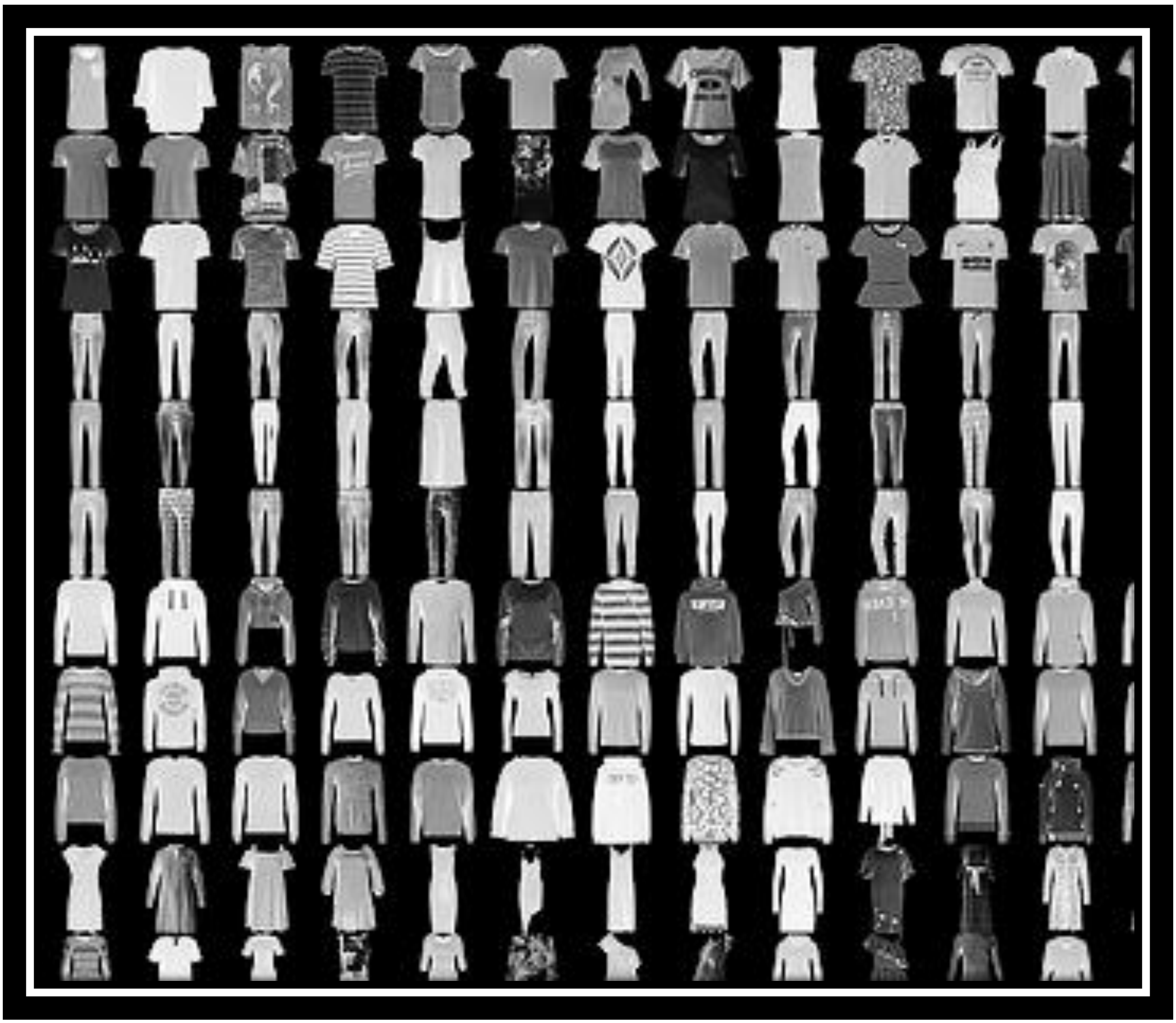
# Supervised Learning - R Project

29th Nov 2018 **Suhas Patil**

**Fashion MNIST** (https://www.kaggle.com/zalando-research/fashionmnist)
**An MNIST-like dataset of 70,000 28x28 labeled fashion images**



# Part 1 : Report Summary

## 1.1 Background

Have you ever searched using only images? Many consumers tend to want to get specific information through images search. In keeping with this trend, Google's image search allows people to search using images.

If the e-commerce company (such as JCPenney, Macy's, Myntra, Amazon) uses the image classification function for convenient shopping for potential customers, can the company get business benefits? Our group started the group project here.

## 1.2 Objectives

We have three goals:

✓ First, finding and analyzing algorithms that best analyze the dataset
- Classify images into to one of the following labels by analyzing the pixel darkness at different pixel locations: 0
- T-shirt/top, 1 Trouser, 2 Pullover, 3 Dress, 4 Coat, 5 Sandal, 6 Shirt, 7 Sneaker, 8 Bag, 9 Ankle boot

✓ Second, what is the most accurate model to suggest the company?

- Some companies need to give the customer the most accurate results. For example, if you need to give accurate information on a health care website, using a model that gives accurate results even if it takes a little time, is a top priority for the company and the customer.

✓ Third, what is the model which gives the fastest result?

- For e-commerce websites with high customer drop-off rates, we need an algorithmic model that can yield fast results

# 1.3 Describe the dataset ‘Fashion-MNIST’

## 3-1. Reason for selection

- Other Datasets we came across
  - Youtube Trending Music
  - Credit Risk Analysis
  - China Pollution 2.5PM
  - ECG Heartbeat Categorization
  - Walmart Store Sales Forecasting
  - The Movie Dataset

We didn’t choose these datasets because the datasets were either old or we didn’t have enough domain knowledge.

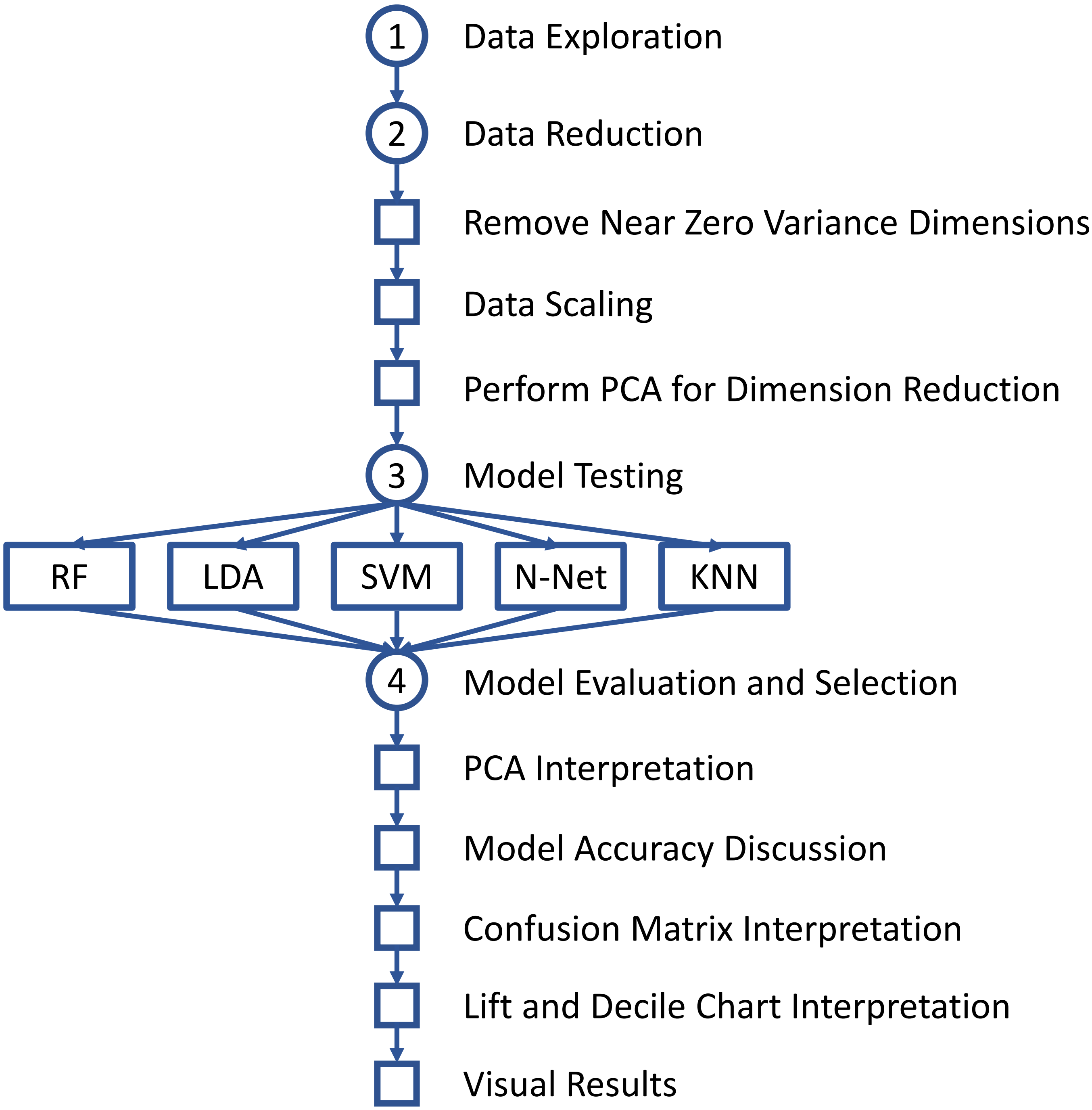
- Why we chose this Dataset?

This is an exciting dataset where we can apply what we have learnt in the class. Making predictions on this dataset will expose us to real world problems.

## 3-2. Dataset Description

- Dataset of Zalando’s article images
- A training set of 60,000 examples and a test set of 10,000 examples.
- Each example is a 28x28 grayscale image, associated with a label from 10 classes.
- Each value is the darkness of the pixel (1 to 255)
- Each training and test example is assigned to one of the following labels:0 T-shirt/top, 1 Trouser, 2 Pullover, 3 Dress, 4 Coat, 5 Sandal, 6 Shirt, 7 Sneaker, 8 Bag, 9 Ankle boot
- No need to data cleansing (checked for NA values): none of NA observations in training and testing dataset

## 1.4 Process


1
Data Exploration
2
Data Reduction
Remove Near Zero Variance Dimensions
Data Scaling
Perform PCA for Dimension Reduction
3
Model Testing
RF
LDA
SVM
N-Net
KNN
4
Model Evaluation and Selection
PCA Interpretation
Model Accuracy Discussion
Confusion Matrix Interpretation
Lift and Decile Chart Interpretation
Visual Results

# 1.5 Data Exploration

Each row is a 28x28 grayscale image, associated with a label from 10 classes.
A training set of 60,000 examples and a test set of 10,000 examples.

3 steps for data exploration
1. Check the number of data observations for each category
Observations per Class in Training Data
All classes have equal no. of observations(n=6000).

2. Explore data as Image
Actual Image plot according to 784 pixel plots
Corners usually have low pixel value(low darkness or no darkness)

3. Scatter plot of Class Mean Pixel value
Pixel Value by Class Analysis
Coat followed by Pullover and Bag have the <u>Highest Mean Pixel Value</u>
Sandal followed by Sneaker have the <u>Lowest Mean Pixel Value</u>

# 1.6 Predictive/Classification Algorithm

| Algorithms | Classification/ Predictive | Supervised / Unsupervised | Concept |
|---|---|---|---|
| Random Forest | Classification | Supervised | Data is randomly sampled to create multiple decision trees and then collects the results of the decision trees and derives the final result. |
| Linear Discriminant Analysis (LDA) | Classification | Supervised | Like logistic regression, it is a classical statistical technique that can be used for classification and profiling. |
| Support Vector Machine (SVM) | Classification | Supervised | Given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. |
| Neural Network (NNET) | Classification | Supervised | Operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. |
| K-Nearest Neighbors (KNN) | Classification | Supervised | It belongs to the family of supervised machine learning algorithms which means we use labeled (Target Variable) dataset to predict the class of new data point |

# 1.7 Results

| model | accuracy_test | Multiclass AUC | SecsTaken_test |
|---|---|---|---|
| rf | 0.86 | 0.92 | 32 |
| svm | 0.90 | 0.94 | 642 |
| lda | 0.79 | 0.89 | 1 |
| nnet | 0.88 | 0.93 | 1126 |
| knn | 0.86 | 0.92 | 345 |

- SVM (Support Vector Machine) is <u>the most accurate model</u> among 5 algorithms.
  → It means that to get an accurate image classification result, e-commerce companies would better to choose SWM model

- IDA is <u>the fastest model</u> among 5 algorithms.
  → It implies that to get a fast result, companies would use IDA algorithms to satisfy the customers who needs fast results

- <u>Highest AUC</u> is given by Support Vector Machine(SVM) model
  → *we used Confusion matrix and AUC*
  For prediction, metrics we use Average Error, MAPE, and RMSE (based on the validation data).
  For <u>classification tasks</u>, metrics based on the confusion matrix include overall accuracy, specificity and sensitivity, and metrics that account for misclassification costs.

# Part 2 : Initial Data Exploration

## Data Exploration

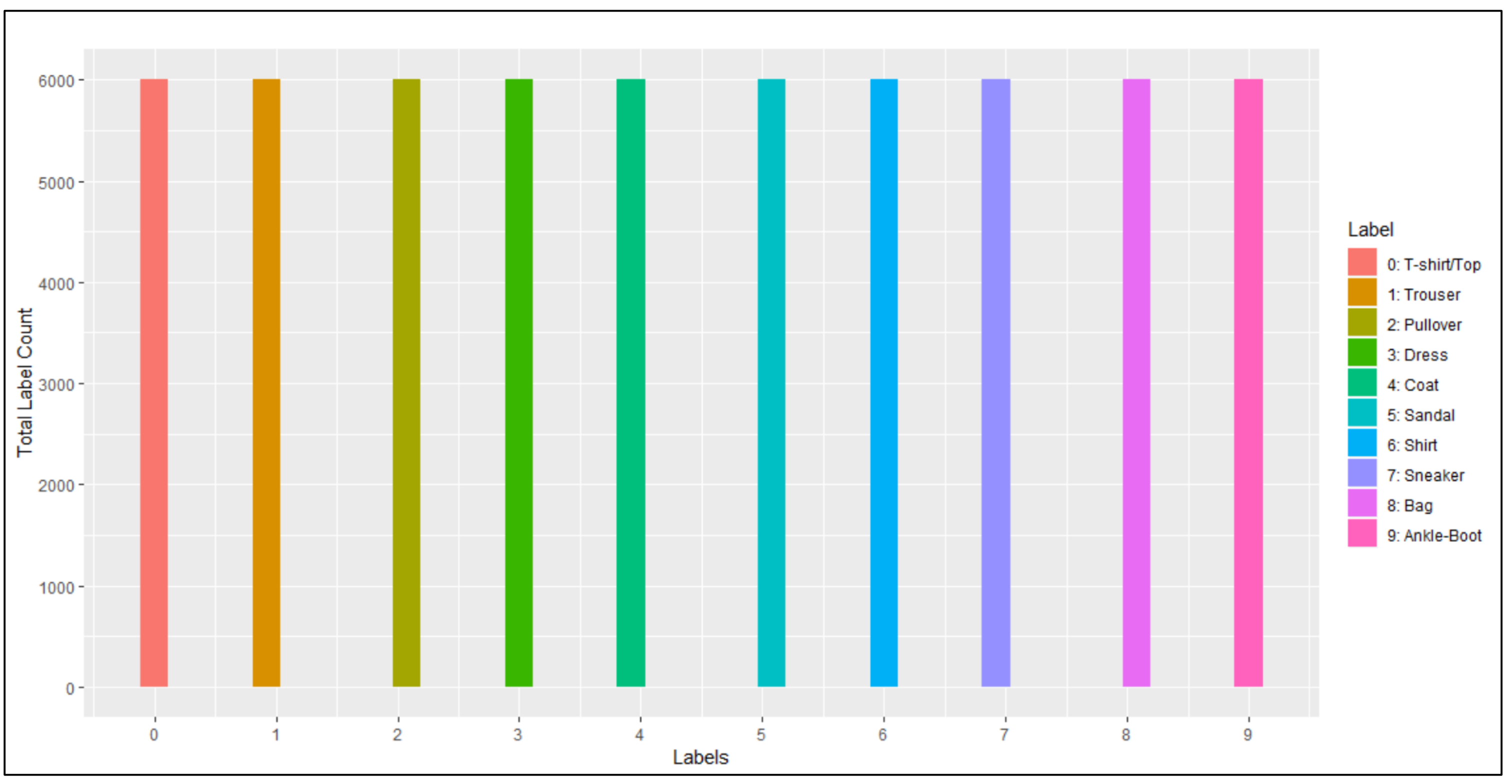
Each row is a 28x28 grayscale image, associated with a label from 10 classes.

- Each row is a separate image
- Column 1 is the class label.
- Remaining columns are pixel numbers (784 total).
- Each value is the darkness of the pixel (0 to 255)

Observations by Class in Training Data

**Parameter used : Observations by Class Bar Plot**

All classes have equal no. of observations(n=6000).



# Data Exploration

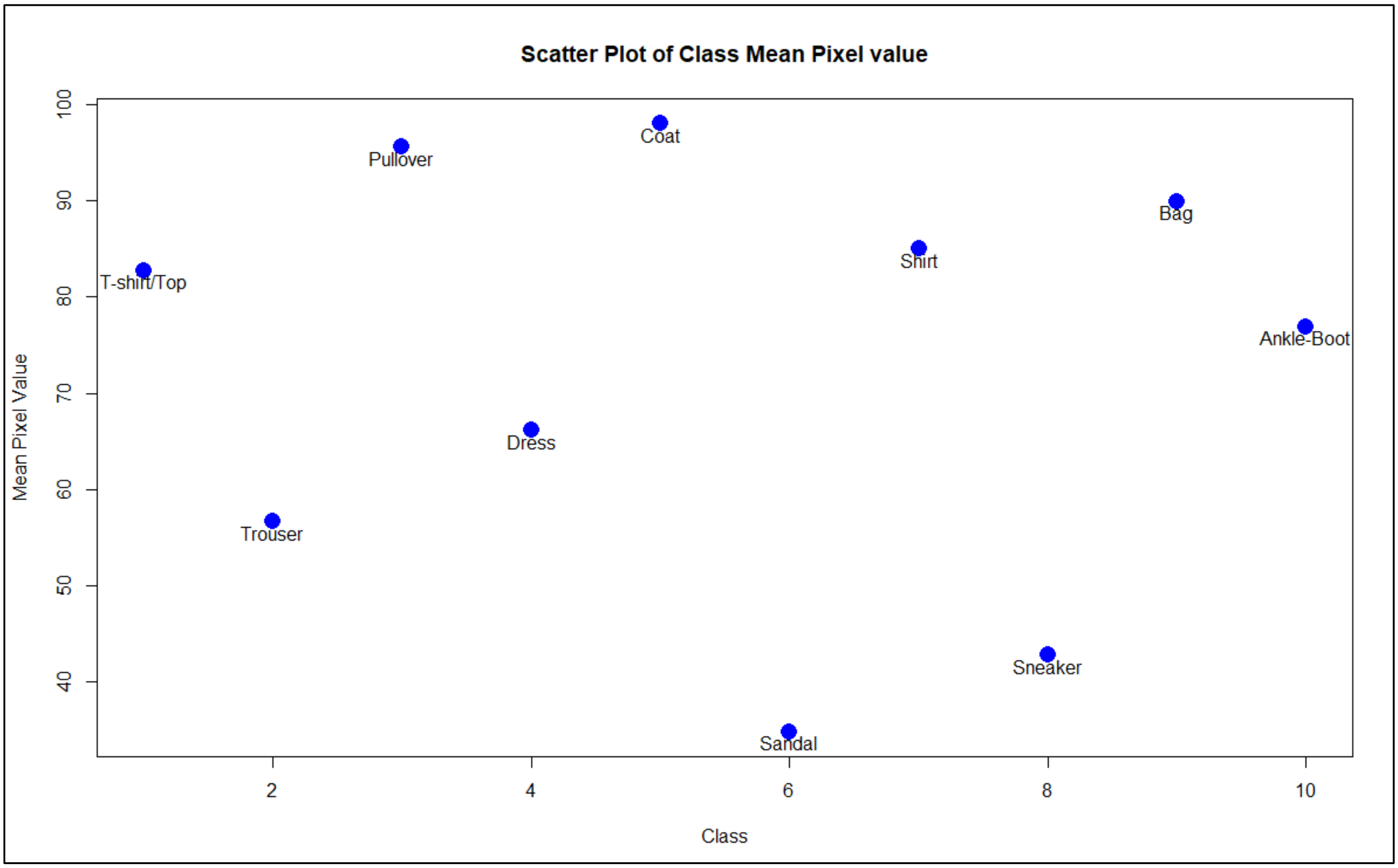
Actual Image plot according to 784 pixel plots

**Parameter used : Image Plot of 25 random values**

Corners usually have low pixel value(low darkness or no darkness)

# Data Exploration

Pixel Value by Class Analysis
**Parameter used : Class Mean Pixel Value Scatter Plot**
Coat followed by Pullover and Bag have the Highest Mean Pixel Value
Sandal followed by Sneaker have the Lowest Mean Pixel Value




Overall Interpretation

1. Data cleaning is not required(No NAs, etc)
2. Firstly we can reduce dimensions by looking at pixel that have zero variance or near zero variance
3. We can scale the data by dividing all columns by 255(except column 1) because our pixel values range between 0 ~ 255
4. We should consider Principal Component Analysis for further dimension reduction

# Part-2 Data Exploration R-Code

```
#Load packages
Packages <- c("dplyr", "ggplot2","knitr","readr","gplots","ggplot2","ggthemes","plotly","caret","gains")
lapply(Packages, library, character.only = TRUE)

#Load datatset
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
train <- read.csv("fashion-mnist_train.csv")
valid<- read.csv("fashion-mnist_test.csv")
dim(train)
sum(is.na(train))

##Name Labels
l.names <- c("T-shirt/Top","Trouser","Pullover","Dress","Coat","Sandal","Shirt","Sneaker","Bag","Ankle-Boot")
```

**Load Package, Dataset, Check NA, Check Dimensions**

```
#Explore Dataset
##Exploring no of observations per label name
label.factor <- as.factor(train$label)
levels(label.factor) <- l.names

###Making frequency chart
label_count_plot_train <- ggplot(data = train, aes(x = label, fill = as.factor(label))) +
  geom_histogram(bins = 40) +
  scale_x_continuous(breaks = seq(min(0), max(25), by = 1), na.value = TRUE) +
  scale_y_continuous(breaks = seq(min(0), max(10000), by = 1000)) +
  labs(title = "", x = "Labels", y = "Total Label Count", fill = "Label")+
  scale_fill_discrete(labels=c("0: T-shirt/Top","1: Trouser","2: Pullover",
                        "3: Dress","4: Coat","5: Sandal","6: Shirt",
                        "7: Sneaker","8: Bag","9: Ankle-Boot"))
#png('Data Explored in Categories.png')
label_count_plot_train
ggplotly(label_count_plot_train, width = 700, height = 800)
```

**Observations by Class Value Bar Plot**

```
##Plot a random sample of product images
library(purrr)
xy_axis = data.frame(x = expand.grid(1:28,28:1)[,1],
                     y = expand.grid(1:28,28:1)[,2])
plot_theme = list(raster = geom_raster(hjust = 0, vjust = 0),
                  gradient_fill = scale_fill_gradient(low = "white", high = "black", guide = FALSE),
                  theme = theme(axis.title = element_blank(), panel.background = element_blank(),
                         panel.border = element_blank(),panel.grid.major = element_blank(),
                         panel.grid.minor = element_blank(), plot.background = element_blank(),
                         aspect.ratio = 1))
sample_plots = sample(1:nrow(train),25) %>% map(~ {
  plot_data = cbind(xy_axis, fill = as.data.frame(t(train[.x, -1]))[,1])
  ggplot(plot_data, aes(x, y, fill = fill)) + plot_theme
})
library(gridExtra)
#png('Data Explored in Images.png')
do.call("grid.arrange", c(sample_plots, ncol = 5, nrow = 5))
dev.off()
```

**Image Plot of 25 random values**

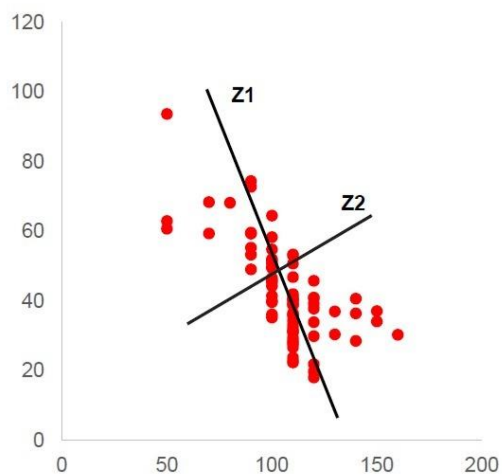
```
##Explore Mean pixel value per Class
###Create Mean Pixel Table
label_means <- train %>%
  group_by(label)%>%
  dplyr::summarize_all(mean)%>%
  rowMeans()
labelmeans<- data.frame(l.names,label_means)
labelmeans[order(labelmeans$label_means,decreasing = TRUE),]
###Create Scatter Plot of Mean Pixel Value
plot(label_means, type="p", ylab="Mean Pixel Value", xlab="Class",
    main = "Scatter Plot of Class Mean Pixel value")
points(label_means, cex = 2, col = "blue",pch = 16)
text(x =label_means,labels =l.names,adj = c(0.5,1.2),cex = 1)
```

**Class Mean Pixel Value Scatter Plot**

# Part 3 : Algorithm Explanation

## 3.1 Principal Component Analysis

- PCA is a method used to reduce number of variables in your data by extracting important one from a large pool. It reduces the dimension of your data with the aim of retaining as much information as possible
- In other words, this method combines highly correlated variables together to form a smaller number of an artificial set of variables which is called "principal components" that account for most variance in the data.
- We use "*prcomp*" function from the "*stats*" package to run PCA




Advantages:

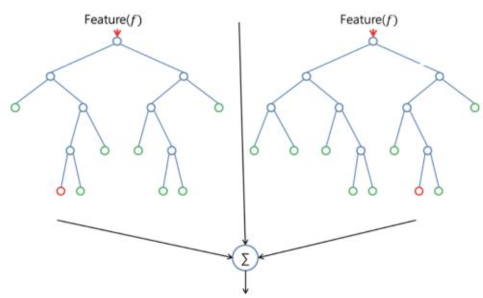
- PCA achieves dimension reduction by creating new, artificial variables called principal components.
- PCA is an unsupervised method, meaning that no information about groups is used in the dimension reduction

Disadvantages:

- PCA assumes that the principle components are a linear combination of the original features. If this is not true, PCA will not give you sensible results.
- PCA uses variance as the measure of how important a particular dimension is. So, high variance axes are treated as PC's, while low variance axes are treated as noise.

# 3.2 Random Forest

- Random Forests is a type of decision tree model which creates multiple trees to reach the node through the most efficient path. The creation of multiple trees using random samples gives it the name Random Forest. It trains each tree independently, using a random sample of the data. This randomness helps to make the model more robust than a single decision tree, and less likely to overfit on the training data. There are typically two parameters in RF - number of trees and no. of features to be selected at each node.
- In our script, we are considering the top 50 PCs derived from the PCA analysis on the original standardized dataset. The random forest is built over the training data consisting of the class labels mapped to their 50 predictor PCs (Capturing 99.9% of the cumulative variance of the data) and building 56 decision trees.
- No. of Features = 50 ,No. of Trees = 56
- We use the "**randomForest**" package to run the model. Since the data is totally uncorrelated through PCs and reduction of features from 786 to 50. We expect the Random forest to be fast and efficient.
- We achieve an accuracy of 86.31% on the classification on the validation dataset by running the Random forest model.




```
library(randomForest)
set.seed(1)
fmnist_rf=randomForest(label~. ,data = train.pca, ntree=56)
```

Advantages:

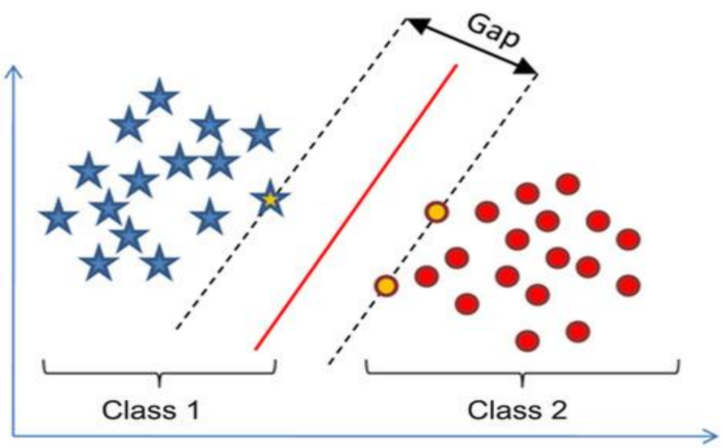
- Decorrelates trees (relative to bagged trees)
  - Important when dealing with multiple features which may be correlated
- Reduced variance (relative to regular trees)

Disadvantages:

- Not as easy to visually interpret

# 3.3 Support Vector Machine

- A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two-dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.
- The SVM model creates the hyperplane to gain the best separation with maximizing margin between the labelled classes and their associated PCs features in the dataset and the hyperplane cuts through the 50 orthogonal dimensions. This model is very efficient on the type of dataset and the accurate multiple classifications which we are expecting gain from it.




- We use the R package "**e1071**" to run the SVM.

```
library(e1071)
set.seed(1)
fmnist_svm <- svm(label ~ ., data=train.pca)
```

Advantages:

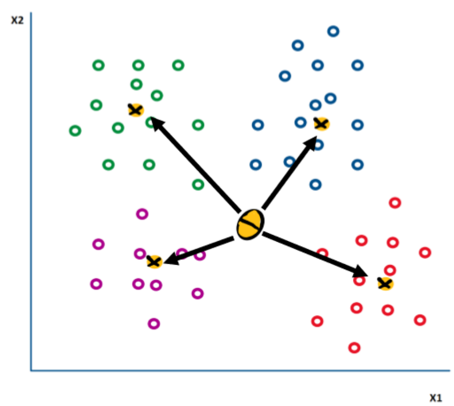
- Performs similarly to logistic regression when linear separation
- Performs well with non-linear boundary depending on the kernel used
- Handle high dimensional data well.

Disadvantages:

- Susceptible to overfitting/training issues depending on kernel

# 3.4 Linear Discriminant Analysis

- Discriminant analysis is a classification method. Like logistic regression, it is a classical statistical technique that can be used for classification and profiling.
- It uses sets of measurements on different classes of records to classify new records into one of those classes (classification). The LDA works like PCA but emphasizes on separation between the classes and uses statistical distance opposed to Euclidean distance. LDA can be used for dimension reduction also but in our approach, we are considering it only for classification.
- The LDA does not do as well as the other models in our data which is based on PCs which works on Euclidean distance.




```r
library(MASS)
set.seed(1)
fmnist_lda <- lda(label~.,data = train.pca)
```

Advantages:

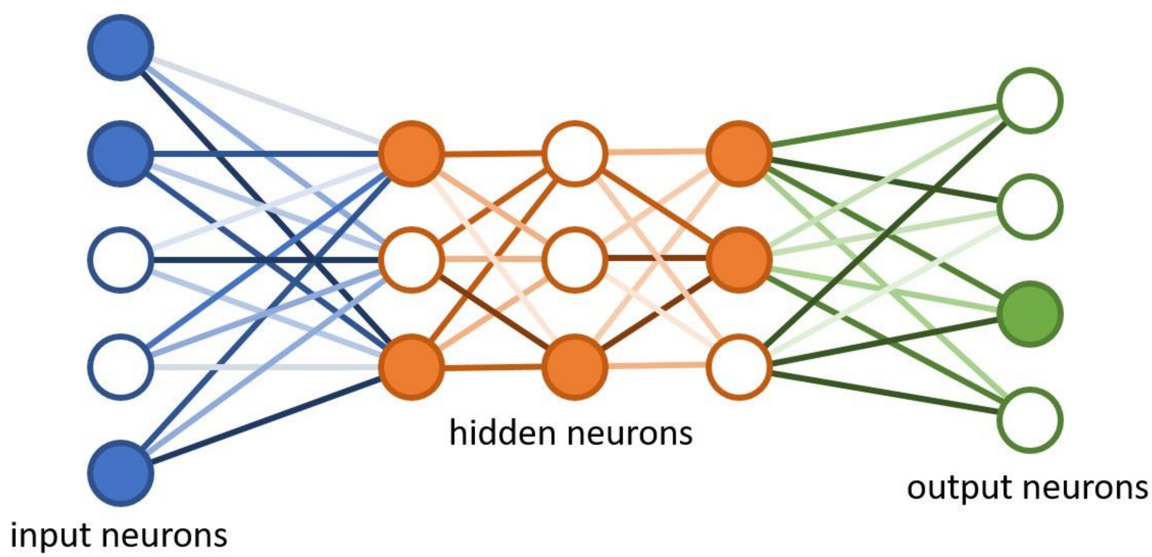
- Simple, Fast and portable
- Still beats some algorithms (logistic regression) when its assumptions are met
- Good to use when beginning a project
- Ability to compute the addition of Multivariate distribution
- Ability to compute CI

Disadvantages :

- Requires Normal Distribution
- Incompetent when there are only few Categories variables
- Suffers multicollinearity

# 3.5 Neural Nets (Feed-Forward neural network)

- A feedforward neural network is a biologically inspired classification algorithm. It consist of a (possibly large) number of simple neuron-like processing *units*, organized in *layers*. Every unit in a layer is connected with all the units in the previous layer. These connections are not all equal: each connection may have a different strength or *weight*. The weights on these connections encode the knowledge of a network. Often the units in a neural network are also called *nodes*.
- Data enters at the inputs and passes through the network, layer by layer, until it arrives at the outputs. During normal operation, that is when it acts as a classifier, there is no feedback between layers. This is why they are called *feedforward* neural networks.
- In the following figure we see an example of a 2-layered network with, from top to bottom: an output layer with 5 units, a *hidden* layer with 4 units, respectively. The network has 3 input units.




- The 3 inputs are shown as circles and these do not belong to any layer of the network (although the inputs sometimes are considered as a virtual layer with layer number 0). Any layer that is not an output layer is a *hidden* layer. This network therefore has 1 hidden layer and 1 output layer. The figure also shows all the connections between the units in different layers. A layer only connects to the previous layer.
- The model finally converges to the list of classes and sorts the record to the one with max weight,

# 3.5 Neural Nets (Feed-Forward neural network)

```
library(nnet)
set.seed(1)
n <- names(train.pca[,-1])
f <- as.formula(paste("label ~", paste(n[!n %in% "medv"], collapse = " + ")))
fmnist_nnet <- nnet(f,data = train.pca,
                    size=120,maxit=130,MaxNWts = 80000)
```

- We use the "*nnet*" package in run to run neural network.
- MaxNwts, are the maximum allowable number of weights, we have chosen 80000 for our model, to tune it for our requirements and balance accuracy with performance speed.
- Maxit, is the maximum number of iterations, we have considered 130 as optimal.
- Size, is the number of units in the hidden layer. We have considered 120 of them.

Advantages:

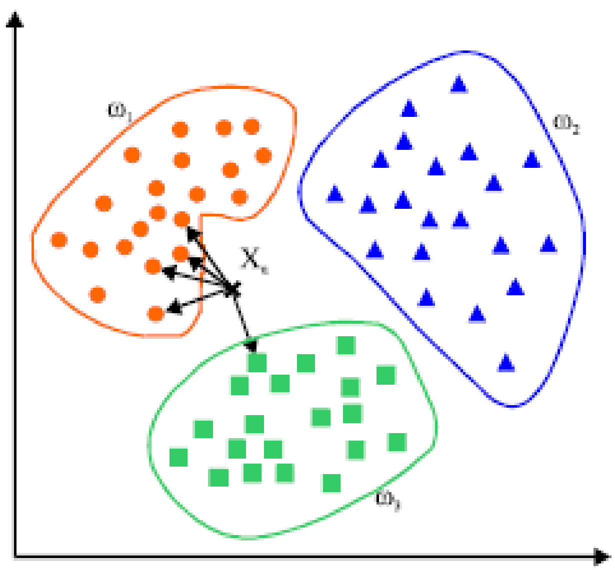
- Neural networks are flexible and can be used for both regression and classification problems. Any data which can be made numeric can be used in the model, as neural network is a mathematical model with approximation functions.
- Neural networks are good to model with nonlinear data with large number of inputs; for example, images. It is reliable in an approach of tasks involving many features. It works by splitting the problem of classification into a layered network of simpler elements.
- Once trained, the predictions are pretty fast.
- Neural networks can be trained with any number of inputs and layers.
- Neural networks work best with more data points.

Disadvantages:

- When NN produces a probing solution, it does not give a clue as to why and how. This reduces trust in the network.
- There is no specific rule for determining the structure of neural networks. Appropriate network structure is achieved through experience and trial and error.
- The network is reduced to a certain value of the error on the sample means that the training has been completed. This value does not give us optimum results.

# 3.6 K- Nearest Neighbors

- K- Nearest Neighbors or also known as K-NN belong to the family of supervised machine learning algorithms which means we use labeled (Target Variable) dataset to predict the class of new data point. The K-NN algorithm is a robust classifier which is often used as a benchmark for more complex classifiers such as Artificial Neural Network (ANN) or Support vector machine (SVM).
- K-NN algorithm is very simple to understand and equally easy to implement. To classify the new data point K-NN algorithm reads through whole dataset to find out K nearest neighbors.



- We use “**caret**” package in R to run the **train** function and specify the method as KNN to run the KNN model. **trainControl** function is used to iterate through the model to find the best K value to get the highest accuracy and tune the model.

```
library(caret)
trctrl <- trainControl(method = "repeatedcv", number = 4, repeats = 1, allowParallel = T)
set.seed(1)
start.time <- Sys.time()
fmnist_knn <- train(label ~., data = train.pca, method = "knn",
                    trControl=trctrl,preProcess = c("center", "scale"),
                    tuneLength = 3)
```

# 3.6 K- Nearest Neighbors

Advantages:

- K-NN does not explicitly build any model. New data entry would be tagged with majority class in the nearest neighbor.
- k-NN is a memory-based approach. The classifier immediately adapts as we collect new training data. It allows the algorithm to respond quickly to changes in the input during real-time use.
- Most of the classifier algorithms are easy to implement for binary problems and needs effort to implement for multi class whereas K-NN adjust to multi class without any extra efforts.
- K-NN can be used both for classification and regression problems.
- K-NN algorithm gives user the flexibility to choose distance while building K-NN model.

Disadvantages:

- As dataset grows efficiency or speed of the algorithm declines very fast.
- One of the biggest issues with K-NN is to choose the optimal number of neighbors to be consider while classifying the new data entry.
- K-NN algorithm is very sensitive to outliers as it simply chose the neighbors based on distance criteria.
- K-NN inherently has no capability of dealing with missing value problem.

# Part 4 : Implementation

## 4.1 Data Reduction

### 4.11 Zero Variance / Near Zero Variance pixel removal

### Interpretation

```
> discard
 [1] "pixel1"   "pixel2"   "pixel27"  "pixel28"  "pixel29"  "pixel30"  "pixel56"
 [8] "pixel57"  "pixel58"  "pixel84"  "pixel85"  "pixel113" "pixel757" "pixel758"
[15] "pixel784"
> cat(sum(nzrv$nzv), "near zero variance predictors have been removed,", "\n")
15 near zero variance predictors have been removed,
> cat(sum(nzrv$zeroVar), "of which were zero variance predictors.")
0 of which were zero variance predictors.
> 
```

Near Zero Variance Pixel Analysis
**Parameter used : nearZeroVar function result**
*15 near zero variance pixels found*
*0 absolute zero variance pixel found*

The 15 near zero variance pixels are mostly the corner pixels– pixel1, 2, 27,28, 758, 784, etc—and hence such pixel data will not help our model.
We thus remove them before performing Principal Component Analysis(PCA).

# 4.12 Principal Component Analysis

PCA Variance Analysis

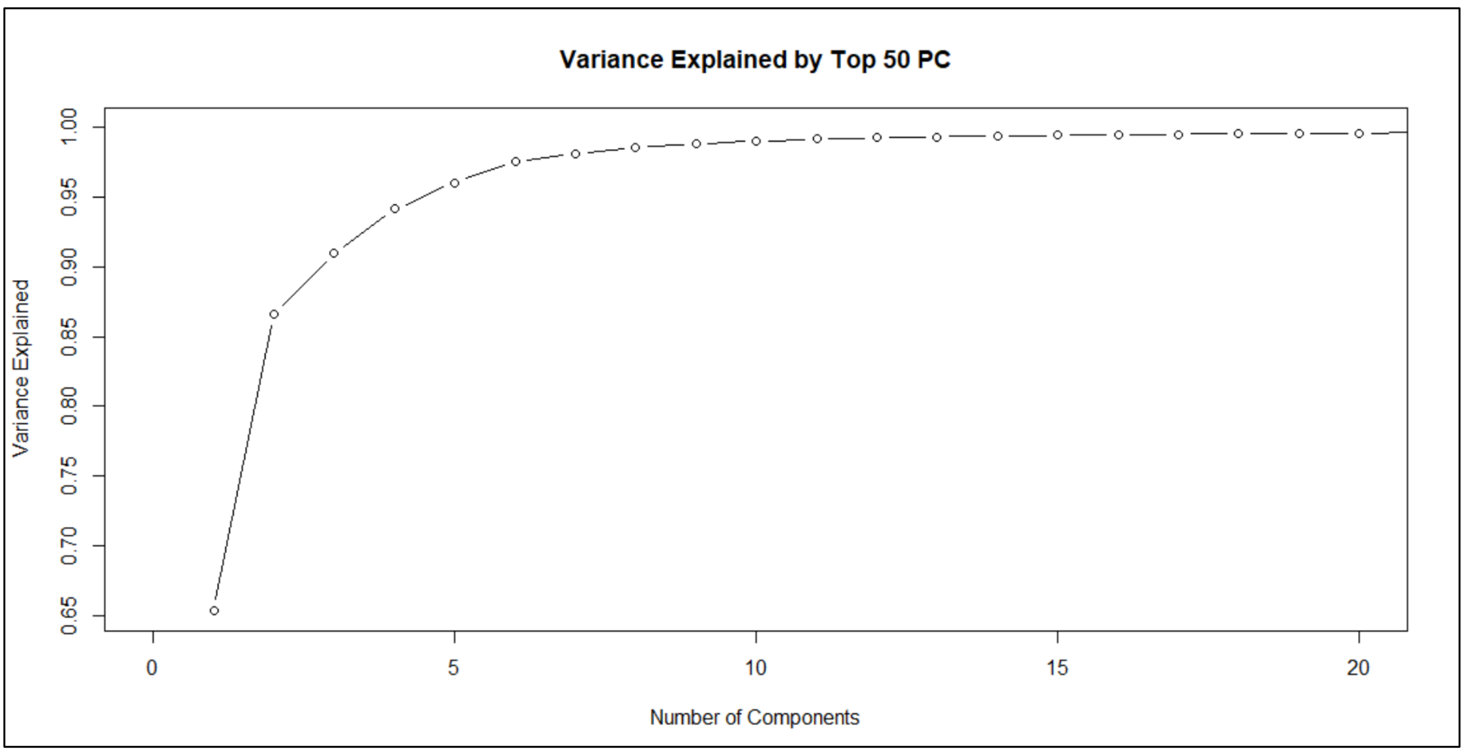
**Parameter used : Principal Component Summary Statistics**

*First 2 PCs cover 86.64% variance*

*First 10 PCs cover 99.05% variance*

*First 50 PCs cover 99.90% variance*

Variance plot reveals a clear elbow at p=5. Hence, we select the first two PC's and obtain a solution containing two out of the 768 dimensions. This is an exceptional reduction and a very interesting result as merely two directions are needed to describe almost 90% (86.64%) of the total variability in the data.




For model testing we will use the first 50 Principal Components which cover 99.9% variance because we are looking for higher accuracy in our models

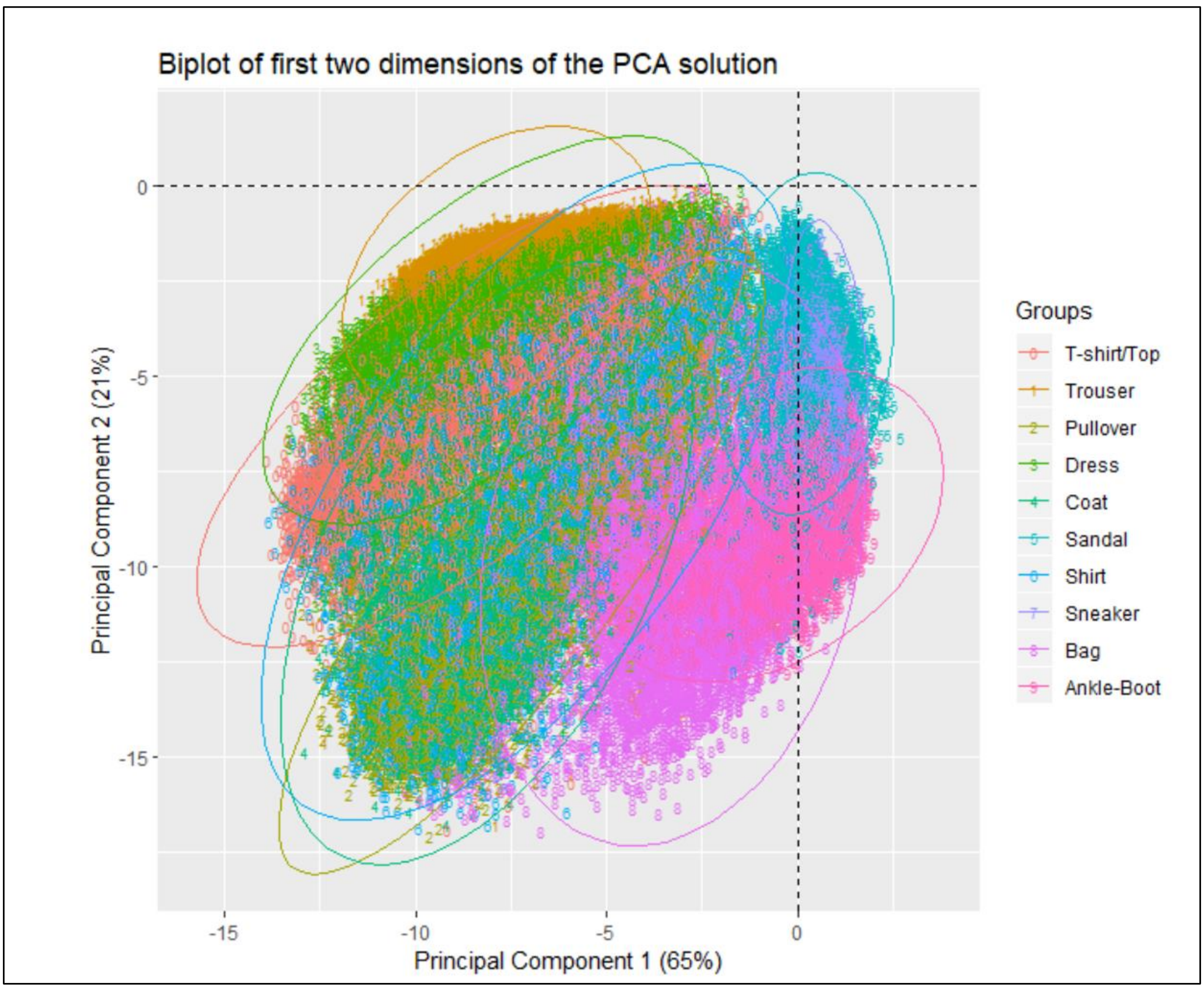
# 4.13 Principal Component Analysis Interpretation

PC1 and PC2 biplot interpretation

**Parameter used : PC1 and PC2 Biplot Color Grouping**

Some products are clustered together and some are distinguishable easily.

*Pullover and Coat* seem to be hardest to distinguish based on the first two dimensions as this group is the most scattered throughout the plot. These findings are intuitive. For instance, *sneakers* and *sandals* are harder to make apart then *bags* and *trousers*. Naturally, these difference are also also rooted in the underlying pixel data which is the variability exploited by PCA.



# Part 4.1 Data Reduction R-Code

```
#Data pre-processing
##Remove near zero variance variable
set.seed(1)
nzrv <- nearZeroVar(train[,-1], saveMetrics = T, freqCut = 300, uniqueCut = 1/4)
discard <- rownames(nzrv[nzrv$nzv,])
keep <- setdiff(names(train), discard)
trainnzv <- train[,keep]
cat(sum(nzrv$nzv), "near zero variance predictors have been removed,", "\n")
cat(sum(nzrv$zeroVar), "of which were zero variance predictors.")

###Scale all columns
label <- as.factor(trainnzv$label)
trainnzv$label <- NULL
trainnzv <- trainnzv / 255
```

**Remove near Zero Variance Dimensions On Training dataset + Data Scaling**

```
##Perform PCA to reduce dimensions
set.seed(1)
train.cov <- cov(trainnzv)
train.pc <- prcomp(train.cov)
options(max.print=1000000)
#summary(train.pc)
var.ex <- train.pc$sdev^2 / sum(train.pc$sdev^2)
var.cum <- cumsum(var.ex)
results <- data.frame(num <- 1:length(train.pc$sdev),ex = var.ex,cum = var.cum)

###Replace Train dataset
train.score <- as.matrix(trainnzv) %*% train.pc$rotation[,1:50]
train.pca <- cbind(label, as.data.frame(train.score))

###Principal Components Variance Plot
plot(results$num, results$cum, type = "b", xlim = c(0,20),
    main = "99.9% Variance Explained by Top 50 PC",
    xlab = "Number of Components", ylab = "Variance Explained")
#dev.off()
```

**Perform PCA And Plot PC vs Variance chart**

```
###Explore Train after PCA by creating PC1 and PC2 plot by groups
Groups = label.factor # define grouping variable
ggplot(data=train.pca, aes(x=train.pca$PC1, y=train.pca$PC2, colour=Groups, shape=Groups)) +
 geom_point(size=2) +
 theme(aspect.ratio=1) +
 scale_shape_manual(values=seq(48,57)) +
 labs(title="Biplot of first two dimensions of the PCA solution", x="Principal Component 1 (65%)",
     y="Principal Component 2 (21%)") + stat_ellipse(type="norm", level=0.95) +
 geom_vline(xintercept=c(-0,0), linetype="dashed", size=0.3) +
 geom_hline(yintercept=c(-0,0), linetype="dashed", size=0.3)
```

**PC1 and PC2 Biplot Color Grouping**

```
###Replace validation
label <- valid[,1]
validwolabels<- valid[,-1]
keep <- setdiff(names(validwolabels), discard)
validnzv <- validwolabels[,keep]
validnzv <- validnzv / 255
valid.score <- as.matrix(validnzv) %*% train.pc$rotation[,1:50]
valid.pca <- cbind(label, as.data.frame(valid.score))
valid.pca$label <- factor(valid.pca$label)
```

**Remove near Zero Variance Dimensions +PC transformation On Testing dataset**

# 4.2 Classification Models
# Accuracy and AUC Interpretation

**SVM model has the Highest Accuracy and Multiclass Area under curve**

Model-wise Performance
**Parameter used : Classification accuracy, Multiclass AUC and Time taken to run in seconds**
Highest accuracy is given by Support Vector Machine(SVM) model
Highest AUC is given by Support Vector Machine(SVM) model
Fastest model is Linear Discriminant Analysis(LDA) which takes less than 1 second to run.

Overall Interpretation
1. We can use LDA model for quick results.
2. We can use SVM model for more accurate results.
3. Neural Networks is the second best model but takes the highest time.
4. In the further sections, we will present SVM model results in detail

| model | accuracy_test | Multiclass AUC | SecsTaken_test |
|---|---|---|---|
| rf | 0.86 | 0.92 | 32 |
| svm | 0.90 | 0.94 | 642 |
| lda | 0.79 | 0.89 | 1 |
| nnet | 0.88 | 0.93 | 1126 |
| knn | 0.86 | 0.92 | 345 |

# Part 4.2 Models & Accuracy R-Code

## Random Forest, SVM and LDA

```
#Start running Models###########

##Create dataframe to store model results
model.accuracytest<- setNames(data.frame(matrix(ncol = 4, nrow = 0)),
                     c("model", "accuracy_test","Multiclass AUC","SecsTaken_test"))
```

**Run Random Forest algorithm, predict and make Confusion Matrix**

```
library(pROC)
##Model with Random forest on PCA Data
library(randomForest)
set.seed(1)
start.time <- Sys.time()
fmnist_rf=randomForest(label~. ,data = train.pca, ntree=56)
t <- Sys.time() - start.time

predrf <- predict(fmnist_rf,valid.pca)
rf_cm <- confusionMatrix(predrf,
                  valid.pca$label,
                  dnn = c("RF-Predicted", "Actual"))

###Compute Multi-class area under the curve
roc_df <- multiclass.roc(predrf,as.numeric(valid.pca$label))
model.accuracytest['rf',] <- c('rf', rf_cm$overall[1],roc_df$auc[1],as.numeric(t, units = "secs"))
```

**Run SVM algorithm, predict and make Confusion Matrix**

```
##Model with SVM on PCA Data
library(e1071)

set.seed(1)
start.time <- Sys.time()
fmnist_svm <- svm(label ~ ., data=train.pca)
t<- Sys.time() - start.time

pred_svm <- predict(fmnist_svm, valid.pca)
svm_cm <- confusionMatrix(pred_svm,
                   valid.pca$label,
                   dnn = c("SVM-Predicted", "Actual"))

###Compute Multi-class area under the curve
roc_svm <- multiclass.roc(pred_svm,as.numeric(valid.pca$label))
model.accuracytest['svm',] <- c('svm', svm_cm$overall[1],roc_svm$auc[1],as.numeric(t, units = "secs"))
```

**Run LDA algorithm, predict and make Confusion Matrix**

```
##Model with LDA on PCA Data
library(MASS)

set.seed(1)
start.time <- Sys.time()
fmnist_lda <- lda(label~.,data = train.pca)
t<- Sys.time() - start.time

pred_lda <- predict(fmnist_lda, valid.pca)
lda_cm <- confusionMatrix(pred_lda$class,
                   valid.pca$label,
                   dnn = c("LDA-Predicted", "Actual"))

###Compute Multi-class area under the curve
roc_lda <- multiclass.roc(pred_lda$class,as.numeric(valid.pca$label))
model.accuracytest['lda',] <- c('lda', lda_cm$overall[1],roc_lda$auc[1],as.numeric(t, units = "secs"))
```

# Part 4.2 Models & Accuracy R-Code

## Neural Network, k-nearest neighbours and model comparison table

```
##Model with Neural Network--nnet--on PCA Data
###We choose no_of_nodes=120 and maxiter=130 after NNET tuning on caret package
library(nnet)
set.seed(1)
n <- names(train.pca[,-1])
f <- as.formula(paste("label ~", paste(n[!n %in% "medv"], collapse = " + ")))
start.time <- Sys.time()
fmnist_nnet <- nnet(f,data = train.pca,
            size=120,maxit=130,MaxNWts = 80000)
t<- Sys.time() - start.time

library(NeuralNetTools)
plotnet(fmnist_nnet,skip = TRUE)

pred_nnet <- predict(fmnist_nnet,valid.pca,type="class")
nnet_cm <- confusionMatrix(factor(pred_nnet),
               valid.pca$label,
               dnn = c("nnet-Predicted", "Actual"))

###Compute Multi-class area under the curve
roc_nnet <- multiclass.roc(pred_nnet,as.numeric(valid.pca$label))
model.accuracytest['nnet',] <- c('nnet', nnet_cm$overall[1],roc_nnet$auc[1],as.numeric(t, units = "secs"))
```

**Run Neural Network algorithm, predict and make Confusion Matrix**

```
##Model with k-nearest neighbours on PCA Data
library(caret)
library('e1071')
###Enable Parallel processing
library(doParallel)
cl <- parallel::makeCluster(detectCores(logical=FALSE), type='PSOCK')
doParallel::registerDoParallel(cl)

trctrl <- trainControl(method = "repeatedcv", number = 4, repeats = 1, allowParallel = T)
set.seed(1)
start.time <- Sys.time()
fmnist_knn <- train(label ~., data = train.pca, method = "knn",
            trControl=trctrl,preProcess = c("center", "scale"),
            tuneLength = 3)
t<- Sys.time() - start.time

#fmnist_knn
#plot(fmnist_knn)
pred_knn <- predict(fmnist_knn, newdata = valid.pca)
knn_cm <- confusionMatrix(pred_knn,
               valid.pca$label,
               dnn = c("knn-Predicted", "Actual"))

parallel::stopCluster(cl)
###Compute Multi-class area under the curve
roc_knn <- multiclass.roc(pred_knn,as.numeric(valid.pca$label))
model.accuracytest['knn',] <- c('knn', knn_cm$overall[1],roc_knn$auc[1],as.numeric(t, units = "secs"))
```

**Run knn algorithm, predict and make Confusion Matrix**

```
#Rounded off all model results
model_accuracyTest<- model.accuracytest
model_accuracyTest$accuracy_test <- round(as.numeric(model.accuracytest$accuracy_test), digits = 2)
model_accuracyTest$`Multiclass AUC` <- round(as.numeric(model.accuracytest$`Multiclass AUC`), digits = 2)
model_accuracyTest$SecsTaken_test <- round(as.numeric(model.accuracytest$SecsTaken_test), digits = 0)
model_accuracyTest
```

**Rounded off Model Comparison table**

# Part 5 : Results & Performance

## 5.1 Confusion Matrix Interpretation

**SVM model has the Highest overall Accuracy**

Class-wise Accuracy

**Parameter used : Sensitivity**

Highest accuracy in predicting Class 8(Bag) followed by Class 1(Trouser), 9(Ankle-Boot) and 5(Sandal).

Lowest accuracy in predicting Class 6(Shirt) followed by Class 2(Pullover), 4(Coat) and 0(T-Shirt/Top).

Misclassification

**Parameter used : Confusion Matrix Actual vs Predicted**

Shirt was mostly misclassified as T-Shirt/Top

Pullover was mostly misclassified as Coat

Coat was mostly misclassified as Shirt and Pullover

T-Shirt/Top was mostly misclassified as Shirt

## 5.11 Confusion Matrix of SVM

```
> svm_cm
Confusion Matrix and Statistics

             Actual
SVM-Predicted   0   1   2   3   4   5   6   7   8   9
            0 865   5  10  20   1   0 138   0   1   0
            1   0 975   0   3   0   0   0   0   0   0
            2  13   2 818  10  53   0  69   0   4   0
            3  28  14  13 922  25   0  21   0   4   0
            4   0   0  87  24 862   0  56   0   0   0
            5   0   1   1   0   0 947   0  16   2   7
            6  86   2  66  20  57   0 706   0   5   0
            7   0   0   0   0   0  37   0 950   1  34
            8   8   1   5   1   2   4  10   0 982   1
            9   0   0   0   0   0  12   0  34   1 958

Overall Statistics

               Accuracy : 0.8985
                 95% CI : (0.8924, 0.9044)
    No Information Rate : 0.1
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8872
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8 Class: 9
Sensitivity            0.8650   0.9750   0.8180   0.9220   0.8620   0.9470   0.7060   0.9500   0.9820   0.9580
Specificity            0.9806   0.9997   0.9832   0.9883   0.9814   0.9970   0.9738   0.9920   0.9964   0.9948
Pos Pred Value         0.8317   0.9969   0.8442   0.8978   0.8377   0.9723   0.7495   0.9295   0.9684   0.9532
Neg Pred Value         0.9849   0.9972   0.9798   0.9913   0.9846   0.9941   0.9675   0.9944   0.9980   0.9953
Prevalence             0.1000   0.1000   0.1000   0.1000   0.1000   0.1000   0.1000   0.1000   0.1000   0.1000
Detection Rate         0.0865   0.0975   0.0818   0.0922   0.0862   0.0947   0.0706   0.0950   0.0982   0.0958
Detection Prevalence   0.1040   0.0978   0.0969   0.1027   0.1029   0.0974   0.0942   0.1022   0.1014   0.1005
Balanced Accuracy      0.9228   0.9873   0.9006   0.9552   0.9217   0.9720   0.8399   0.9710   0.9892   0.9764
```

# 5.12 Confusion Matrix of Random Forest and LDA

```
> rf_cm
Confusion Matrix and Statistics

            Actual
RF-Predicted   0   1   2   3   4   5   6   7   8   9
           0 842   4  14  30   0   1 182   0   1   0
           1   0 962   0   6   0   0   2   0   0   0
           2  15   7 793  14  69   0  96   0   9   0
           3  34  21  11 898  33   0  21   0   3   0
           4   2   0 104  29 832   0  81   0   3   0
           5   2   0   0   0   0 906   0  37   6  26
           6  92   6  67  22  61   1 599   0   7   0
           7   1   0   0   0   0  59   0 899   4  39
           8  12   0  11   1   5   5  19   0 965   0
           9   0   0   0   0   0  28   0  64   2 935

Overall Statistics

               Accuracy : 0.8631
                 95% CI : (0.8562, 0.8698)
    No Information Rate : 0.1
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8479
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8 Class: 9
Sensitivity            0.8420   0.9620   0.7930   0.8980   0.8320   0.9060   0.5990   0.8990   0.9650   0.9350
Specificity            0.9742   0.9991   0.9767   0.9863   0.9757   0.9921   0.9716   0.9886   0.9941   0.9896
Pos Pred Value         0.7840   0.9918   0.7906   0.8795   0.7916   0.9273   0.7006   0.8972   0.9479   0.9086
Neg Pred Value         0.9823   0.9958   0.9770   0.9886   0.9812   0.9896   0.9562   0.9888   0.9961   0.9928
Prevalence             0.1000   0.1000   0.1000   0.1000   0.1000   0.1000   0.1000   0.1000   0.1000   0.1000
Detection Rate         0.0842   0.0962   0.0793   0.0898   0.0832   0.0906   0.0599   0.0899   0.0965   0.0935
Detection Prevalence   0.1074   0.0970   0.1003   0.1021   0.1051   0.0977   0.0855   0.1002   0.1018   0.1029
Balanced Accuracy      0.9081   0.9806   0.8848   0.9422   0.9038   0.9491   0.7853   0.9438   0.9796   0.9623
```

```
> lda_cm
Confusion Matrix and Statistics

             Actual
LDA-Predicted   0   1   2   3   4   5   6   7   8   9
            0 763   1  16  32   0   0 184   0   1   0
            1   0 923   0   3   1   0   0   0   0   0
            2  17  16 649   9  93   0 108   0   6   0
            3  86  43   7 846  29   1  44   0  17   0
            4   1   2 173  36 762   0 108   0   4   0
            5   6   1   1   2   1 870   6  99  17  43
            6  98  14 145  71 112   3 530   0  44   0
            7   1   0   0   0   0  84   0 795   4  54
            8  28   0   9   1   2   7  20   0 906   0
            9   0   0   0   0   0  35   0 106   1 903

Overall Statistics

               Accuracy : 0.7947
                 95% CI : (0.7866, 0.8026)
    No Information Rate : 0.1
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7719
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8 Class: 9
Sensitivity            0.7630   0.9230   0.6490   0.8460   0.7620   0.8700   0.5300   0.7950   0.9060   0.9030
Specificity            0.9740   0.9996   0.9723   0.9748   0.9640   0.9804   0.9459   0.9841   0.9926   0.9842
Pos Pred Value         0.7653   0.9957   0.7227   0.7884   0.7017   0.8317   0.5211   0.8475   0.9311   0.8641
Neg Pred Value         0.9737   0.9915   0.9614   0.9827   0.9733   0.9855   0.9477   0.9774   0.9896   0.9892
Prevalence             0.1000   0.1000   0.1000   0.1000   0.1000   0.1000   0.1000   0.1000   0.1000   0.1000
Detection Rate         0.0763   0.0923   0.0649   0.0846   0.0762   0.0870   0.0530   0.0795   0.0906   0.0903
Detection Prevalence   0.0997   0.0927   0.0898   0.1073   0.1086   0.1046   0.1017   0.0938   0.0973   0.1045
Balanced Accuracy      0.8685   0.9613   0.8107   0.9104   0.8630   0.9252   0.7379   0.8896   0.9493   0.9436
```

# 5.13 Confusion Matrix of Neural Network and KNN

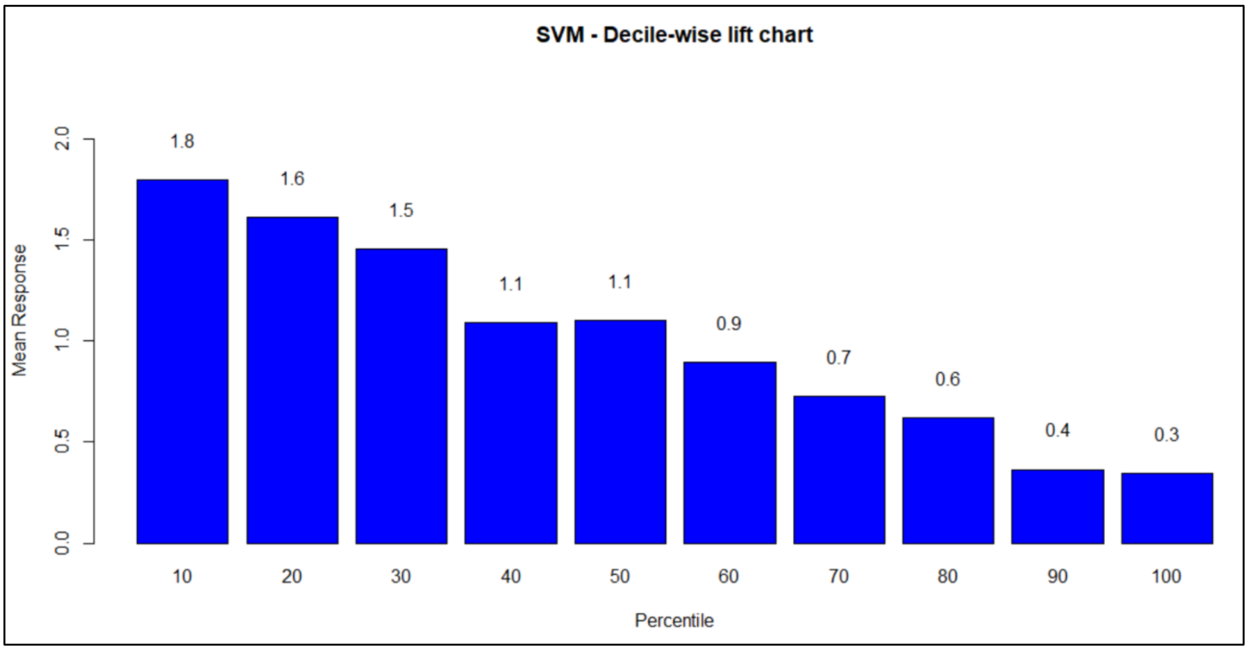
```
> nnet_cm
Confusion Matrix and Statistics

              Actual
nnet-Predicted   0   1   2   3   4   5   6   7   8   9
             0 842   0  10  22   1   0 134   0   4   0
             1   3 983   0   8   0   0   2   0   1   0
             2  16   2 792   6  64   0  67   0   4   0
             3  18  10   7 907  27   0  26   0   2   0
             4   2   1 102  26 840   0  72   0   1   0
             5   1   2   0   1   1 936   0  24   3   9
             6 109   2  86  25  64   0 688   0   7   0
             7   0   0   0   0   0  43   0 936   1  43
             8   9   0   3   4   3   5  11   1 977   1
             9   0   0   0   1   0  16   0  39   0 947

Overall Statistics

               Accuracy : 0.8848
                 95% CI : (0.8784, 0.891)
    No Information Rate : 0.1
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.872
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8 Class: 9
Sensitivity            0.8420   0.9830   0.7920   0.9070   0.8400   0.9360   0.6880   0.9360   0.9770   0.9470
Specificity            0.9810   0.9984   0.9823   0.9900   0.9773   0.9954   0.9674   0.9903   0.9959   0.9938
Pos Pred Value         0.8312   0.9860   0.8328   0.9097   0.8046   0.9580   0.7013   0.9150   0.9635   0.9442
Neg Pred Value         0.9824   0.9981   0.9770   0.9897   0.9821   0.9929   0.9654   0.9929   0.9974   0.9941
Prevalence             0.1000   0.1000   0.1000   0.1000   0.1000   0.1000   0.1000   0.1000   0.1000   0.1000
Detection Rate         0.0842   0.0983   0.0792   0.0907   0.0840   0.0936   0.0688   0.0936   0.0977   0.0947
Detection Prevalence   0.1013   0.0997   0.0951   0.0997   0.1044   0.0977   0.0981   0.1023   0.1014   0.1003
Balanced Accuracy      0.9115   0.9907   0.8872   0.9485   0.9087   0.9657   0.8277   0.9632   0.9864   0.9704
```

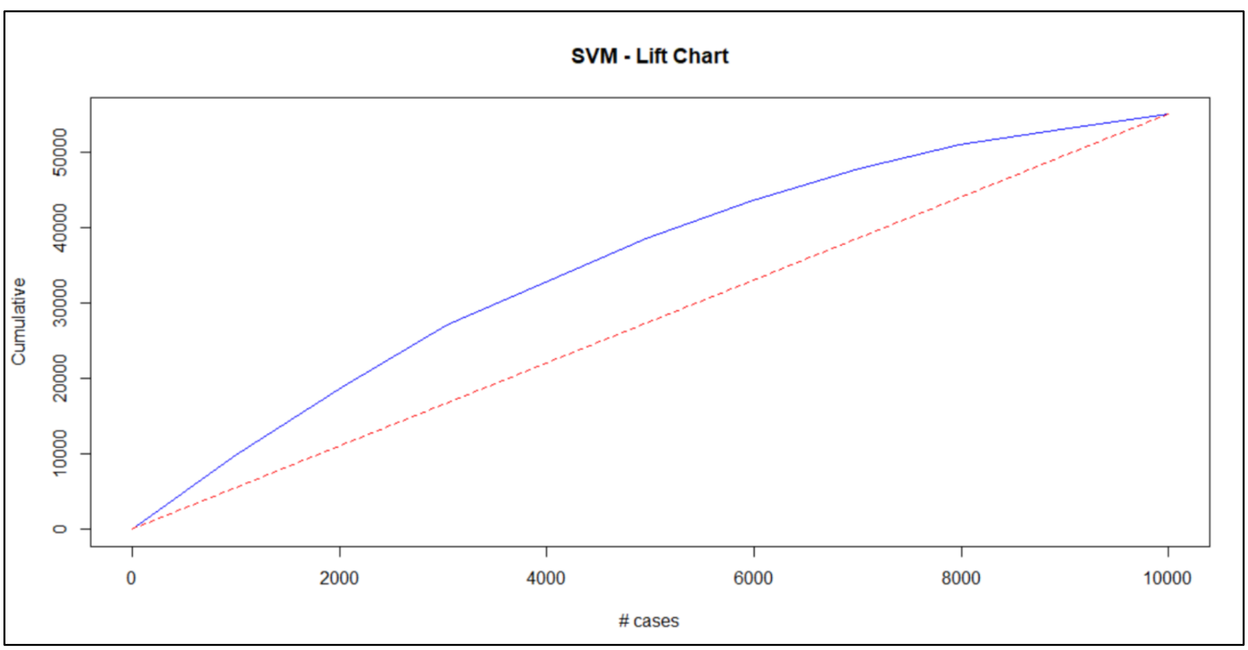
```
> knn_cm
Confusion Matrix and Statistics

             Actual
knn-Predicted   0   1   2   3   4   5   6   7   8   9
            0 836   4  25  28   4   2 183   0   5   0
            1   1 976   0   4   1   0   3   0   2   0
            2  14   1 769  10  67   1  70   0   7   0
            3  15  11  17 895  27   2  22   0   4   0
            4   6   0 105  36 801   0  71   0   3   0
            5   0   1   1   0   0 867   1  10   4   4
            6 115   6  76  26  99   3 638   1   4   0
            7   5   0   4   1   0  80   1 931   8  33
            8   7   1   3   0   1   2  11   1 961   0
            9   1   0   0   0   0  43   0  57   2 963

Overall Statistics

               Accuracy : 0.8637
                 95% CI : (0.8568, 0.8704)
    No Information Rate : 0.1
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8486
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8 Class: 9
Sensitivity            0.8360   0.9760   0.7690   0.8950   0.8010   0.8670   0.6380   0.9310   0.9610   0.9630
Specificity            0.9721   0.9988   0.9811   0.9891   0.9754   0.9977   0.9633   0.9853   0.9971   0.9886
Pos Pred Value         0.7691   0.9889   0.8190   0.9013   0.7838   0.9764   0.6591   0.8758   0.9737   0.9034
Neg Pred Value         0.9816   0.9973   0.9745   0.9883   0.9778   0.9854   0.9599   0.9923   0.9957   0.9959
Prevalence             0.1000   0.1000   0.1000   0.1000   0.1000   0.1000   0.1000   0.1000   0.1000   0.1000
Detection Rate         0.0836   0.0976   0.0769   0.0895   0.0801   0.0867   0.0638   0.0931   0.0961   0.0963
Detection Prevalence   0.1087   0.0987   0.0939   0.0993   0.1022   0.0888   0.0968   0.1063   0.0987   0.1066
Balanced Accuracy      0.9041   0.9874   0.8751   0.9421   0.8882   0.9323   0.8007   0.9582   0.9791   0.9758
```

# 5.2 Lift and Decile-wise Chart Interpretation

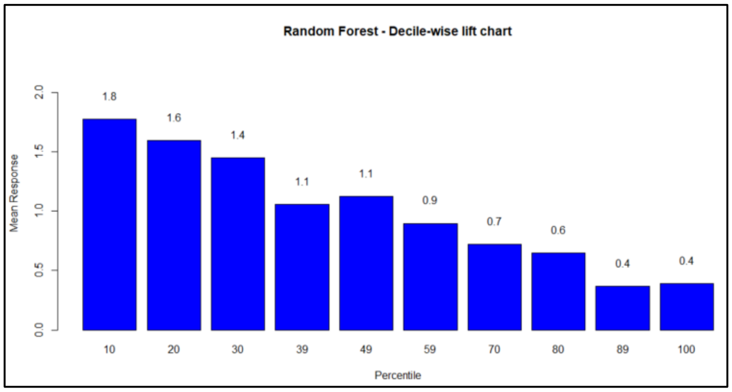
Cumulative Lift of 1.8 for top 10 percentile, means that when selecting 10% of the records based on the model, one can expect 1.8 times the total number of targets found by randomly selecting 10% of images without a model. The decile chart indicates that we can use the model to select the top 10% records with the highest propensities
and still perform almost twice as well as random.

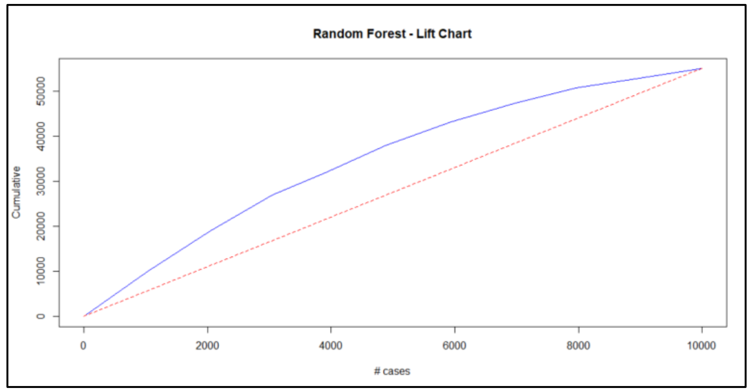
*Since we are not interested in a Particular Class, Lift and Decile-wise charts do not help us so much.*

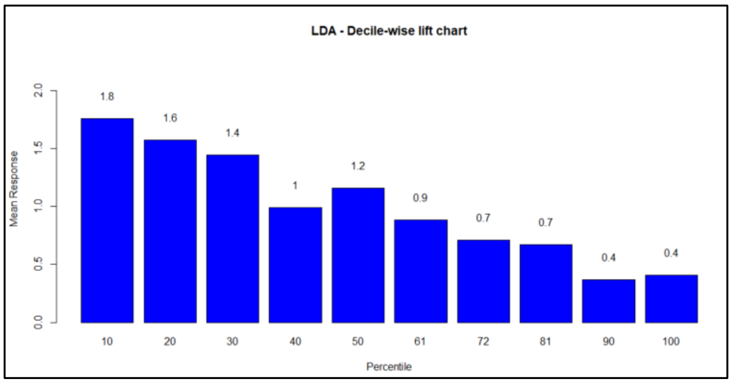


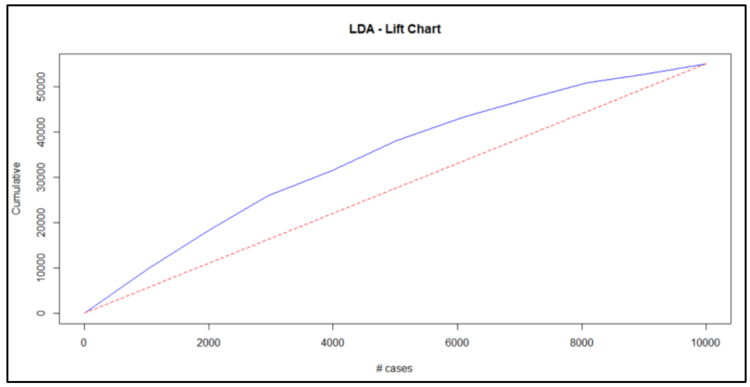



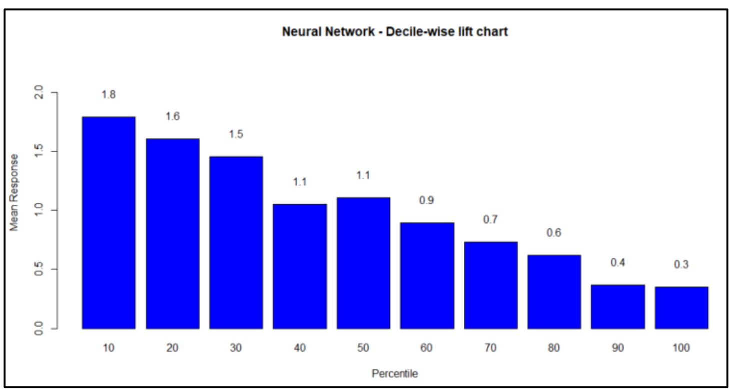
# 5.2 Lift and Decile-wise Chart Interpretation

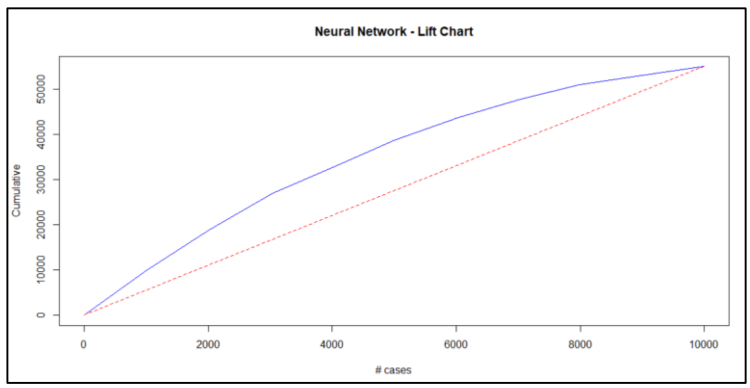


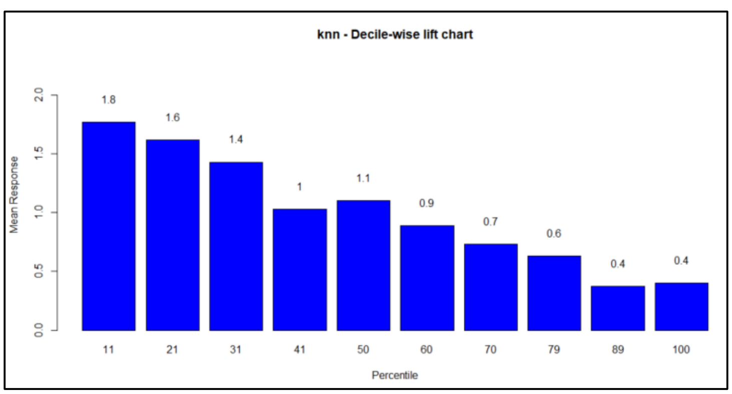



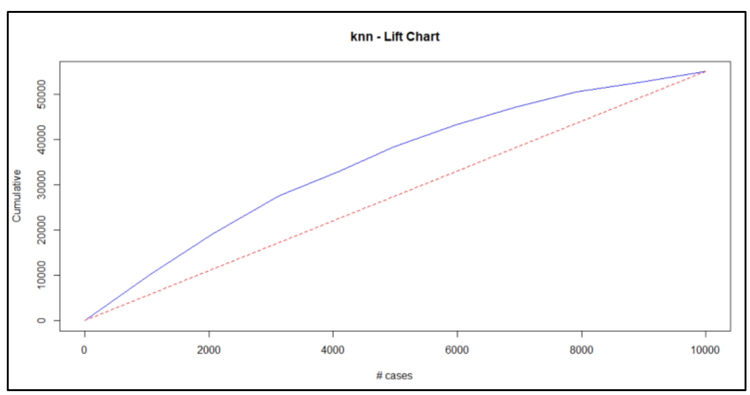



# Part 5.2 Results & Performance R-Code

## Lift and Decile-wise Chart for Random Forest, LDA and Neural Network

```
#Lift-chart and Decile chart
##Lift-chart and Decile chart Random Forest
actual = as.numeric(valid.pca$label)
predicted = as.numeric(predrf)
gain <- gains(actual, predicted, groups=10)
###Lift-Chart
plot(c(0, gain$cume.pct.of.total*sum(actual)) ~ c(0, gain$cume.obs),xlab = "# cases",
ylab = "Cumulative", type="l",col = "blue",main = "Random Forest - Lift Chart")
lines(c(0,sum(actual))~c(0,dim(valid.pca)[1]), col="red", lty=2)
###Gain-chart
heights <- gain$mean.resp/mean(actual)
midpoints <- barplot(heights, names.arg = gain$depth, ylim = c(0 ,2.3),
                xlab = "Percentile", ylab = "Mean Response",
                main = "Random Forest - Decile-wise lift chart",col = "blue")
text(midpoints, heights+0.2, labels=round(heights, 1), cex = 1)
```

**Life and Decile-wise chart for Random Forest**

```
##Lift-chart and Decile chart LDA
predicted = as.numeric(pred_lda$class)
gain <- gains(actual, predicted, groups=10)
###Lift-Chart
plot(c(0, gain$cume.pct.of.total*sum(actual)) ~ c(0, gain$cume.obs),
    xlab = "# cases", ylab = "Cumulative", type="l",col = "blue",main = "LDA - Lift Chart")
lines(c(0,sum(actual))~c(0,dim(valid.pca)[1]), col="red", lty=2)
###Gain-chart
heights <- gain$mean.resp/mean(actual)
midpoints <- barplot(heights, names.arg = gain$depth, ylim = c(0 ,2.3),
                xlab = "Percentile", ylab = "Mean Response",
                main = "LDA - Decile-wise lift chart",col = "blue")
text(midpoints, heights+0.2, labels=round(heights, 1), cex = 1)
```

**Life and Decile-wise chart for LDA**

```
##Lift-chart and Decile chart Neural Network
predicted = as.numeric(pred_nnet)
gain <- gains(actual, predicted, groups=10)
###Lift-Chart
plot(c(0, gain$cume.pct.of.total*sum(actual)) ~ c(0, gain$cume.obs),
    xlab = "# cases", ylab = "Cumulative", type="l",col = "blue",
    main = "Neural Network - Lift Chart")
lines(c(0,sum(actual))~c(0,dim(valid.pca)[1]), col="red", lty=2)
###Gain-chart
heights <- gain$mean.resp/mean(actual)
midpoints <- barplot(heights, names.arg = gain$depth, ylim = c(0 ,2.3),
                xlab = "Percentile", ylab = "Mean Response",
                main = "Neural Network - Decile-wise lift chart",col = "blue")
text(midpoints, heights+0.2, labels=round(heights, 1), cex = 1)
```

**Life and Decile-wise chart for Neural Network**

# Part 5.2 Results & Performance R-Code

## Lift and Decile Chart for knn and SVM

```
#Lift-chart and Decile chart

##Lift-chart and Decile chart k-Nearest Neighbours
predicted = as.numeric(pred_knn)
gain <- gains(actual, predicted, groups=10)
###Lift-Chart
plot(c(0, gain$cume.pct.of.total*sum(actual)) ~ c(0, gain$cume.obs),
    xlab = "# cases", ylab = "Cumulative", type="l",col = "blue",
    main = "knn - Lift Chart")
lines(c(0,sum(actual))~c(0,dim(valid.pca)[1]), col="red", lty=2)

###Gain-chart
heights <- gain$mean.resp/mean(actual)
midpoints <- barplot(heights, names.arg = gain$depth, ylim = c(0 ,2.3),
                 xlab = "Percentile", ylab = "Mean Response",
                 main = "knn - Decile-wise lift chart",col = "blue")
text(midpoints, heights+0.2, labels=round(heights, 1), cex = 1)
```

**Life and Decile-wise chart for knn**

```
##Lift-chart and Decile chart SVM
predicted = as.numeric(pred_svm)
gain <- gains(actual, predicted, groups=10)
###Lift-Chart
plot(c(0, gain$cume.pct.of.total*sum(actual)) ~ c(0, gain$cume.obs),
    xlab = "# cases", ylab = "Cumulative", type="l",col = "blue",
    main = "SVM - Lift Chart")
lines(c(0,sum(actual))~c(0,dim(valid.pca)[1]), col="red", lty=2)

###Gain-chart
heights <- gain$mean.resp/mean(actual)
midpoints <- barplot(heights, names.arg = gain$depth, ylim = c(0 ,2.3),
                 xlab = "Percentile", ylab = "Mean Response",
                 main = "SVM - Decile-wise lift chart",
                 col = "blue")
text(midpoints, heights+0.2, labels=round(heights, 1), cex = 1)
```

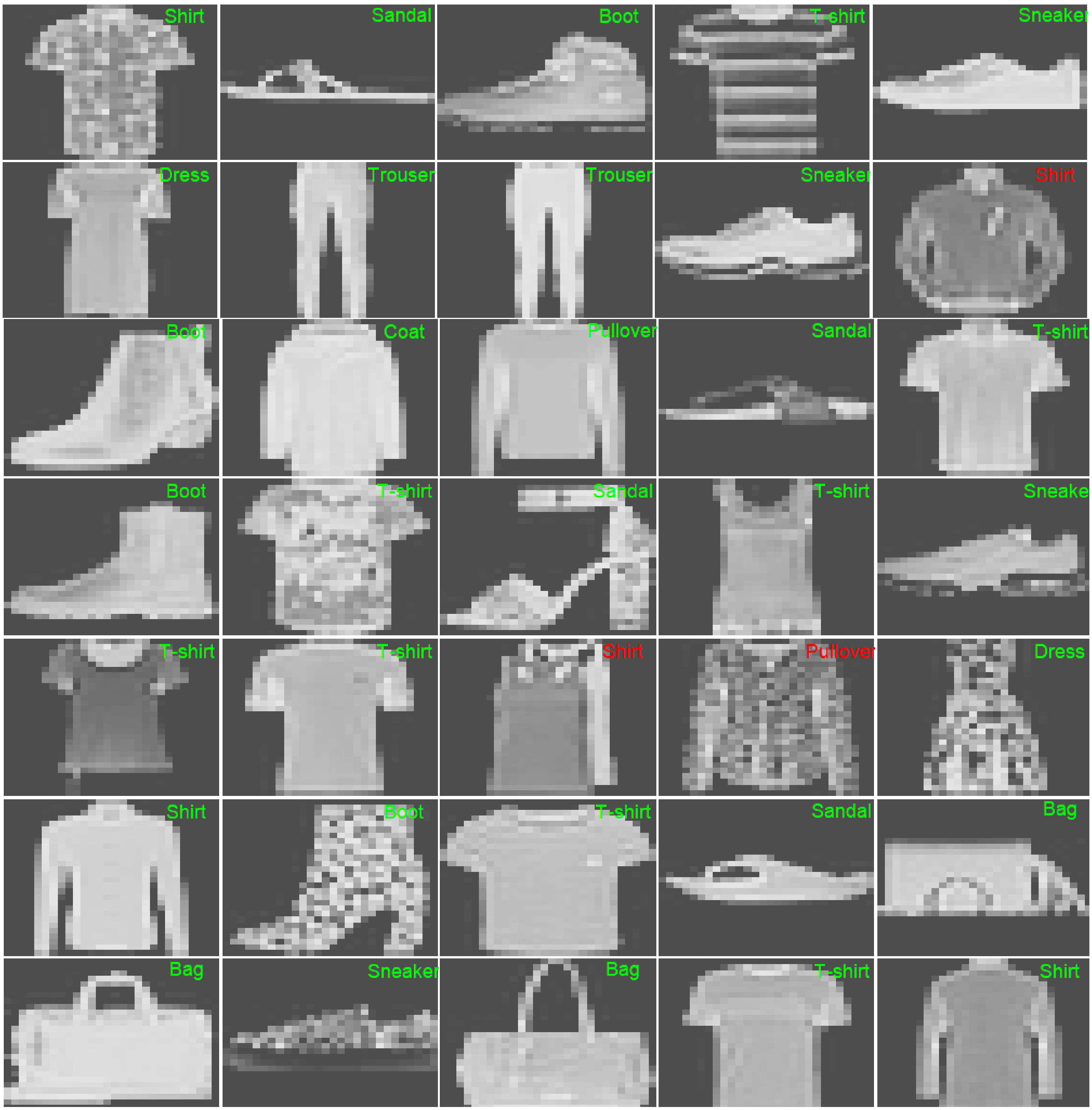
**Life and Decile-wise chart for SVM**

# Part 6 : SVM model Visual Results

Let's select a random sample of observations from Test / Validation dataset , classify them using our SVM model and turn the results into images.

Overall Observation

1. Pullover, Coat and T-shirt is mostly misclassified.



# SVM model Visual Results

# Part 6 Visual Results R-Code

## Visual best model(SVM) Results

```
#########
####################VISUALIZE PREDICTIONS#####################3
#Visualize predictions with actual digits using SVM model
dev.off()
svm.preds <- predict(fmnist_svm, valid.pca, array.layout = "colmajor")
#svm.preds[1:5]
label.name = svm.preds
categories = c("T-shirt", "Trouser", "Pullover", "Dress", "Coat",
            "Sandal", "Shirt", "Sneaker", "Bag", "Boot")
levels(label.name) = categories # add category column (character)



##Plot images with green as correctly and red as wrong predicted class
validoriginal <- data.matrix(valid)
validrep<- validoriginal
validrep[,-1] <- validrep[,-1]/255

plotResults <- function(images, preds, name){
  op <- par(no.readonly=TRUE)
  x <- ceiling(sqrt(length(images)))
  par(mfrow=c(x,x), mar=c(.1,.1,.1,.1))

  for (i in images){
    m <- matrix(validrep[i,-1], nrow=28, byrow=TRUE)
    m <- apply(m, 2, rev)
    image(t(m), col=grey.colors(255), axes=FALSE)
    text(0.86,0.95,
       col=ifelse(preds[i]==validrep[i,1],"green","red"),
       cex=1.7, name[i])
  }
  par(op)
}

plotResults(sample(1:length(svm.preds), 25, replace=F),
              svm.preds, label.name)
```

**Visualise SVM model prediction. Label is Green when correctly identified and Red otherwise.**

```
#Create a loop function for random images every sec seconds
loop <- function(sec){
  i = 1
  while(TRUE){
    if (i %% 8 == 0){
      break          #A condition to break out of the loop
    }
    plotResults(sample(1:length(svm.preds), 9, replace=F),
            svm.preds, label.name)          #Run your code
    Sys.sleep(time = sec) #Time in seconds

    i = i + 1
  }
}

#run loop function

loop(sec=2)
```

**Create a loop to visualise SVM model prediction multiple times every 2 seconds**

# Citations and References


*Book*

- Galit Shmueli, Peter Bruce, Inbal Yahav, Nitin Patel, and Kenneth Lichtendahl. *Data Mining for Business Analytics: Concepts, Techniques, and Applications in R, 1st edition, 2018.* Wiley.

*Company Website*

- Chris Tufts 2015. Classification Model Pros and Cons. https://github.com/ctufts/Cheat_Sheets/wiki/Classification-Model-Pros-and-Cons
- HackingNote 2018. Machine Learning Algorithms Pros and Cons. https://www.hackingnote.com/en/machine-learning/algorithms-pros-and-cons/
- Genesis September 2018. Pros and Cons of K-Nearest Neighbors. https://www.fromthegenesis.com/pros-and-cons-of-k-nearest-neighbors/
- Mapt beta. Pros and cons of neural networks. https://www.packtpub.com/mapt/book/big_data_and_business_intelligence/9781788397872/1/ch01lvl1sec27/pros-and-cons-of-neural-networks
- Djmw 2004. Feedforward neural networks 1. What is a feedforward neural network?. http://www.fon.hum.uva.nl/praat/manual/Feedforward_neural_networks_1__What_is_a_feedforward_ne.html
- Czar Yobero 2017. Linear Discriminant Analysis: A Brief Tutorial. https://rpubs.com/cyobero/lda
- Tom Bromley, Quantum researcher at Xanadu Feb 20, Making a Neural Network, Quantum https://medium.com/xanaduai/making-a-neural-network-quantum-34069e284bcf


------End of Report------