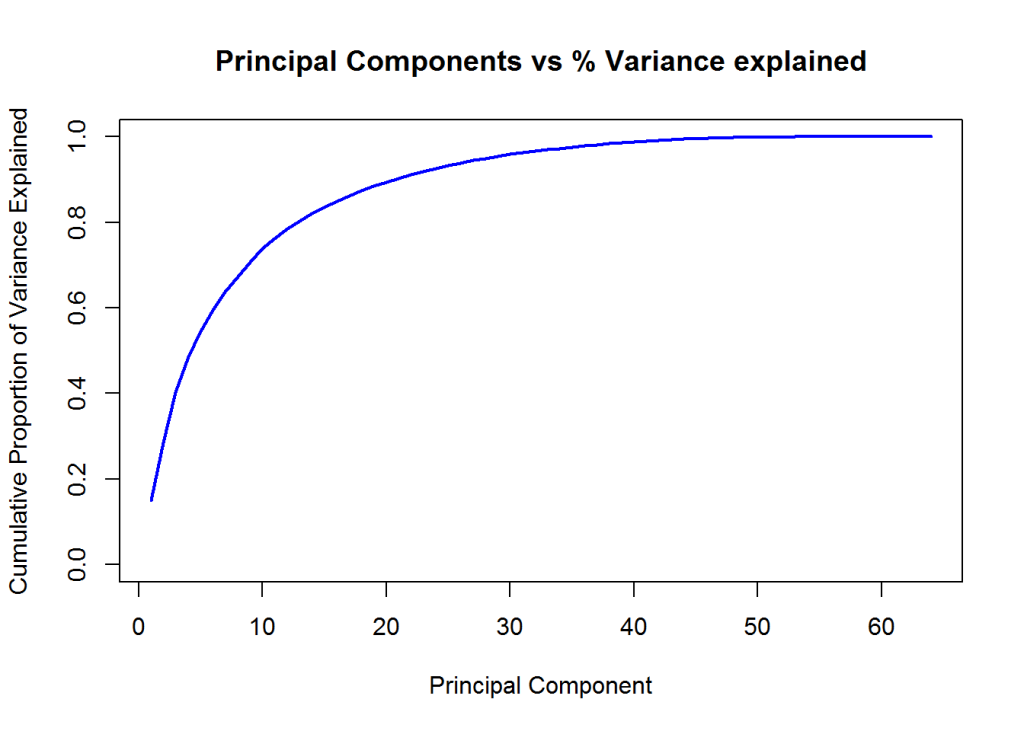
# Plot the percent variance exlained as a function of the number of principal components

#plot(cumsum(percentVarExp), xlab="Principal Component",

# ylab="Cumulative Proportion of Variance Explained",

# main="Principal Components vs % Variance explained",ylim=c(0,1),type='l',lwd=2,

# col="blue")

****

**1.1c Principal Component Analysis (PCA) – Python code**

import numpy as np

from sklearn.decomposition import PCA

from sklearn import decomposition

from sklearn import datasets

import matplotlib.pyplot as plt

from sklearn.datasets import load\_digits

# Load the digits data

digits = load\_digits()

# Select only the first 2 principal components

pca = PCA(2) # project from 64 to 2 dimensions

#Compute the first 2 PCA

projected = pca.fit\_transform(digits.data)

# Plot a scatter plot of the first 2 principal components

plt.scatter(projected[:, 0], projected[:, 1],

c=digits.target, edgecolor='none', alpha=0.5,

cmap=plt.cm.get\_cmap('spectral', 10))

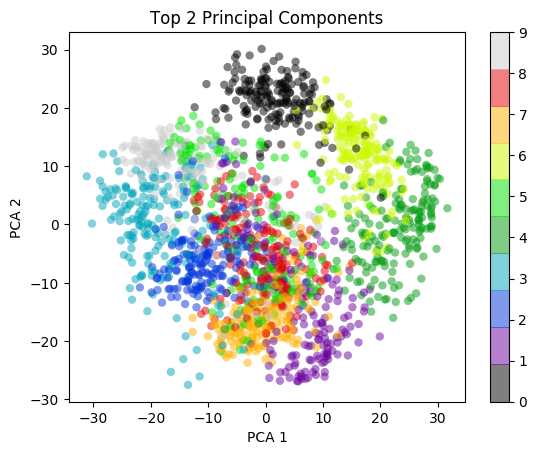
plt.xlabel('PCA 1')

plt.ylabel('PCA 2')

plt.colorbar();

plt.title("Top 2 Principal Components")

plt.savefig('fig1.png', bbox\_inches='tight')

****

**1.1 b Variance vs no principal components**

**– Python code**

import numpy as np

from sklearn.decomposition import PCA

from sklearn import decomposition

from sklearn import datasets

import matplotlib.pyplot as plt

from sklearn.datasets import load\_digits

digits = load\_digits()

# Select all 64 principal components

pca = PCA(64) # project from 64 to 2 dimensions

projected = pca.fit\_transform(digits.data)

# Obtain the explained variance for each principal component

varianceExp= pca.explained\_variance\_ratio\_

# Compute the total sum of variance

totVarExp=np.cumsum(np.round(pca.explained\_variance\_ratio\_, decimals=4)\*100)

# Plot the variance explained as a function of the number of principal components

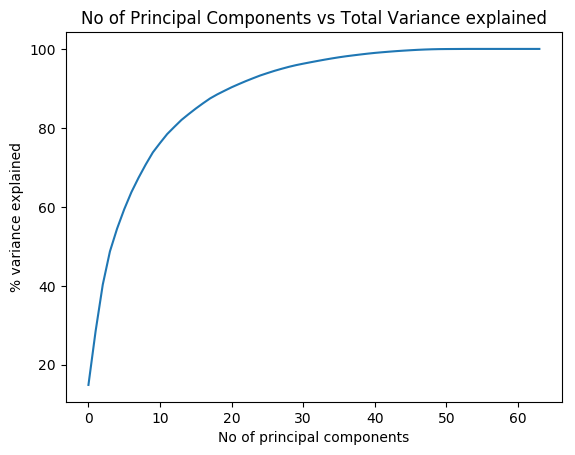
plt.plot(totVarExp)

plt.xlabel('No of principal components')

plt.ylabel('% variance explained')

plt.title('No of Principal Components vs Total Variance explained')

plt.savefig('fig2.png', bbox\_inches='tight')

****

**1.2a K-Means – R code**

In the code first the scatter plot of the first 2 Principal Components of the handwritten digits is plotted as a scatter plot. Over this plot 10 centroids of the 10 different clusters corresponding the 10 diferent digits is plotted over the original scatter plot.

library(ggplot2)

# Read the digits data

digits= read.csv("digits.csv")

# Create digit classes target variable

digitClasses <- factor(digits$X0.000000000000000000e.00.29)

# Compute the Principal COmponents

digitsPCA=prcomp(digits[,1:64])

# Create a data frame of Principal components and the digit classes

df <- data.frame(digitsPCA$x)

df1 <- cbind(df,digitClasses)

# Pick only the first 2 principal components

a<- df[,1:2]

# Compute K Means of 10 clusters and allow for 1000 iterations

k<-kmeans(a,10,1000)

# Create a dataframe of the centroids of the clusters

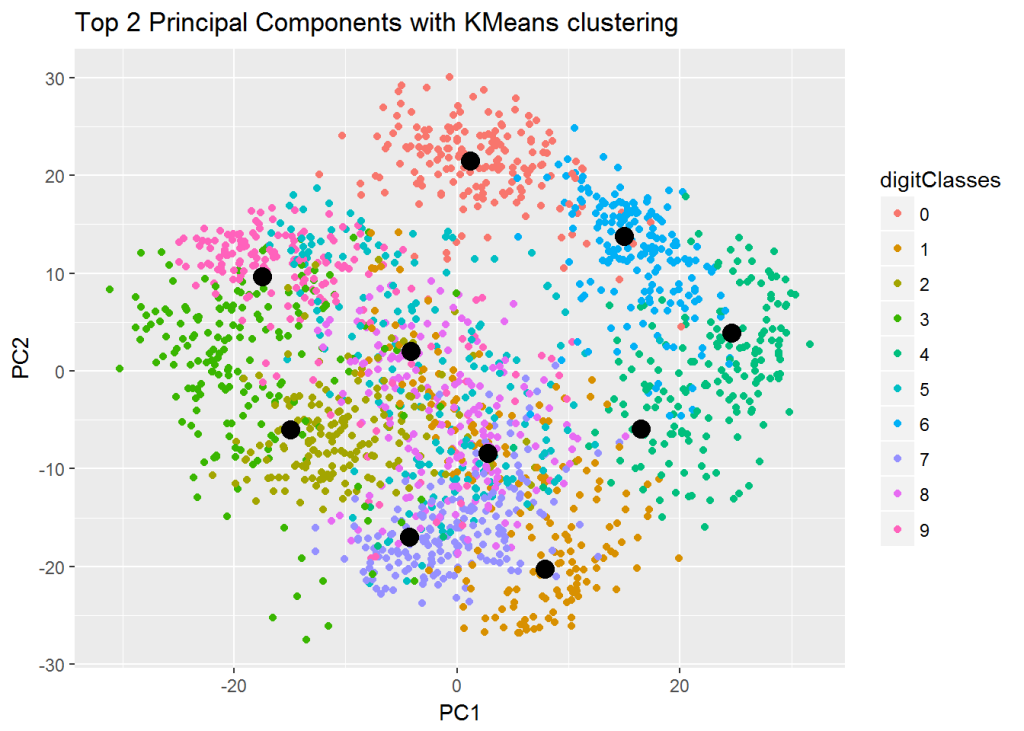
df2<-data.frame(k$centers)

#Plot the first 2 principal components with the K Means centroids

ggplot(df1,aes(x=PC1,y=PC2,col=digitClasses)) + geom\_point() +

geom\_point(data=df2,aes(x=PC1,y=PC2),col="black",size = 4) +

ggtitle("Top 2 Principal Components with KMeans clustering")

****

**1.2b K-Means – Python code**

The centroids of the 10 different handwritten digits is plotted over the scatter plot of the first 2 principal components.

import numpy as np

from sklearn.decomposition import PCA

from sklearn import decomposition

from sklearn import datasets

import matplotlib.pyplot as plt

from sklearn.datasets import load\_digits

from sklearn.cluster import KMeans

digits = load\_digits()

# Select only the 1st 2 principal components

pca = PCA(2) # project from 64 to 2 dimensions

projected = pca.fit\_transform(digits.data)

# Create 10 different clusters

kmeans = KMeans(n\_clusters=10)

# Compute the clusters

kmeans.fit(projected)

y\_kmeans = kmeans.predict(projected)

# Get the cluster centroids

centers = kmeans.cluster\_centers\_

centers

#Create a scatter plot of the first 2 principal components

plt.scatter(projected[:, 0], projected[:, 1],

c=digits.target, edgecolor='none', alpha=0.5,

cmap=plt.cm.get\_cmap('spectral', 10))

plt.xlabel('PCA 1')

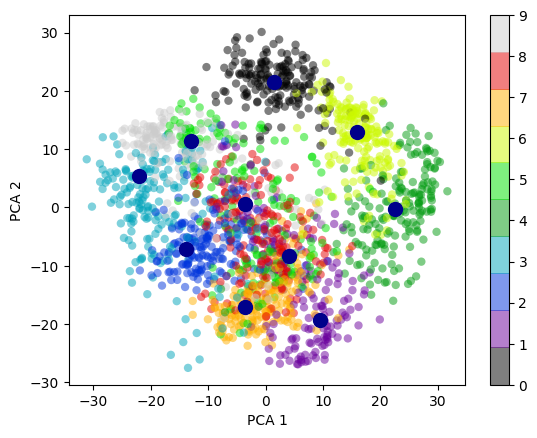
plt.ylabel('PCA 2')

plt.colorbar();

# Overlay the centroids on the scatter plot

plt.scatter(centers[:, 0], centers[:, 1], c='darkblue', s=100)

plt.savefig('fig3.png', bbox\_inches='tight')

****

**1.3a Heirarchical clusters – R code**

Herirachical clusters is another type of unsupervised learning. It successively joins the closest pair of objects (points or clusters) in succession based on some ‘distance’ metric. In this type of clustering we do not have choose the number of centroids. We can cut the created dendrogram mat an appropriate height to get a desired and reasonable number of clusters These are the following ‘distance’ metrics used while combining successive objects

* Ward
* Complete
* Single
* Average
* Centroid

# Read the IRIS dataset

iris <- datasets::iris

iris2 <- iris[,-5]

species <- iris[,5]

#Compute the distance matrix

d\_iris <- dist(iris2)

# Use the 'average' method to for the clsuters

hc\_iris <- hclust(d\_iris, method = "average")

# Plot the clusters

plot(hc\_iris)

# Cut tree into 3 groups

sub\_grp <- cutree(hc\_iris, k = 3)

# Number of members in each cluster

table(sub\_grp)

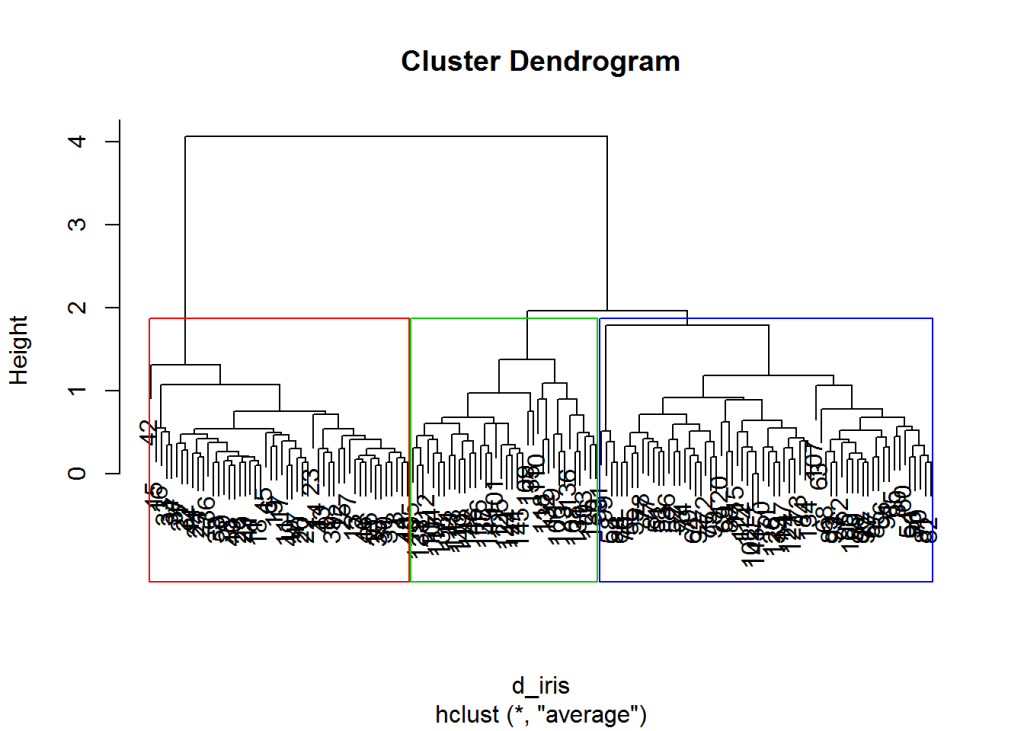
## sub\_grp

## 1 2 3

## 50 64 36

# Draw rectangles around the clusters

rect.hclust(hc\_iris, k = 3, border = 2:5)

****

**1.3a Heirarchical clusters – Python code**

from sklearn.datasets import load\_iris

import matplotlib.pyplot as plt

from scipy.cluster.hierarchy import dendrogram, linkage

# Load the IRIS data set

iris = load\_iris()

# Generate the linkage matrix using the average method

Z = linkage(iris.data, 'average')

#Plot the dendrogram

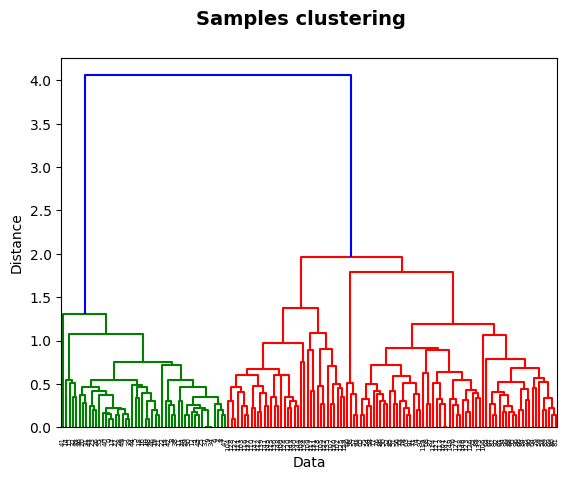
#dendrogram(Z)

#plt.xlabel('Data')

#plt.ylabel('Distance')

#plt.suptitle('Samples clustering', fontweight='bold', fontsize=14);

#plt.savefig('fig4.png', bbox\_inches='tight')

****