**California State University, East Bay**

# 

# Capstone Project

**Quality Control in Manufacturing Process**

**Trung Phan,**

**Duy Ly,**

**Thao Dinh,**

**Baoanh Phan**

**Professor Jiming Wu**

**ITM 6899**

**November 2018**

**EXECUTIVE SUMMARY**

**OBJECTIVE**

In this era of big data, many researchers and companies are harnessing and experimenting with various methods of extracting value from big data. In this paper, we used the data collected from a manufacturing company to build different machine learning models, then apply them to predict the quality of their products on production. The datasets include materials, material attributes, machine parameters, etc. From the results, we composed a report to outline our findings.

**METHODS**

To predict the good quality product from manufacturing dataset, we ran a series of analytic tests on two sets of data including the train data and the test data. In the report, we have created four common models, which are Neural Network (H2O), Naive Bayes, Decision Tree and Support Vector Machine (SVM), for each of the dataset to perform our analysis.

**FINDINGS**

After performing classification on training and testing data set with four algorithms: Deep Learning Neural Network H20, Naive Bayes, Decision Tree, and SVM, we found out that Decision Tree returned the highest accuracy with 90.42% on testing data part of *param\_data\_train* and Naïve Bayes gave the highest proportion of good product prediction with 99.89% on *param\_data\_test*.

**1. Introduction:**

With a highly competitive market due to the growth of global demand for goods and services, quality control becomes an essential and crucial part in any production process. To ensure that the products can be produced comply with strict requirements, manufacturers or regulatory authorities increasingly rely on several techniques during manufacturing process. Among those are quality control products during manufacturing by classifying production outcome based upon input information. This approach is to determine the quality of the product and allow any necessary changes to be addressed early on. To understand in-depth about manufacturing process and how quality control with classification can be applied, let’s take a quick look at their definition and explanation.

What is manufacturing process? As a Wikipedia definition, manufacturing process is defined as “the steps through which raw materials are transformed into a final product. The manufacturing process begins with the product design, and materials specification from which the product is made. These materials are then modified through manufacturing processes to become the required part.”

How does quality control in manufacturing process with classification work? Specifically, quality control in manufacturing process with classification is a simple approach to detect a good or bad quality batch production with historical data from similar previous batch production. These archived information included input raw material, ratio of raw material in use, or material brand names in last production process. The information then will be used as a training data set to establish a model classifier. For each algorithms or techniques used in creating a classifier, the prediction output may be different. Particularly, in this project, we applied 4 different algorithms for classification. Details of these algorithms will be explained deeper in the next section Algorithms and Techniques. The model classifier will be used to predict the quality of current batch production and provide or make appropriate decision for next stage in production process.

**2. Algorithm and Techniques:**

As mentioned previously, we applied 4 common various algorithms to establish a model classifier for our project. They are Naive Bayes, Decision Tree C5.0, Support Vector Machine SVM, and Deep Learning Neural Network H2O.

**2.4 Deep Learning Neural Network H2O**

H2O’s Deep Learning is based on a multi-layer feedforward artificial neural network that is trained with stochastic gradient descent using back-propagation. The network can contain a large number of hidden layers consisting of neurons with tanh, rectifier, and maxout activation functions. In our project, we also split out data to create validation set during training process.

**2.1 Naive Bayes**

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels as vectors of feature values. These class labels correspond to those drawn from training set. The classifiers are a gathering of many algorithms that are based on the same common principle: given the class variable, all naive Bayes classifiers assume that all value of features are separately independent from each other.

**2.2 Decision Tree**

Decision tree learners are powerful classifiers, which utilize a tree structure to model the relationships among the features and the potential outcomes. A decision tree classifier uses a structure of branching decisions, which channel examples into a final predicted class value.

Decision trees are built using a heuristic called recursive partitioning. This approach is also commonly known as divide and conquer because it splits the data into subsets, which are then split repeatedly into even smaller subsets, and so on and so forth, until the process stops when the algorithm determines the data within the subsets are sufficiently homogenous, or another stopping criterion has been met.

There are numerous implementations of decision trees, but one of the most well-known implementations is the C5.0 algorithm. Therefore, in this project, we use C5.0 algorithms for our classification problem.

**2.3 Support Vector Machine**

A Support Vector Machine (SVM) can be imagined as a surface that creates a boundary between points of data plotted in multidimensional that represent examples and their feature values. To partition data into groups of similar class values, SVMs use a flat boundary called a hyperplane, which divides the space to create fairly homogeneous partitions on either side.

**3. Methodology:**

**3.1 Data description**

This product quality data had been collected during the manufacturing process. Data set includes two CSV data files: param\_data\_train and param\_data\_test. There are 12 features in training data set, and similarly, 11 features in testing data set for lack of the label feature. While training set contains 43,960 observations, testing set has 43,432 observations. Since data had been collected from the company, all the information has been confidentially encoded for security purpose.

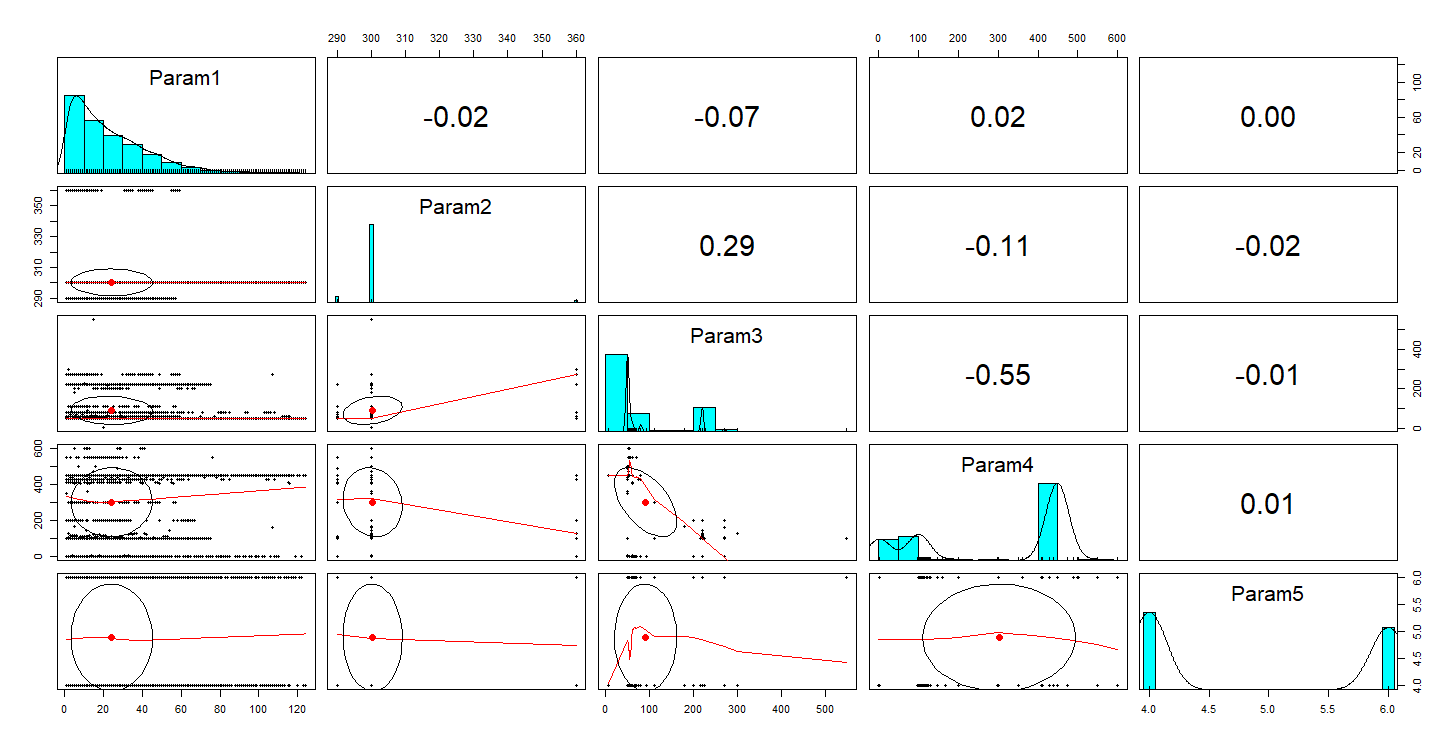
Features include product\_no (number of batch product), label (good (1) or bad (0)), Material A, Material B, Brand Name, Material Size, Mix Proportion and some other numerical parameters. The purpose of this project is to create a model classifier and implement it on the test datasets to predict the product quality (predict quality label).

**3.2 Exploring and Preprocessing data**

**3.2.1 Data Visualization**

**3.2.1.1 Numerical Features**

***Figure 1: Scatterplot matrix for all numerical features***

****

The diagonal of the scatterplots illustrates histograms of five numerical variables with informative distribution of values for each feature, whereas plots below diagonal visually represent correlation among those features. Correlation ellipses, which indicate the strength of correlation between two features, obtain the point at the mean values as their centers. We can easily recognize how strong two features relate to each other through the shape of the ellipse. The more it is stretched, the stronger the correlation. As we can see, the data had a very strong correlation between Param 3 and Param 4, while there is almost no relationship between Param 1 and Param 5.

**3.2.1.2 Principal Component Analysis (PCA)**

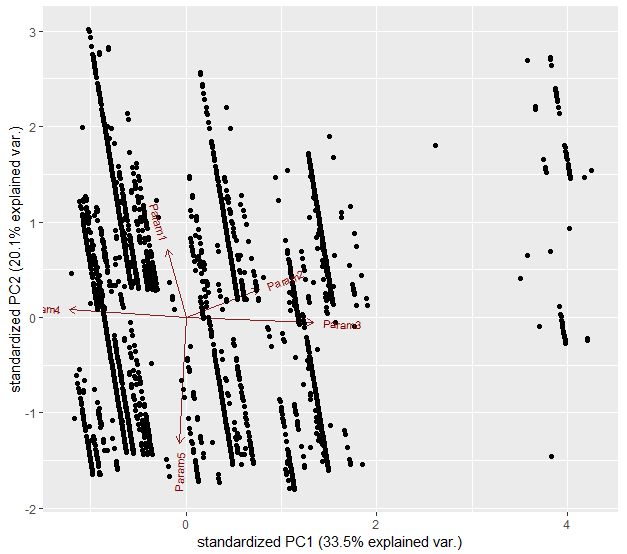
Principal Component Analysis (PCA) is a useful technique that allows us to visualize the variation present in the dataset with many variables. PCA simplifies a dataset by turning original variables into “Principal Components” which represents the underlying structure in the data. Each principal component illustrates the direction where the data is most spread out. PCA is a linear transformation procedure that fits the dataset to a new coordinated system where correlated variables contribute to the same principal component. Therefore, each principal component sums up a certain percentage of the total variation in the dataset.

Because PCA works best with numerical variables, we exclude all categorical variables in the dataset. The remaining numerical variables are Param1, Param2, Param3, Param4, and Param5. We performed PCA and obtained the following results:

As shown in the summary table, we obtained 5 principal components (PC1-5). At first glance, performing PCA on numerical variables of this dataset does not reduce the number of dimensions. This outcome is due the nature of our dataset, in which one variable is not highly correlated to one another. Each principal component explains a percentage of the total variation of the dataset. PC1 explains 33.49% of the total variance. Similarly, PC2, PC3, PC4, and PC5 explain 20.12%, 19.85%, 18.13%, and 8.41%, respectively.

To give a better visualization, we plotted PCA on a biplot that includes the position of each sample in terms of PC1 and PC2, which are the two principal components that explain the most variation. It also illustrates how much each original variable contribute to each principal component.

***Figure 2: PCA plot***



**3.2.2 Data Preprocessing**

Due to confidential issue existing in input data, our team decided to decode whole data sets. The aim is to make it more interpretable and accessible during training and processing data. The data after that, will be imputed to tackle missing value issues, as well as normalized to transform data into normal distribution before feeding it into the training stage.

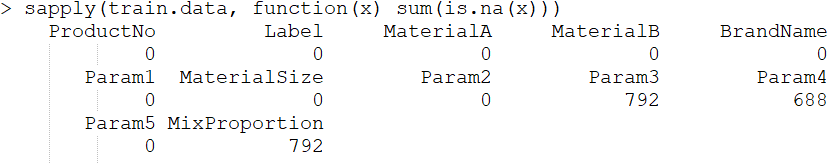
**3.2.2.1 Data decoding:**

We used unique() function to exam the number of different observations in each variable. We then renamed all of those value into meaningful names corresponding to their labels. For instance, variable Material A contains 4 different types of materials needed for production process, thus we renamed them as A1, A2, A3, and A4. Similarly to all other variables, we also converted some numerical variables, which presented as a string data type, into numeric data type.

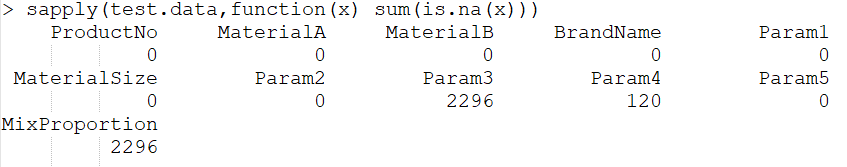
**3.2.2.2 Missing value imputation**

After doing several steps to determine the absence of values in data set, we figure out that missing value is presented in this input data and need to be tackle before using.

***Figure 2: Missing value table of training data***

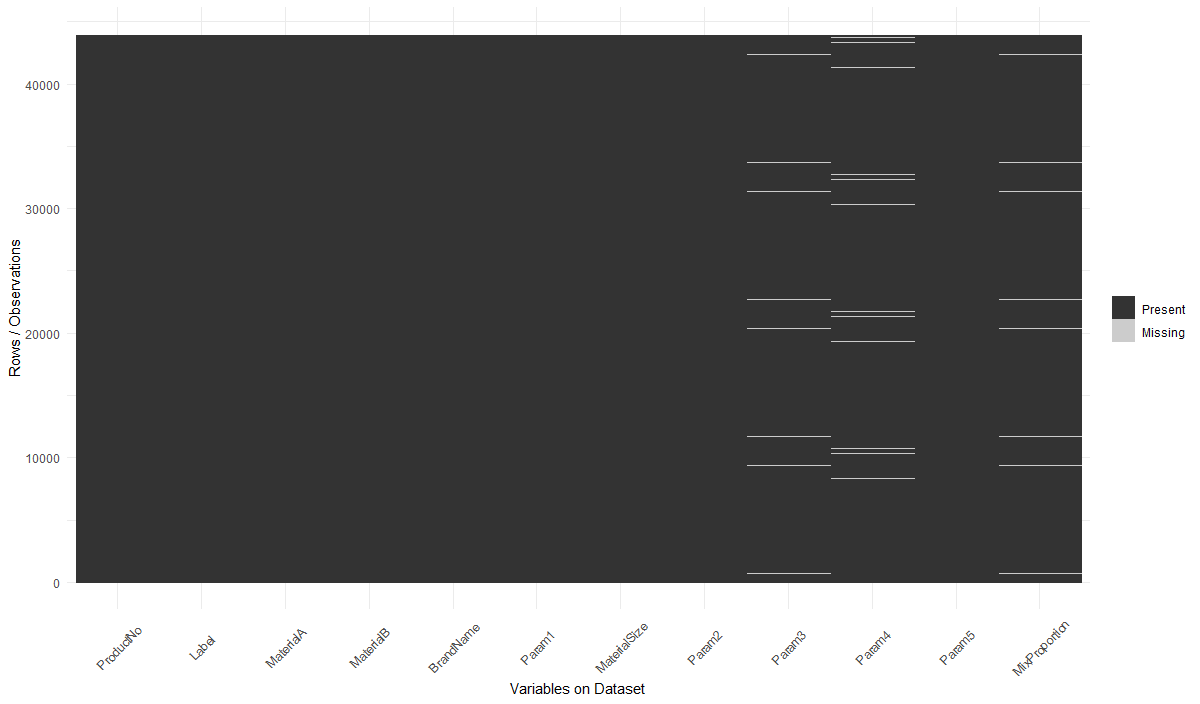


***Figure 3: Missing value table of testing data***

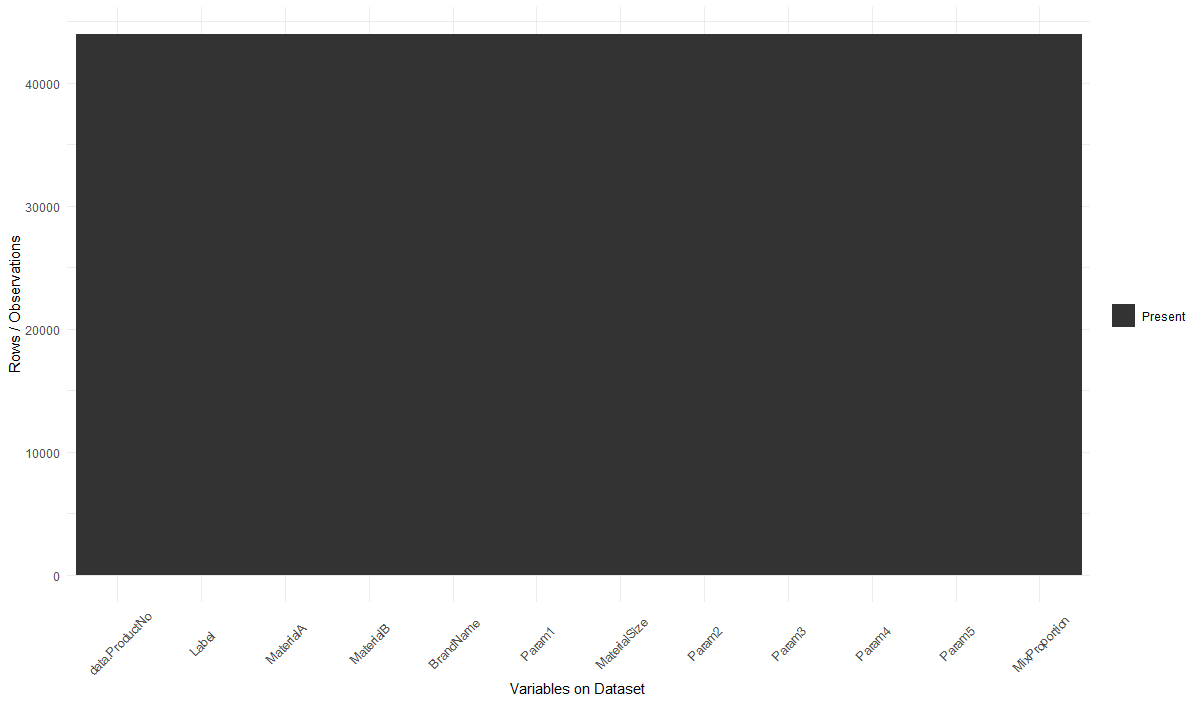


As we can see, training and testing data contain several missing values in variables: Param3, Param4, and MixProportion. The first step of imputing missing values is to make sure that all missing values will be recognized as NAs values. Thus, some lines had been added to convert all those values in to NA data type.

After that, we used missForest approach to overcome this issue together with ggplot to map the missing values. For instance, two figures below show the comparison of data before and after implementing the missing value imputation in the training data.

***Figure 4: Missing values map before imputation in training set***

***Figure 5: Missing values map after imputation in training set***

******

**3.2.2.3 Handling mismatched features existence**

***Figure 6: Mismatched features between training and testing data***

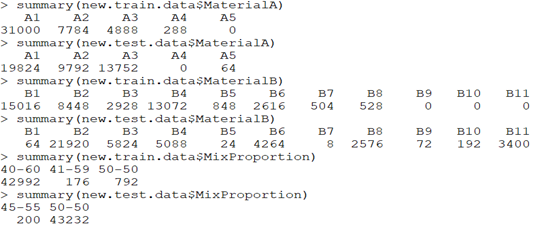
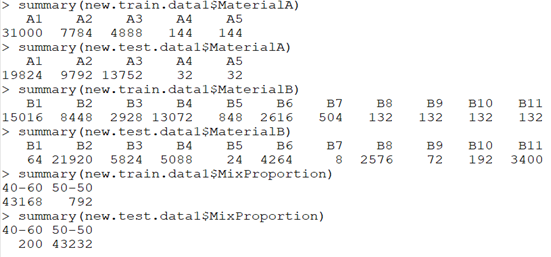
****

Figure 4 above proved that there are differences in number of features between two sets. Material A in training set does not include A5 type, while A4 is absent in testing set. Material B in training set does not record 3 last material types B9, B10, and B11. Moreover, Mix Proportion of training set had 3 various ratio which are not corresponding to two of the testing data.

Since the inconsistency only happened in small number of records, our team decided to handle vacant features by spitting out the existing values. Particularly, in Material A of training data, we filled Material A5 with a half of A4, and do similarly for testing data. In Material B, our team decided to equally split B8 into 4 parts and convert three of them into B9, B10, and B11. Finally, for Mix Proportion we kept the number of ratio in 2 for both set with 40-60 and 50-50. As 41-59 is just slightly different from 40-60 in the training set, thus, we combined them together. The ratio 45-55 will be relabeled as 40-60 in the testing set. Summary of those features after handling inconsistency will show as figure 5 below.

***Figure 7: Mismatched features after handling***

****

**3.2.2.4 Factoring categorical features and create dummy variables**

Since some machine learning algorithms require inputs as numbers, all categorical data need to be factored before feeding into the models. We factored using dummy method for all categorical features.

**3.2.2.5 Splitting data**

In order to examine the performance and select the model with a best fit for our data, we decided to divide the training data into two subsets, which will be sub training and sub testing data, and establish a classifier on each algorithm for comparison. Which model yield the best accuracy will be used on input testing data. Data had been split with the portion of 75% for training and 25% for testing.

**3.2.2.4 Data normalization**

Both input training and testing data also need to be normalized before actually feeding into model classifier. We used min-max method for our data set.

**3.3 Implementation and Model Evaluation**

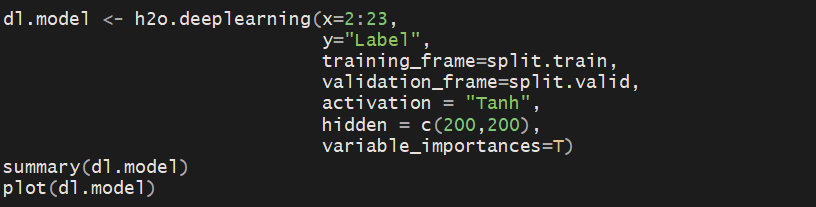
We used 4 common approaches (Naive Bayes, Decision Tree C5.0, Support Vector Machine SVM, and Deep Learning Neural Network H2O) for prediction the good-quality product. To evaluate the algorithm performance, we showed the prediction that the algorithm produced and Calculate the Accuracy of the predictions. The accuracy is calculated by the formula:

Accuracy = (TP + TN) / ( TP + TN + FP + FN)

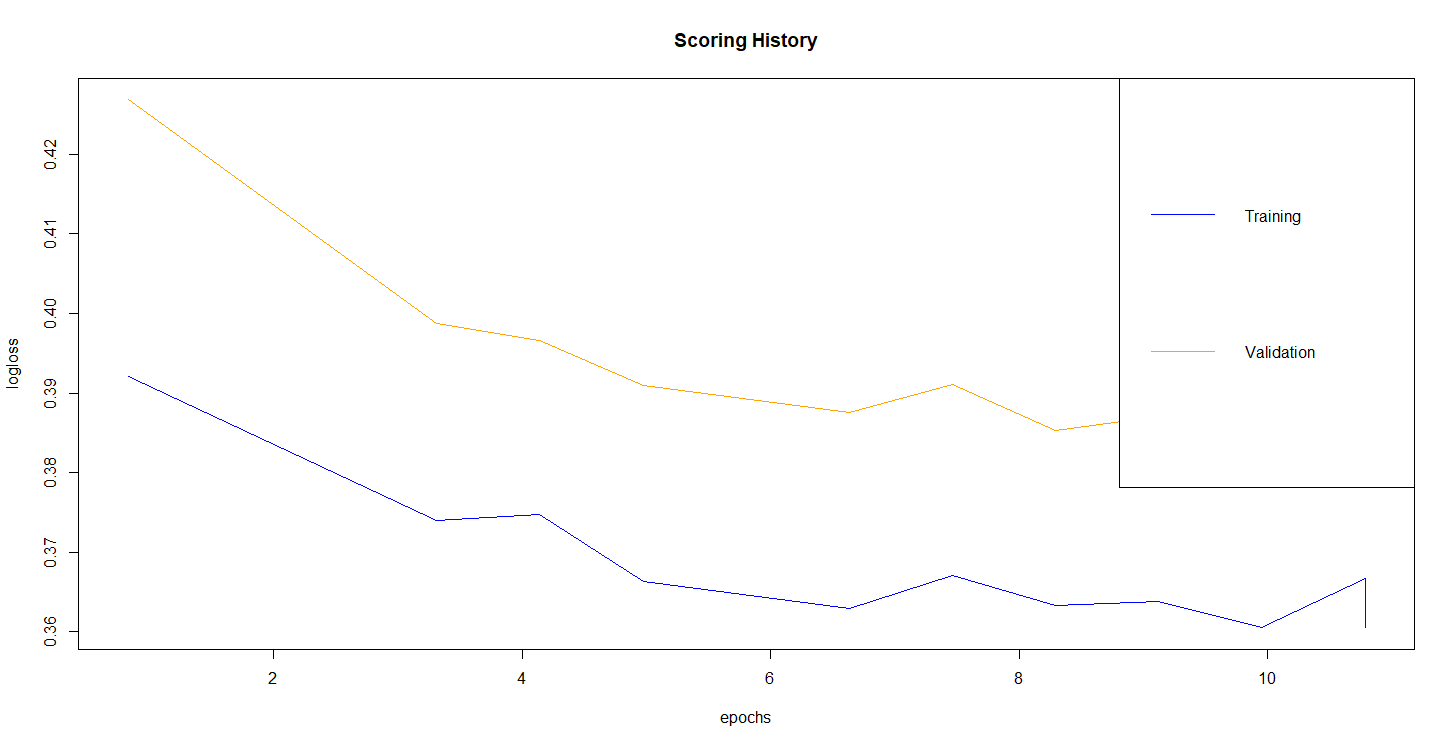
A true positive (TP) is an outcome where the model correctly predicts the positive class and a true negative (TN) is an outcome where the model correctly predicts the negative class. Whereas, a false positive (FP) is an outcome where the model incorrectly predicts the positive class and a false negative (FN) is an outcome where the model incorrectly predicts the negative class.

**3.3.1 Neural Network (H2O)**

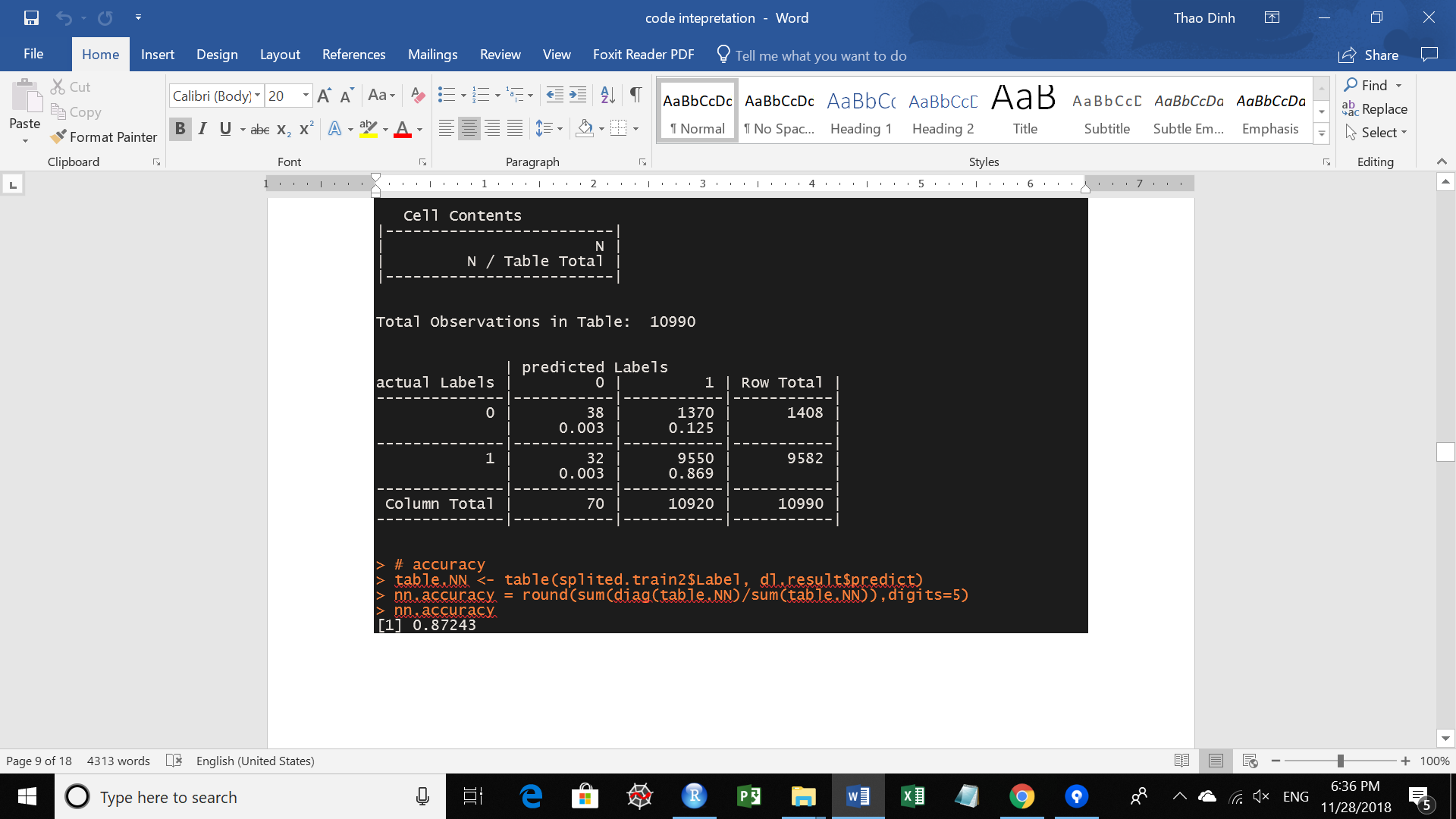
After splitting train.hex into training part (split.train) and validation part (split.valid), we built neural network model (H2O Binomial) with 2-class classification, Bernoulli distribution and 200 layers. Model also list out most of the important variables, and which model recognize as the most significant variables.



***Figure 8: Scoring history of training data with validation in 10 epochs***



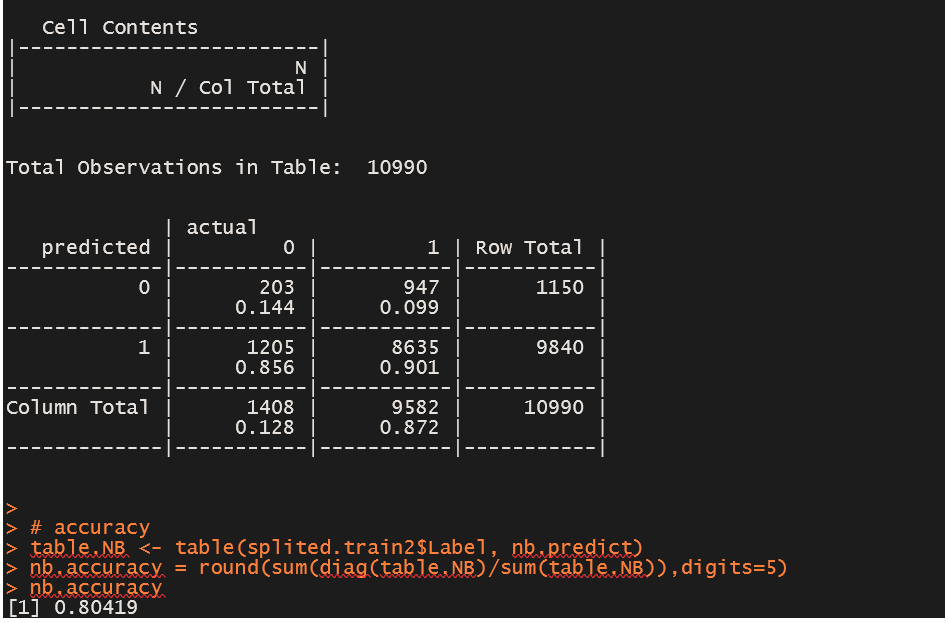
This is the plot for demonstrating the model we built. Model had run under 10 epochs. The logloss for both training and validation set had been reduced over the time. This means that during the training process, the model has been improved for every time it went through layers.



The accuracy of the model is approximately 87.2%

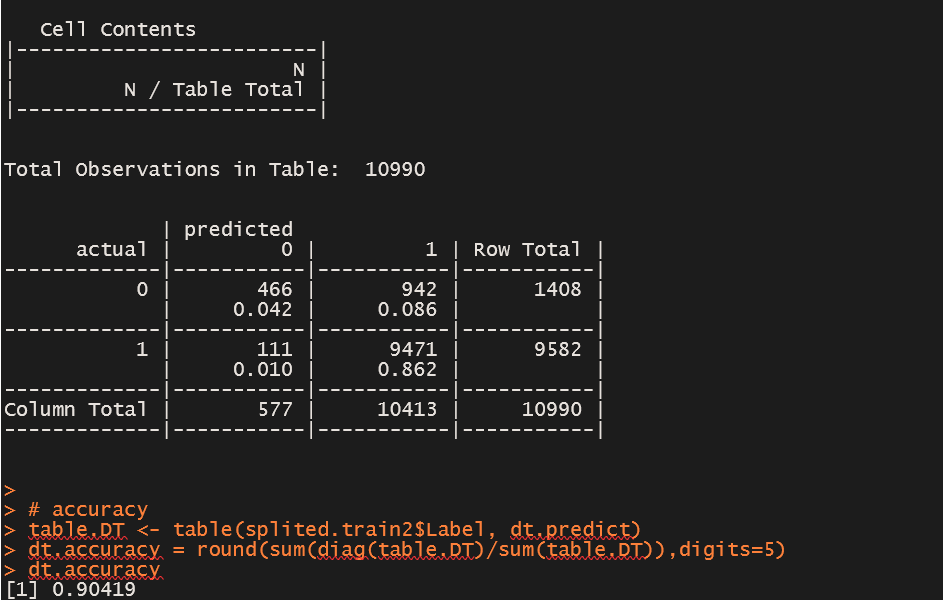
**3.3.2 Naive Bayes**

In term of Naive Bayes model, we gain the outcome of 80.42% of accuracy.

****

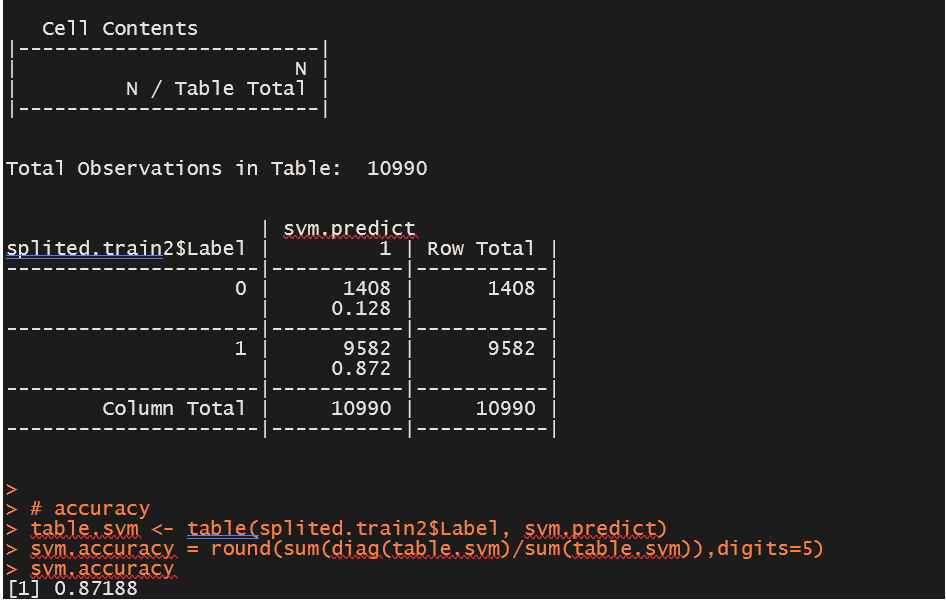
**3.3.2 Decision Trees**

Using “c50” package, we built the Decision Trees model and predicted the class with estimated accuracy of 90.4%

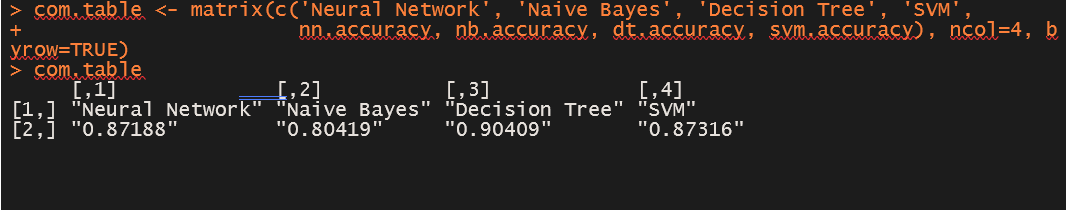
****

**3.3.4 Support Vector Machine (SVM)**

We utilized library (e1071) to establish the SVM model. The predicted model is calculated with 87.2% accuracy.



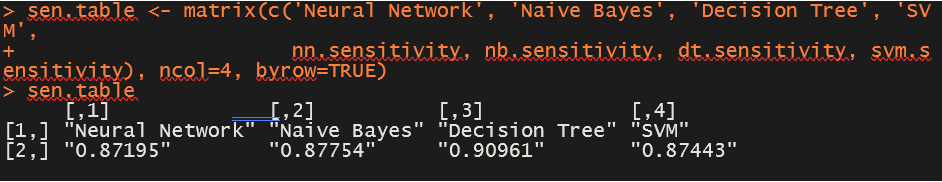
Taking everything into accounts, we would compare each of algorithms’ results with the figure below.



The Decision Tree model performs the most precise outcome, following by SVM, Neural Network and Naive Bayes when we applied the model for train dataset.

The difference of accuracy among all four models is insignificant. So, we decided to continue evaluating Sensitivity and Specificity of these models to know which model the most appropriate approach for testing data would be. In our case, since the number of good products is higher than the number of bad products, the accuracy tends to be correlated with Sensitivity in regardless of Specificity. Thus, we will calculate Sensitivity for our on-going evaluation.

In statistics, there are two types of errors. In this case, Type I error is when False Positive (FP) happens and Type II error is when False Negative (FN) happens. From the point of view of a manufacturing firm, Type II error will be costlier because a firm does not want to produce bad products and mistaken them as good ones, then sell those products to customers. Having said that, company may lose its reputation by providing customers with poor products. Therefore, we will choose the model with the smallest FN or in other words, the model with the highest Sensitivity where Sensitivity = TP / (TP + FN)



As can be seen from Sensitivity Tables, Decision Tree yielded the highest sensitivity with 90.96%. All in all, this algorithm could be considered as the most appropriate approach to predict the quality of products on testing data since it has the highest ratios in terms of Accuracy and Sensitivity.

**4. Prediction and Result Analysis:**

In similar way, we applied the 4 common algorithms to *param\_data\_test* dataset and got the predicting results below.

***Figure 9: Table of good quality prediction for each model classifier***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Neural Network** | **Naive Bayes** | **Decision Tree** | **SVM** |
| Percentage of good quality prediction **(%)** | 96.77657 | 8.123043 | 84.34334 | 100 |

Among four of the approaches, SVM gave the highest percentage good quality prediction with 100%, followed by Neural Network and Decision Tree with 96.78% and 84.34% respectively. Surprisingly, there is a big gap between Naive Bayes model outcome and the other models’ ones.

Based on the accuracy and sensitivity evaluation on the train data, we are led to the conclusion that Decision Trees have the highest accuracy. In other words, Decision Tree is the most dependable algorithm for our dataset. In terms of prediction for good quality product on the test data, we got 84.34% of test data, which are classified as good product.

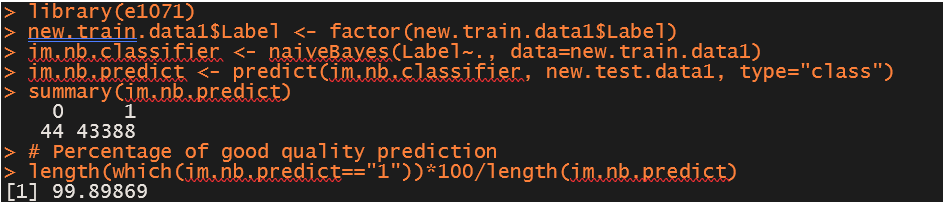
To have a better visualization of our research, we created a graph for the overall results.

**5. Improvement and conclusion:**

**5.1 Improvement:**

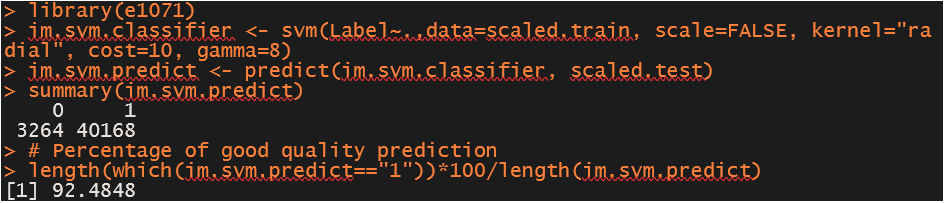
As we looked at our result from Result Analysis, we figured out some abnormal outcome which will need to be improved to give a better analysis.

First, it seems to be incorrect that Naïve Bayes method produced a percent of good quality prediction at an extremely low rate compared to other methods. This raises a question “are input testing and training data sets really suitable for this model?”. After deeply inspecting the input data, we found out that, our team had performed Naïve Bayes model on normalized and one-hot encoded data which will lead to false interpretation and thus wrongly produces prediction. To improve this, we decided to use input data sets which are not normalized or created dummy variables.



No surprises, Naïve Bayes showed that 99.899% of products are classified as good quality of production. Much better than 8.12% of its result before.

Second, SVM method, somehow, predicted the class of quality product with 100% good. As we all know there is no way that a method can be completely predict all data set as a single class. This make us think about our model parameters again and figured out that we are lacking cost and gramma use in our parameters. The result had been underfitting since the model use the default cost parameter is 1 which is too small. In SVM cost parameter controls the tradeoff between classification of training points accurately and a smooth decision boundary. Using too small cost will lead to no penalty for misclassification when fitting input data into the model, especially, it is true for data sets which are not well-separated. This is what our model had been performed. To tackle this issue, we select the cost parameter cost=10 and using gramma=8 to smoothen our model. The result return more reliable result as shown below:



In fact, SVM only produced 92.48% of product classified as good quality of production.

***Figure 10: Table of good quality prediction for each model classifier after improvement***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Neural Network** | **Naive Bayes** | **Decision Tree** | **SVM** |
| Percentage of good quality prediction **(%)** | 96.77657 | 99.89869 | 84.34334 | 92.4848 |

**5.2 Conclusion:**

In today’s business environment, product quality is one of the most crucial factors that will define the success of a company. More and more manufacturing companies are committed to zero-defect policy, in order to deliver the best products and experiences to their customers. This quality challenge was what drove us to conducting this research. The proposed approach was to build predictive models that learn from previous production outcomes to identify the most relevant features to the quality of the finished products. These models are designed to detect infrequent and unexpected quality events early before they occur in production.

The defect detection was formulated as a binary classification problem. As illustrated in the previous sections, several supervised ML algorithms can be applied. The data were preprocessed before feeding to those models. The models were trained and tested by experimental datasets. By evaluating the performance of each model, we can select the appropriate method to solve the given manufacturing research problem. In our study, Naïve Bayes, followed by Neural Network, and SVM, are three predictive models that gave us the highest prediction results in terms of the proportion of good quality. Among them, Decision Tree is the most reliable model since it gives us highest sensitivity and accuracy in the training phase.

By using different classifiers to predict the quality of manufactured products in production, manufacturing companies can optimize their manufacturing processes to maximize efficiency and productivity. They can make changes to the materials used, brand selections, environmental factors, machine tool parameters, etc. to improve process and product quality sustainably. The proposed approach can be adapted and widely applied to manufacturing processes to boost the performance of traditional quality methods and potentially move quality standards forward, where soon virtually no defective product will reach the market.

**Appendix**

**A1. Main code**

source**(**"utils.R"**)**

# Load datasets

install.packages**(**"data.table"**)**

library**(**data.table**)**

train.data **<-** fread**(**"param\_data\_train.csv", header **=** **TRUE)**

head**(**train.data**)**

test.data **<-** fread**(**"param\_data\_test.csv", header **=** **TRUE)**

head**(**test.data**)**

### Data Modification ###

## processing raw data

# Remove blank columns

train.data **<-** train.data**[**,**(**13**:**15**):=** **NULL]**

# Change columns' name

train.data **=** renameTrainDataCols**(**train.data**)**

test.data **=** renameTestDataCols**(**test.data**)**

# Check the unique items in dataset

sapply**(**train.data, **function(**x**)** unique**(**x**))**

sapply**(**test.data, **function(**x**)** unique**(**x**))**

# convert all missing values into NAs in train and test data

train.data**$**MixProportion **=** ifelse**(**train.data**$**MixProportion**==**"", **NA**, train.data**$**MixProportion**)**

test.data**$**MixProportion **=** ifelse**(**test.data**$**MixProportion**==**"", **NA**, test.data**$**MixProportion**)**

# Convert all categorical columns into factors and create dummy variables

# for training data set

train.data**$**MaterialA **=** labelMaterialA**(**train.data**$**MaterialA**)**

train.data**$**MaterialB **=** labelMaterialB**(**train.data**$**MaterialB**)**

train.data**$**BrandName **=** labelBrandName**(**train.data**$**BrandName**)**

train.data**$**MaterialSize **=** labelMaterialSize**(**train.data**$**MaterialSize**)**

train.data**$**ProductNo **<-** as.factor**(**train.data**$**ProductNo**)**

train.data**$**MixProportion **<-** as.factor**(**train.data**$**MixProportion**)**

# for testing data set

test.data**$**MaterialA **=** labelMaterialA**(**test.data**$**MaterialA**)**

test.data**$**MaterialB **=** labelMaterialB**(**test.data**$**MaterialB**)**

test.data**$**BrandName **=** labelBrandName**(**test.data**$**BrandName**)**

test.data**$**MaterialSize **=** labelMaterialSize**(**test.data**$**MaterialSize**)**

test.data**$**ProductNo **<-** as.factor**(**test.data**$**ProductNo**)**

test.data**$**MixProportion **<-** as.factor**(**test.data**$**MixProportion**)**

## handle missing values #########

# Use ggplot\_missing funtion to map missing values

install.packages**(**"reshape2"**)**

install.packages**(**"dplyr"**)**

install.packages**(**"ggplot2"**)**

install.packages**(**"missForest"**)**

library**(**reshape2**)**

library**(**dplyr**)**

library**(**ggplot2**)**

library**(**missForest**)**

# Map mising values using the function

ggplotMissingData**(**train.data**)**

sapply**(**train.data, **function(**x**)** sum**(**is.na**(**x**)))**

ggplotMissingData**(**test.data**)**

sapply**(**test.data,**function(**x**)** sum**(**is.na**(**x**)))**

# Imputation for missing values

new.train.data **<-** imputeMissingValues**(**train.data**)**

sapply**(**new.train.data, **function(**x**)** sum**(**is.na**(**x**)))** # recheck missing values

new.test.data **<-** imputeMissingValues**(**test.data**)**

sapply**(**new.test.data, **function(**x**)** sum**(**is.na**(**x**)))** # recheck missing values

# Visualization for numberic features

valName **<-** names**(**new.train.data**)**

drawHistogram**(**new.train.data**$**Param1, valName**[**6**])**

drawHistogram**(**new.train.data**$**Param2, valName**[**8**])**

drawHistogram**(**new.train.data**$**Param3, valName**[**9**])**

drawHistogram**(**new.train.data**$**Param4, valName**[**10**])**

drawHistogram**(**new.train.data**$**Param5, valName**[**11**])**

install.packages**(**"psych"**)**

library**(**psych**)**

cor**(**new.train.data**[**c**(**"Param1", "Param2", "Param3", "Param4", "Param5"**)])**

pairs.panels**(**new.train.data**[**c**(**"Param1", "Param2", "Param3", "Param4", "Param5"**)])**

# map mising values using the function

ggplotMissingData**(**new.train.data**)**

ggplotMissingData**(**new.test.data**)**

### Data Preparation ###

## Process unmatched features between train and test sets

new.train.data1 **<-** featureMatch**(**new.train.data**)**

new.test.data1 **<-** featureMatch**(**new.test.data**)**

## create dummies variables for categorical attributes

processed.train **=** processNominalVars**(**new.train.data1**)**

scaled.train **=** normalizeData**(**processed.train**)**

summary**(**scaled.train**)**

processed.test **=** processNominalVars**(**new.test.data1**)**

scaled.test **=** normalizeData**(**processed.test**)**

summary**(**scaled.test**)**

## Partition for training data only

# Split the training data into training set and testing set

install.packages**(**"caret"**)**

library**(**caret**)**

set.seed**(**200**)**

# Lets do stratified sampling. Select rows to based on Class variable as strata

TrainingDataIndex **<-** createDataPartition**(**scaled.train**$**Label, p**=**0.75, list**=FALSE)**

# Create Training Data as subset

splited.train1 **<-** scaled.train**[**TrainingDataIndex,**]**

# Everything else not in training is test data.

splited.train2 **<-** scaled.train**[-**TrainingDataIndex,**]**

## Reduce dimension by performing PCA

nomvars **<-** c**(**1, 7**:**26**)**

colnames**(**splited.train1**[**,**-**nomvars**])**

splited.train1.pca **<-** prcomp**(**splited.train1**[**,**-**nomvars**]**, center **=** **TRUE**, scale. **=** **TRUE)**

install.packages**(**"devtools"**)**

library**(**devtools**)**

install\_github**(**"vqv/ggbiplot"**)**

library**(**ggbiplot**)**

ggbiplot**(**splited.train1.pca**)**

### Modeling data ###

## Neural Network with H2O method

install.packages**(**"h2o"**)**

library**(**h2o**)**

h2o.init**(**nthreads**=**8, max\_mem\_size**=**"2G"**)**

h2o.removeAll**()** ## clean slate - just in case the cluster was already running

train.hex **<-** as.h2o**(**splited.train1**)**

test.hex **<-** as.h2o**(**splited.train2**)**

splits **<-** h2o.splitFrame**(**train.hex, 0.8, seed**=**777**)**

split.train **<-** h2o.assign**(**splits**[[**1**]]**, "train.hex"**)** # 80%

split.valid **<-** h2o.assign**(**splits**[[**2**]]**, "valid.hex"**)** # 20%

dl.model **<-** h2o.deeplearning**(**x**=**2**:**23,

y**=**"Label",

training\_frame**=**split.train,

validation\_frame**=**split.valid,

activation **=** "Tanh",

hidden **=** c**(**200,200**)**,

variable\_importances**=**T**)**

summary**(**dl.model**)**

plot**(**dl.model**)**

dl.model.predict **<-** h2o.predict**(**dl.model, test.hex**)**

dl.result **<-** as.data.frame**(**dl.model.predict**)**

dl.result

# Measure performance of H20 DL model

perf **<-** h2o.performance**(**dl.model, test.hex**)**

h2o.confusionMatrix**(**perf**)**

h2o.shutdown**()**

# examine the dl.result

summary**(**dl.result**)**

#### Explanation

# p0 is the probability that 0 is chosen.

# p1 is the probability that 1 is chosen.

# predict: is made by applying a threshold to p1

# List the important variables

head**(**as.data.frame**(**h2o.varimp**(**dl.model**)))**

# Confusion Matrix

# install.packages("gmodels")

library**(**gmodels**)**

CrossTable**(**splited.train2**$**Label, dl.result**$**predict,

prop.chisq **=** **FALSE**, prop.c **=** **FALSE**, prop.r **=** **FALSE**,

dnn **=** c**(**'actual Labels', 'predicted Labels'**))**

# accuracy

table.NN **<-** table**(**splited.train2**$**Label, dl.result**$**predict**)**

nn.accuracy **=** round**(**sum**(**diag**(**table.NN**)/**sum**(**table.NN**))**,digits**=**5**)**

nn.accuracy

# Sensitivity

nn.sensitivity **=** round**(**table.NN**[**4**]/**sum**(**table.NN**[**3**:**4**])**,digits**=**5**)**

nn.sensitivity

## Naive Bayes

splited.train1**$**Label **<-** factor**(**splited.train1**$**Label**)**

splited.train2**$**Label **<-** factor**(**splited.train2**$**Label**)**

install.packages**(**"e1071"**)**

library**(**e1071**)**

nb.classifier**<-** naiveBayes**(**Label**~**., data**=**splited.train1**)**

nb.predict **<-** predict**(**nb.classifier, splited.train2**)**

head**(**nb.predict**)**

# Confusion Matrix

library**(**gmodels**)**

CrossTable**(** splited.train2**$**Label, nb.predict,

prop.chisq **=** **FALSE**, prop.t **=** **FALSE**, prop.r **=** **FALSE**,

dnn **=** c**(**'actual', 'predict'**))**

# accuracy

table.NB **<-** table**(**splited.train2**$**Label, nb.predict**)**

nb.accuracy **=** round**(**sum**(**diag**(**table.NB**)/**sum**(**table.NB**))**,digits**=**5**)**

nb.accuracy

# Sensitivity

nb.sensitivity **=** round**(**table.NB**[**4**]/**sum**(**table.NB**[**3**:**4**])**,digits**=**5**)**

nb.sensitivity

## Decision Tree using C50

install.packages**(**"C50"**)**

library**(**C50**)**

dt.classifier **<-** C5.0**(**splited.train1**[-**1**]**, splited.train1**$**Label**)**

# generate predictions for the testing dataset

dt.predict **<-** predict**(**dt.classifier, splited.train2**)**

# cross tabulation of predicted versus actual classes

library**(**gmodels**)**

CrossTable**(**splited.train2**$**Label, dt.predict,

prop.chisq **=** **FALSE**, prop.c **=** **FALSE**, prop.r **=** **FALSE**,

dnn **=** c**(**'actual', 'predicted'**))**

# accuracy

table.DT **<-** table**(**splited.train2**$**Label, dt.predict**)**

dt.accuracy **=** round**(**sum**(**diag**(**table.DT**)/**sum**(**table.DT**))**,digits**=**5**)**

dt.accuracy

# Sensitivity

dt.sensitivity **=** round**(**table.DT**[**4**]/**sum**(**table.DT**[**3**:**4**])**,digits**=**5**)**

dt.sensitivity

## SVM Support Vector Machine

library**(**e1071**)**

svm.classifier **<-** svm**(**Label**~**.,data**=**splited.train1, scale**=FALSE)**

svm.predict **<-** predict**(**svm.classifier, splited.train2**)**

summary**(**svm.predict**)**

# cross tabulation of predicted versus actual classes

library**(**gmodels**)**

CrossTable**(**splited.train2**$**Label, svm.predict,

prop.chisq **=** **FALSE**, prop.c **=** **FALSE**, prop.r **=** **FALSE**,

dnn **=** c**(**'actual', 'predicted'**))**

# accuracy

table.svm **<-** table**(**splited.train2**$**Label, svm.predict**)**

svm.accuracy **=** round**(**sum**(**diag**(**table.svm**)/**sum**(**table.svm**))**,digits**=**5**)**

svm.accuracy

# Sensitivity

svm.sensitivity **=** round**(**table.svm**[**4**]/**sum**(**table.svm**[**3**:**4**])**,digits**=**5**)**

svm.sensitivity

### Model Evaluation ###

# Table to compare each algorithms' results

acc.table **<-** matrix**(**c**(**'Neural Network', 'Naive Bayes', 'Decision Tree', 'SVM',

nn.accuracy, nb.accuracy, dt.accuracy, svm.accuracy**)**, ncol**=**4, byrow**=TRUE)**

acc.table

sen.table **<-** matrix**(**c**(**'Neural Network', 'Naive Bayes', 'Decision Tree', 'SVM',

nn.sensitivity, nb.sensitivity, dt.sensitivity, svm.sensitivity**)**, ncol**=**4, byrow**=TRUE)**

sen.table

### Apply on test data ###

## Apply Naive Bayes algorithm

library**(**e1071**)**

nb.classifier.data **<-** naiveBayes**(**Label**~**., data**=**scaled.train**)**

nb.predict.data **<-** predict**(**nb.classifier.data, scaled.test, type**=**"class"**)**

summary**(**nb.predict.data**)**

# Percentage of good quality prediction

length**(**which**(**nb.predict.data**==**"1"**))\***100**/**length**(**nb.predict.data**)**

## Apply SVM algorithm

library**(**e1071**)**

svm.classifier.data **<-** svm**(**Label**~**.,data**=**scaled.train, scale**=FALSE)**

svm.predict.data **<-** predict**(**svm.classifier.data, scaled.test**)**

summary**(**svm.predict.data**)**

# Percentage of good quality prediction

length**(**which**(**svm.predict.data**==**"1"**))\***100**/**length**(**svm.predict.data**)**

## Apply Decision Tree algorithm

library**(**C50**)**

dt.classifier.data **<-** C5.0**(**scaled.train**[-**1**]**, scaled.train**$**Label**)**

dt.predict.data **<-** predict**(**dt.classifier.data, scaled.test**)**

summary**(**dt.predict.data**)**

# Percentage of good quality prediction

length**(**which**(**dt.predict.data**==**"1"**))\***100**/**length**(**dt.predict.data**)**

## Apply Deep Learning Neural Network H2O #########################

library**(**h2o**)**

h2o.init**(**nthreads**=**8, max\_mem\_size**=**"2G"**)**

h2o.removeAll**()** ## clean slate - just in case the cluster was already running

h2o.init**()**

train.hex **<-** as.h2o**(**scaled.train**)**

test.hex **<-** as.h2o**(**scaled.test**)**

splits **<-** h2o.splitFrame**(**train.hex, 0.8, seed**=**777**)**

split.train **<-** h2o.assign**(**splits**[[**1**]]**, "train.hex"**)** # 80%

split.valid **<-** h2o.assign**(**splits**[[**2**]]**, "valid.hex"**)** # 20%

dl.model **<-** h2o.deeplearning**(**x**=**2**:**26,

y**=**"Label",

training\_frame**=**split.train,

validation\_frame**=**split.valid,

activation **=** "Tanh",

hidden **=** c**(**200,200**)**,

variable\_importances**=**T**)**

summary**(**dl.model**)**

plot**(**dl.model**)**

dl.model.predict.data **<-** h2o.predict**(**dl.model, test.hex**)**

dl.result.data **<-** as.data.frame**(**dl.model.predict.data**)**

h2o.shutdown**()**

# examine the dl.result

summary**(**dl.result.data**)**

# Percentage of good quality prediction

length**(**which**(**dl.result.data**==**"1"**))\***100**/**length**(**dt.predict.data**)**

## Improvement:

# Naive Bayes

library**(**e1071**)**

new.train.data1**$**Label **<-** factor**(**new.train.data1**$**Label**)**

im.nb.classifier **<-** naiveBayes**(**Label**~**., data**=**new.train.data1**)**

im.nb.predict **<-** predict**(**im.nb.classifier, new.test.data1, type**=**"class"**)**

summary**(**im.nb.predict**)**

# Percentage of good quality prediction

length**(**which**(**im.nb.predict**==**"1"**))\***100**/**length**(**im.nb.predict**)**

# svm method:

library**(**e1071**)**

im.svm.classifier **<-** svm**(**Label**~**.,data**=**scaled.train, scale**=FALSE**, kernel**=**"radial", cost**=**10, gamma**=**8**)**

im.svm.predict **<-** predict**(**im.svm.classifier, scaled.test**)**

summary**(**im.svm.predict**)**

# Percentage of good quality prediction

length**(**which**(**im.svm.predict**==**"1"**))\***100**/**length**(**im.svm.predict**)**

**A2. external functions for main code (utils.R)**

renameTrainDataCols **<-** **function(**data**)** **{**

names**(**data**)[**1**]<-**"ProductNo"

names**(**data**)[**2**]<-**"Label"

names**(**data**)[**3**]<-**"MaterialA"

names**(**data**)[**4**]<-**"MaterialB"

names**(**data**)[**5**]<-**"BrandName"

names**(**data**)[**6**]<-**"Param1"

names**(**data**)[**7**]<-**"MaterialSize"

names**(**data**)[**8**]<-**"Param2"

names**(**data**)[**9**]<-**"Param3"

names**(**data**)[**10**]<-**"Param4"

names**(**data**)[**11**]<-**"Param5"

names**(**data**)[**12**]<-**"MixProportion"

return **(**data**)**

**}**

renameTestDataCols **<-** **function(**data**)** **{**

names**(**data**)[**1**]<-**"ProductNo"

names**(**data**)[**2**]<-**"MaterialA"

names**(**data**)[**3**]<-**"MaterialB"

names**(**data**)[**4**]<-**"BrandName"

names**(**data**)[**5**]<-**"Param1"

names**(**data**)[**6**]<-**"MaterialSize"

names**(**data**)[**7**]<-**"Param2"

names**(**data**)[**8**]<-**"Param3"

names**(**data**)[**9**]<-**"Param4"

names**(**data**)[**10**]<-**"Param5"

names**(**data**)[**11**]<-**"MixProportion"

return **(**data**)**

**}**

labelMaterialA **<-** **function(**data**)** **{**

labeled\_data **=** factor**(**data, levels **=** c**(**

"75a5f96063fbc3290b07b0e81c3249d0",

"42a6854ae47630b3a32e84823d147e0b",

"f5c2479be5048388c45cb2e81edfbd3f",

"59a06f37e2f0262919e5aab9e083ccbd",

"b27d68f669d9758e698a35c20fa2bab3"

**)**,

labels **=** c**(**

"A1", "A2", "A3", "A4", "A5"

**))**

return**(**labeled\_data**)**

**}**

labelMaterialB **<-** **function(**data**)** **{**

labeled\_data **=** factor**(**data, levels **=** c**(**

"4610065df728e0bd399446ef5fd3ea74",

"6fc672934f2f6e2a64898efd18c24111",

"0e05731278221a3ac6ebaa1d795b6177",

"fe68fc884ca9961c9e8ec90e7fc5ec9d",

"e3d6228b4bb426d6d7004613eb04e124",

"5437f1e2e5cf83c784e5dba00ee42e7a",

"9e617be991132586229e9964b8bf4463",

"129d425bfebdb61df18ad3940118751b",

"dfac0f954eb95d09d014305d0c5dbe3f",

"2e54a9875f82b6686db826c4ab45cb8a",

"5a9af4e32acd65b9baed049e037d86d3"

**)**,

labels **=** c**(**"B1", "B2", "B3", "B4", "B5", "B6", "B7", "B8", "B9", "B10", "B11" **))**

return **(**labeled\_data**)**

**}**

labelBrandName **<-** **function(**data**)** **{**

labeled\_data **=** factor**(**data, levels **=** c**(**

"29f4d775d7fd37f40a72f66f39b7453f",

"d69b8ab522882db30043159ad9918216"**)**,

labels **=** c**(** "BrandName1", "BrandName2" **))**

return **(**labeled\_data**)**

**}**

labelMaterialSize **<-** **function(**data**)** **{**

labeled\_data **=** factor**(**data, levels **=** c**(**

"0.115\*600", "0.115\*720", "0.115\*580", "0.115\*620", "0.115\*300"**)**,

labels **=** c**(** "0.115\*600", "0.115\*720", "0.115\*580", "0.115\*620", "0.115\*300" **))**

return **(**labeled\_data**)**

**}**

processNominalVars **<-** **function(**data**)** **{**

processed.data **<-** data

# convert Label column into factor

**if** **(**length**(**processed.data**)==**12**)** **{**processed.data**$**Label **<-** as.factor**(**processed.data**$**Label**)}**

# 4 dummy variables for 5 type of materialsA

processed.data**$**A1 **<-** ifelse**(**processed.data**$**MaterialA**==**"A1",1,0**)**

processed.data**$**A2 **<-** ifelse**(**processed.data**$**MaterialA**==**"A2",1,0**)**

processed.data**$**A3 **<-** ifelse**(**processed.data**$**MaterialA**==**"A3",1,0**)**

processed.data**$**A4 **<-** ifelse**(**processed.data**$**MaterialA**==**"A4",1,0**)**

# 10 dummies variable for 11 type of materialsB

processed.data**$**B1 **<-** ifelse**(**processed.data**$**MaterialB**==**"B1",1,0**)**

processed.data**$**B2 **<-** ifelse**(**processed.data**$**MaterialB**==**"B2",1,0**)**

processed.data**$**B3 **<-** ifelse**(**processed.data**$**MaterialB**==**"B3",1,0**)**

processed.data**$**B4 **<-** ifelse**(**processed.data**$**MaterialB**==**"B4",1,0**)**

processed.data**$**B5 **<-** ifelse**(**processed.data**$**MaterialB**==**"B5",1,0**)**

processed.data**$**B6 **<-** ifelse**(**processed.data**$**MaterialB**==**"B6",1,0**)**

processed.data**$**B7 **<-** ifelse**(**processed.data**$**MaterialB**==**"B7",1,0**)**

processed.data**$**B8 **<-** ifelse**(**processed.data**$**MaterialB**==**"B8",1,0**)**

processed.data**$**B9 **<-** ifelse**(**processed.data**$**MaterialB**==**"B9",1,0**)**

processed.data**$**B10 **<-** ifelse**(**processed.data**$**MaterialB**==**"B10",1,0**)**

# 1 dummy variable for 2 type of BrandName

processed.data**$**BN1 **<-** ifelse**(**processed.data**$**BrandName**==**"BrandName1",1,0**)**

# 1 dummy variable for 2 type of MixProduction

processed.data**$**Mix50.50 **<-** ifelse**(**processed.data**$**MixProportion**==**"50-50",1,0**)**

# 4 dummy variables for 5 type of MaterialSize

processed.data**$**s600 **<-** ifelse**(**processed.data**$**MaterialSize**==**"0.115\*600",1,0**)**

processed.data**$**s720 **<-** ifelse**(**processed.data**$**MaterialSize**==**"0.115\*720",1,0**)**

processed.data**$**s580 **<-** ifelse**(**processed.data**$**MaterialSize**==**"0.115\*580",1,0**)**

processed.data**$**s620 **<-** ifelse**(**processed.data**$**MaterialSize**==**"0.115\*620",1,0**)**

return **(**processed.data**)**

**}**

ggplotMissingData **<-** **function(**x**){**

x %>% is.na %>% melt %>% ggplot**(**data **=** ., aes**(**x **=** Var2, y **=** Var1**))** **+**

geom\_raster**(**aes**(**fill **=** value**))** **+**

scale\_fill\_grey **(**name **=** '', labels **=** c**(**'Present', 'Missing'**))** **+**

theme\_minimal**()** **+**

theme**(**axis.text.x **=** element\_text**(**angle **=** 45, vjust **=** 0.5**))** **+**

labs**(**x **=** 'Variables on Dataset', y **=** 'Rows / Observations'**)**

**}**

imputeMissingValues **<-** **function(**data**)** **{**

# exclude column ProductNo (product code is not a concerned variable)

data.imp **<-** missForest**(**data**[**,**!**"ProductNo", with**=FALSE])**

data.imp**$**ximp

new.data **<-** data.frame**(**data**$**ProductNo, data.imp**$**ximp**)**

return **(**new.data**)**

**}**

featureMatch **<-** **function(**data**)**

**{** # start featureMatch

# Match MaterialA Feature:

**{**

**if** **(**length**(**data**)==**12**)** **{**

index.A5 **<-** which**(**data**$**MaterialA**==**"A4"**)**

data**$**MaterialA**[**index.A5**[**1**:**144**]]** **<-** "A5"**}**

**else** **{**

index.A4 **<-** which**(**data**$**MaterialA**==**"A5"**)**

data**$**MaterialA**[**index.A4**[**1**:**32**]]** **<-** "A4" **}**

**}**

# Match MaterialB Feature:

**{**

**if** **(**length**(**data**)==**12**)** **{**

index.B **<-** which**(**data**$**MaterialB**==**"B8"**)**

data**$**MaterialB**[**index.B**[**133**:**264**]]** **<-** "B9"

data**$**MaterialB**[**index.B**[**265**:**396**]]** **<-** "B10"

data**$**MaterialB**[**index.B**[**397**:**528**]]** **<-** "B11" **}**

**}**

# Match MixProduction Feature:

**{**

**if** **(**length**(**data**)==**12**)** **{**

data**$**MixProportion **<-** as.character**(**data**$**MixProportion**)**

data**$**MixProportion**[**which**(**data**$**MixProportion**==**"41-59"**)]** **<-** "40-60"

data**$**MixProportion **<-** as.factor**(**data**$**MixProportion**)**

**}**

**else** **{**

data**$**MixProportion **<-** as.character**(**data**$**MixProportion**)**

data**$**MixProportion**[**which**(**data**$**MixProportion**==**"45-55"**)]** **<-** "40-60"

data**$**MixProportion **<-** as.factor**(**data**$**MixProportion**)** **}**

**}**

# Match MaterialSize Feature:

**{**

**if** **(**length**(**data**)==**11**)** **{**

index.s620 **<-** which**(**data**$**MaterialSize**==**"0.115\*300"**)**

data**$**MaterialSize**[**index.s620**[**1**:**236**]]** **<-** "0.115\*620"

**}**

**}**

return**(**data**)**

**}** # End featureMatch

normalizeData **<-** **function(**data**)** **{**

# Nomalization

normalize **<-** **function(**x**)** **{**return**((**x **-** min**(**x**))** **/** **(**max**(**x**)** **-** min**(**x**)))}**

**if** **(**length**(**data**)==**32**)** **{**

nom.train **<-** lapply**(**data**[**c**(**6,8**:**11**)]**,normalize**)**

scaled.data **<-** data.frame**(**data**[**2**]**,nom.train,data**[**,13**:**32**])**

return **(**scaled.data**)** **}**

**else** **{**

nom.test **<-** lapply**(**data**[**c**(**5,7**:**10**)]**,normalize**)**

scaled.data **<-** data.frame**(**nom.test,data**[**,12**:**31**])**

return **(**scaled.data**)** **}**

**}**

drawHistogram **<-** **function(**data, valName**)** **{**

hist**(**

data,

main**=** paste**(**"Histogram for", valName**)**,

xlab**=** valName,

border**=**"black",

col**=**"blue" **)}**

**References**

1. Lantz, Brett. *Machine Learning with R: Discover How to Build Machine Learning Algorithms, Prepare Data, and Dig Deep into Data Prediction Techniques with R*. Packt Publishing, 2015.

2. PCA Analysis in R.” *DataCamp Community*, www.datacamp.com/community/tutorials/pca-analysis-r.

3. Wikipedia contributors. "Manufacturing." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 19 Nov. 2018. Web. 28 Nov. 2018