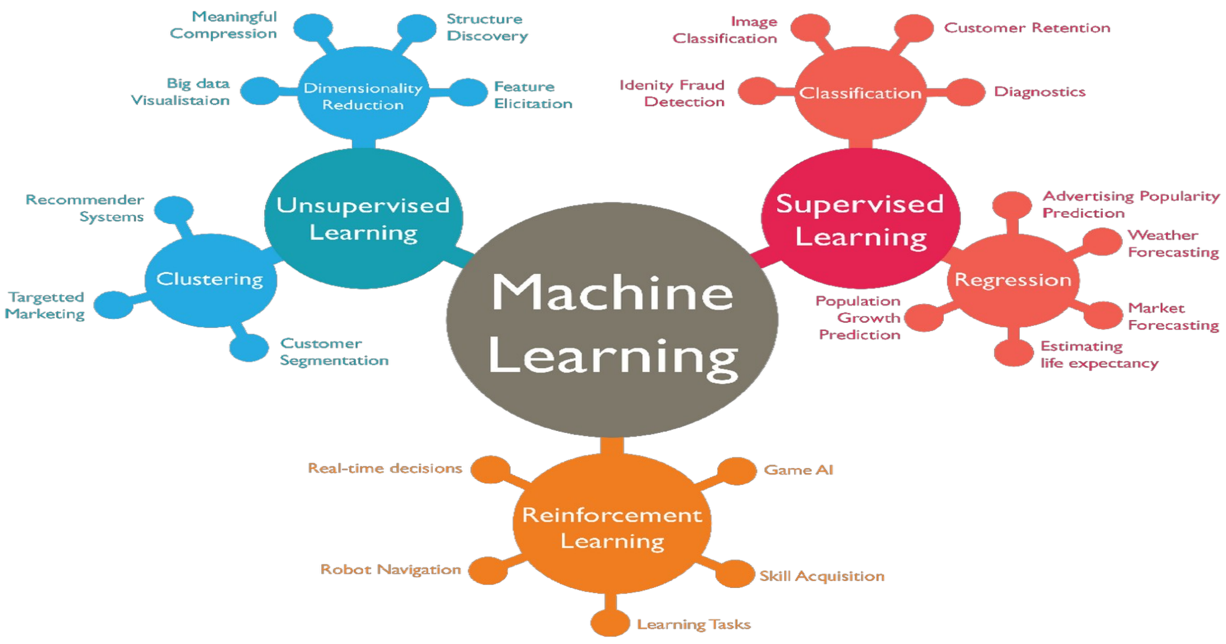
Analyzing Steel Plate Faults



<u>Prepared by:</u>

- Basem Sopeh
- Chandra Vamshi Dasyam
- Abdullah All Mamun
- Hassan Bin Javaid
- Farjad Shahbazi Avarvand

Dataset:

This dataset originates from Semeion, Research Center of Sciences and Communication, Via Sersala 117, 00128, Rome, Italy (1), and it is located on the machine learning repository since 2010 (2):

- The aim of the dataset was to classify the defective surface of stainless steel into six types (plus other).
- The dataset is a multivariate dataset, contains 1941 observations and 27 predictors.

Variables:

There are 27 variables between integers and reals, the 7 types of defects are added in the last seven columns in the dataset encoded as one-hot encoding. The variables describe the geometry shape of the defect and its outline.

Because Semeion was commissioned by the Centro Sviluppo Materiali (Italy), there are no clear details on the nature of the variables or the types of defects. The names of the variables and their data types are clarified in Table-1.

Table 1: variables and their data type

Variable name	**type**	**Variable name**	**type**
X_Minimum	Numeric (Int ,continuous)	Edges_Index	Numeric (double ,continuous)
X_Maximum	Numeric (Int ,continuous)	Empty_Index	Numeric (double ,continuous)
Y_Minimum	Numeric (Int ,continuous)	Square_Index	Numeric (double ,continuous)
Y_Maximum	Numeric (Int ,continuous)	Outside_X_Index	Numeric (double ,continuous)
Pixels_Areas	Numeric (Int ,continuous)	Edges_X_Index	Numeric (double ,continuous)
X_Perimeter	Numeric	Edges_Y_Index	Numeric

	(Int ,continuous)		(double ,continuous)
Y_Perimeter	Numeric (Int ,continuous)	Outside_Global_Index	Numeric (double, discrete)
Sum_of_Luminosity	Numeric (Int ,continuous)	LogOfAreas	Numeric (double ,continuous)
Minimum_of_Luminosity	Numeric (Int ,continuous)	Log_X_Index	Numeric (double ,continuous)
Maximum_of_Luminosity	Numeric (Int ,continuous)	Log_Y_Index	Numeric (double ,continuous)
Length_of_Conveyer	Numeric (Int ,continuous)	Orientation_Index	Numeric (double ,continuous)
TypeOfSteel_A300	logical	Luminosity_Index	Numeric (double ,continuous)
TypeOfSteel_A400	logical	SigmoidOfAreas	Numeric (double ,continuous)
Steel_Plate_Thickness	Numeric (Int ,continuous)		

Labels:

There are 7 types of labels (defects) encoded in the dataset as one-hot encoding i.e. each label can take 0 or 1 value and they are clarified in table 2.

Table 2: type of defects

Label	Type
Pastry	logical
Z_Scratch	logical
K_Scatch	logical
Stains	logical
Dirtiness	logical

Bumps	logical
Other_Faults	logical

Modeling:

the problem we are solving in this data is predicting the type of the defects depending on the input variables, clearly, this is a classification problem.

We will analyze the data and build a predictive model using the following ML methods:

- Support Vector Machine (SVM) with multiple classes (using R).
- Decision Tree Model (using R).
- Neural Networks (using Python)

we start by giving a little theoretical and mathematical overview of the principle of these three methods and then in the practical part, we explain how we apply them on our dataset and evaluate the resulted models and compare them.

1- Support Vector Machine (SVM)[3]

It is a supervised machine learning algorithm that can be employed for both classification and regression purposes. the idea is to of find a hyperplane that best divides a dataset into two classes, as shown in the image below.

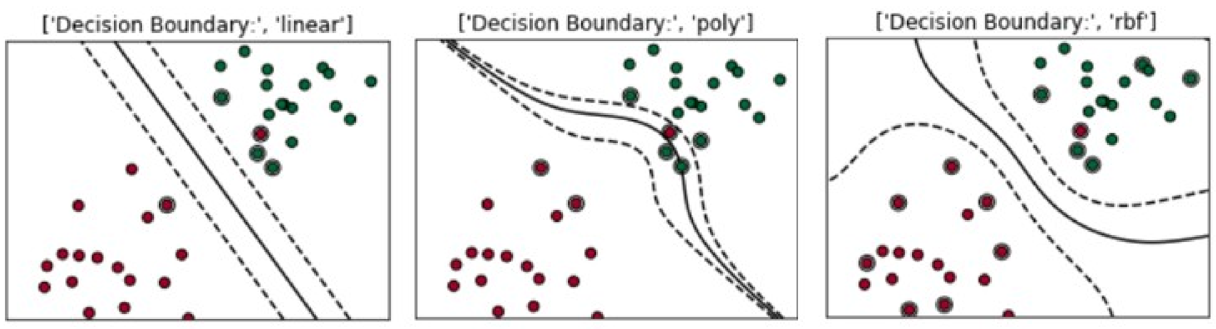


Figure 1: SVM

If we have two classes (and 2 dim), then we can define the hyperplane in a form:

$$\beta_0 + \beta_1 x + \beta_2 y = 0$$

(for 3_dim, the hyperplane would be $\beta_0 + \beta_1 x + \beta_2 y + \beta_3 z = 0$)

with this meaning, if we consider the two classes 1, -1, then

for each point of the data (x,y):

$$\beta_0 + \beta_1 x + \beta_2 y \geq 0 \quad \rightarrow \text{ the point of class 1}$$

$$\text{otherwise} \rightarrow \text{ the point of class -1}$$

these two rules could be written in the form:

$$y_i (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}) > 0$$

for each point (x_{i1} , x_{i2}) and its class y_i

the distance between each point and the hyperplane is $y_i (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})$ with the constrain $\beta_1^2 + \beta_2^2 = 1$.

therefore, to ensure that the hyperplane classify the data, the distance between each point and the hyperplane must be at least M:

$$y_i (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}) \geq M$$

then the aim is to determine the parameters β_i that maximize the margin M:

$$M = \max_{\beta_0, \beta_1, \beta_2} \{ \min_i \{ y_i (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}) \} \},$$

$$\text{subject to } \Sigma \beta_i^2 = 1.$$

data points that lie on the edge of the margin, their distance from the optimal hyperplane is M. These points are called the support vectors.

Because the hyperplane usually can not perfectly separate classes, we adapt the previous approach to allow some points to be in the margin area and the SVM algorithm can be summarized as following:

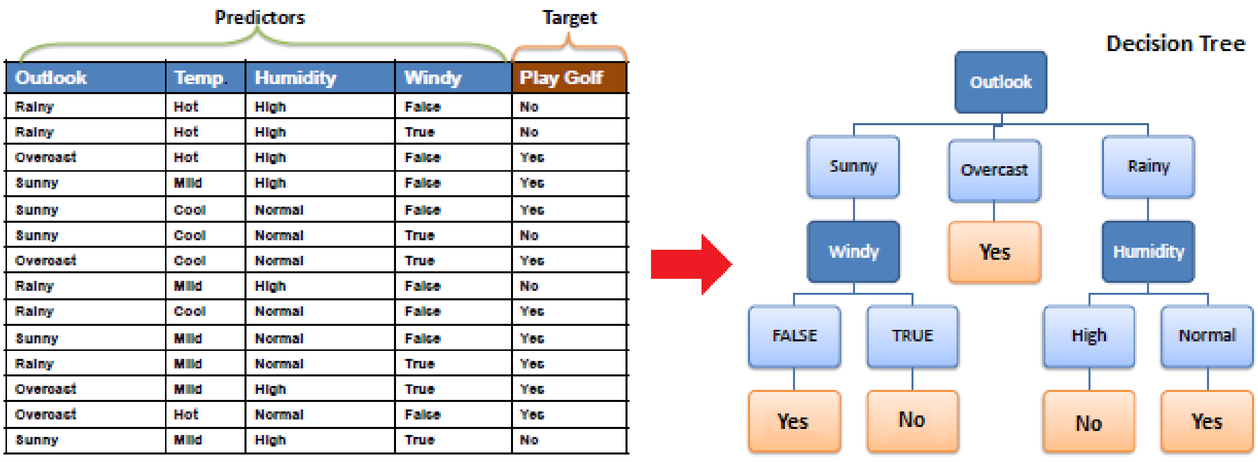
- $\max_{\beta_0, \beta_1, \beta_2, 1, \ldots, n} M$, such that
- $y_i (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}) \geq M(1 - \varepsilon_i)$ for i=1, . . . n,
- **subject to** $\Sigma \beta_i^2 = 1$, $\varepsilon_i > 0$ and $\Sigma \varepsilon_i < C$, C is a positive smoothing parameter.

For multiple classes such as our case we repeat the algorithm for all possible two-class combinations.

2- Decision Tree Model(4):

Decision tree is a type of supervised learning and can be used for both classification and regression problem.

It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.



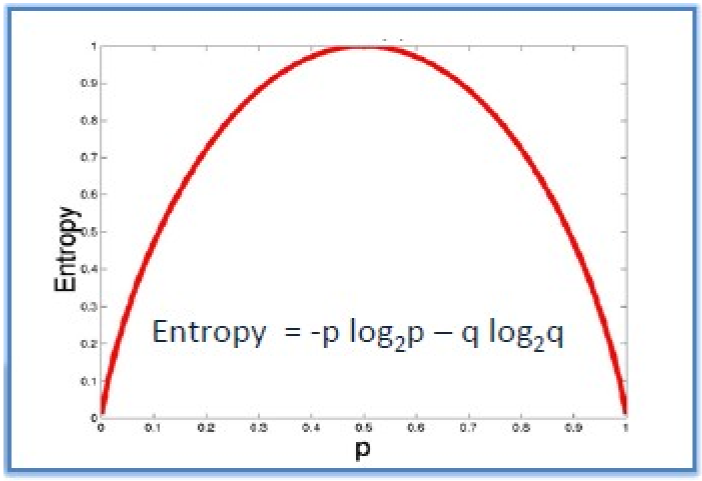
Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

Figure 2: example decision tree

Entropy:

building top-down from a root node → partitioning the data into subsets that contain instances with similar values (homogenous).

If the sample is completely homogeneous the entropy is zero and if the sample is an equally divided it has entropy of one.



$$\text{Entropy} = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$$

To build the tree, → calculate two types of entropy :

a) Entropy using the frequency table of one attribute:

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

b) Entropy using the frequency table of two attributes:

$$E(T, X) = \sum_{c \in X} P(c) E(c)$$

Assume K classification classes, and M nodes.

For node m we compute the proportion of observations in the training data which end up in node m:

and have outcome value y = 1, we call this pm1

and have outcome value y = 2, we call this pm2

and so on…

The predicted value for node m is then arg $\max_k \{p_{mk}\}$ i.e. the value of k with the largest p_{mk} .

error at node m : 1- $\max_k \{p_{mk}\}$.

3. Neural Network[(5)]

Neural networks are multi-layer networks of neurons that we use to classify things, make predictions.

The network consists of many hidden layers of neuron such as the figure 3 . to train the network, we pick a sample from the training dataset and apply its feature on each neuron of the input layer to get an output of the form

$$z_1^{(1)} = w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + b_1^{(1)}.$$

then we apply activation function such as sigmoid to activate the neuron if the output above a certain level.

$$\sigma(v) = \frac{1}{1 + e^{-v}}.$$

And then we repeat the process on each neuron of the second hidden layers with different parameters w. and then we move to another example of the training data.

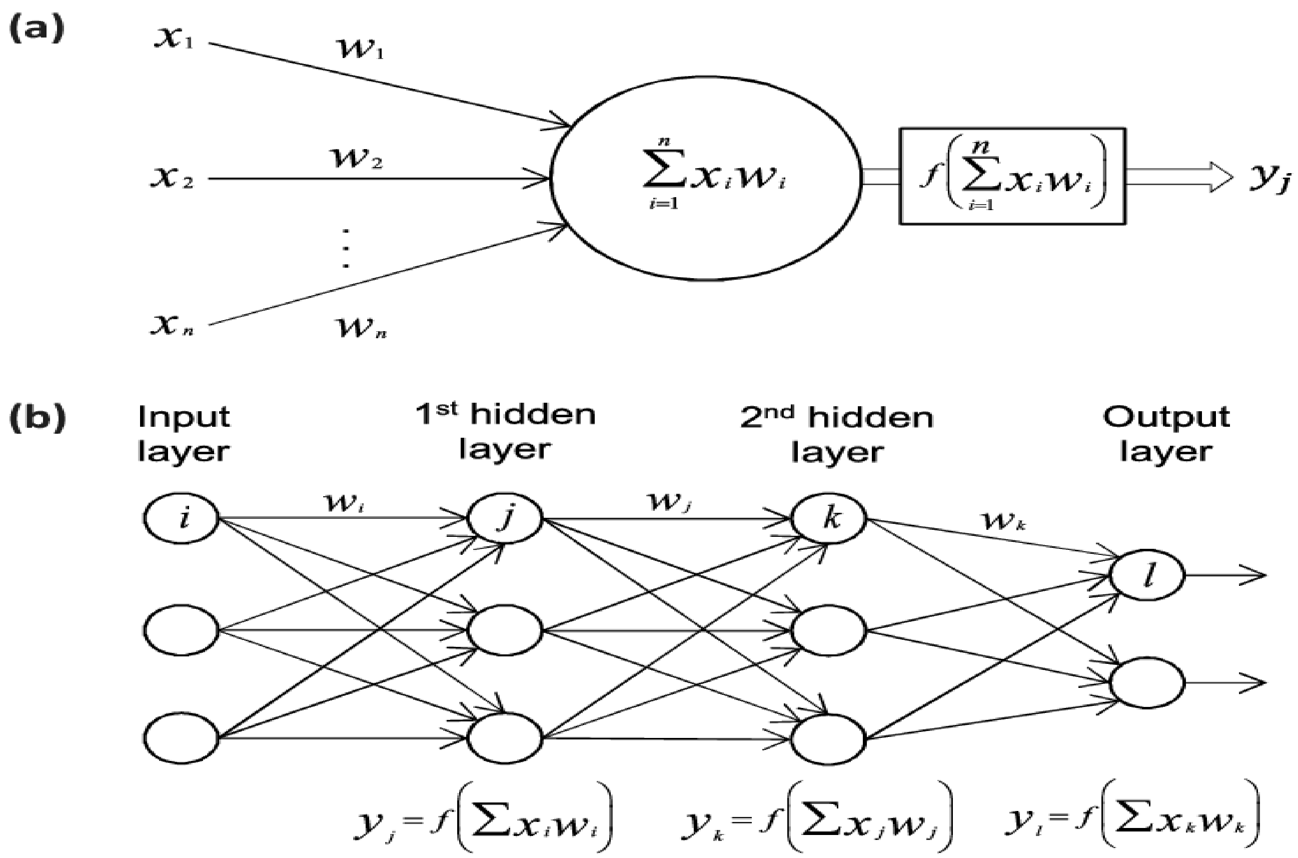


Figure 3: Neural Network Example

the output layer has k neurons as we have k classes in the classification, and each neuron is activated according to the class of the selected sample.

By training the network, we try to modify the parameters w in a way that reduce the loss of the output using a back propagation technique.

The loss in classification neural network is given by the cross-entropy function:

$$R_i = -\sum_{k=1}^{K} y_{ik} \log(\pi_k(\boldsymbol{x}_i)).$$

Practical Part:

1- Support Vector Machine (SVM) using R

- ➔ 1- we import the data set into the R studio,
- ➔ 2- as the labels are stored in the dataset in 7 columns (one-hot encoding), we compact these 7 columns in one column targetVar such that the classes are encoded as integer from 1 to 7, then we covert this column to be a factor.

  The new dataset consists of 27 input variables and one target(7 classes).

  Dim (1941, 28)

- ➔ we do some basic analysis to check the name of the variables and if there is a missing values.
- ➔ We split the independent variables in a data frame steel_ds_x and the target in steel_ds_y. Then we check the percentage of samples per class.
- ➔ We try to understand the data better by visualizing the density of the independent variables shown in figure 4.
- ➔ we look at the density of each variable per class shown in figure 5

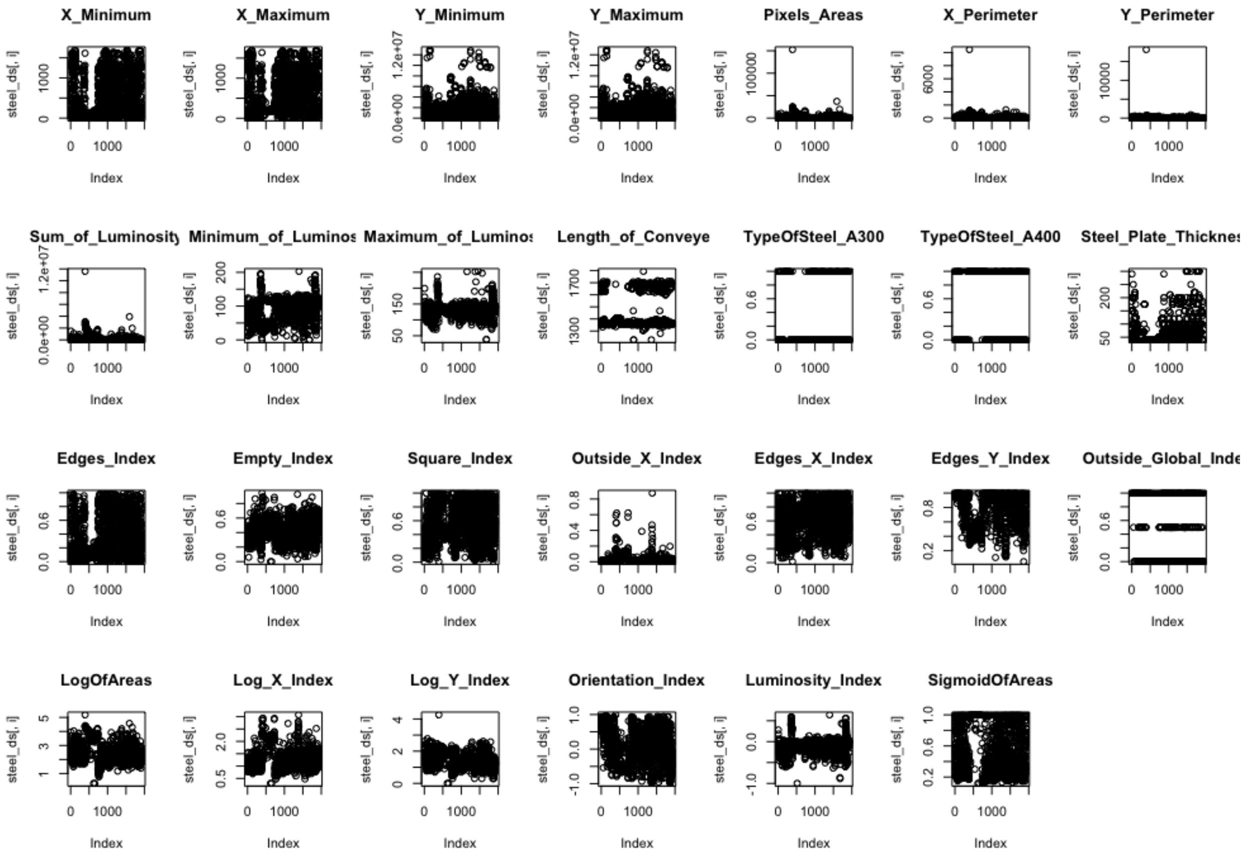


Figure 4: density of the independent variables

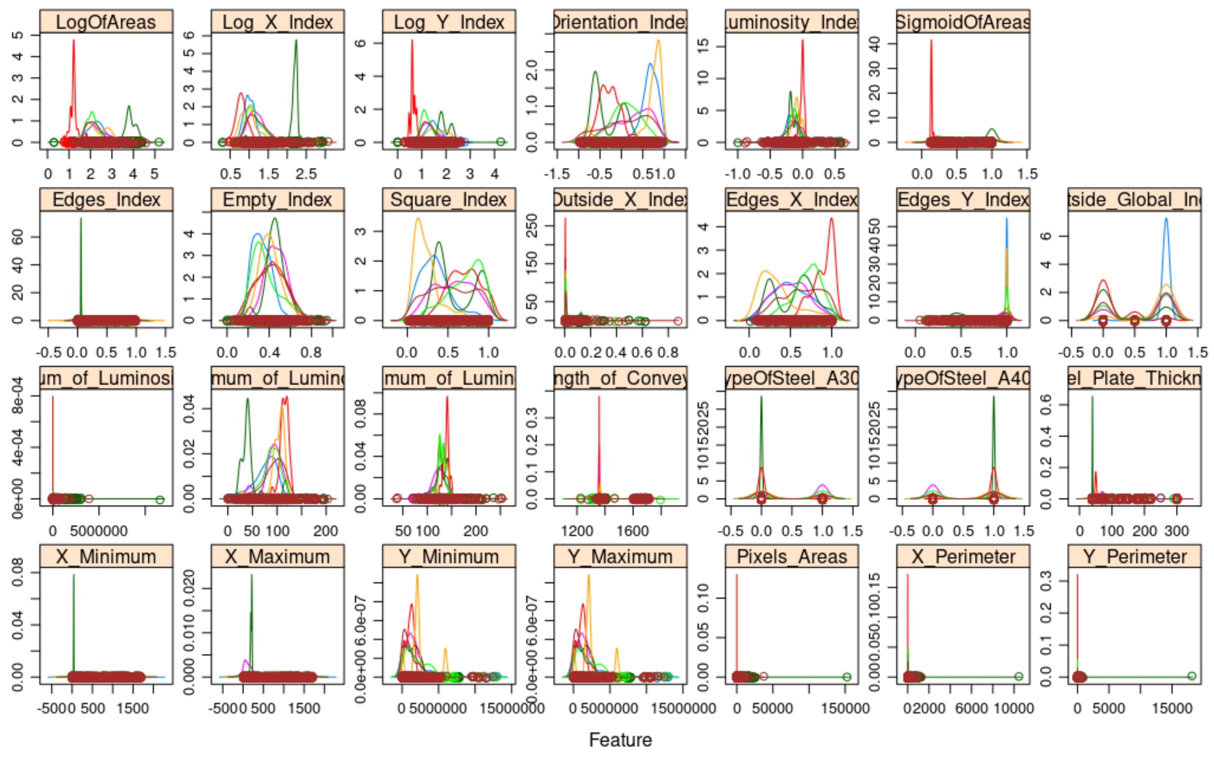


Figure 5: density of each variable per class

➔ we look at the Correlation plot of x_variables to each other in figure 6

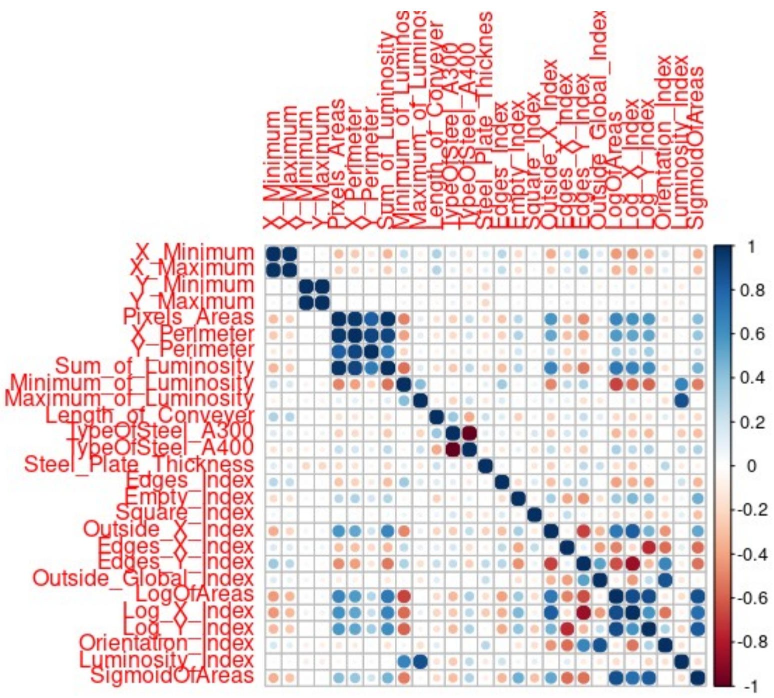


Figure 6: coorelation between the variables

➔ split the data into training 80% and test 20%

➔ we scale the training data

➔ we fit the training data into SVM model with kernel = 'radial',gamma = 0.1 and check, and then apply it on the test data and check the error between y and y_predict.

➔ At this step, we train the model again after applying cross_validation on the training data (K=10 folds) (here we do not use test data any more) and the same HP : kernel = 'radial',gamma = 0.1, we check after that the accuracy in the validation dataset over the k folds and we get:

acc= 0.9941176

➔ now let us tune the HP gamme between 0.5 and 2 and retrain the model using the best value,

we see that the best gamma=0.5 with cost =4 and error = 2.863407 as show in the figure7:

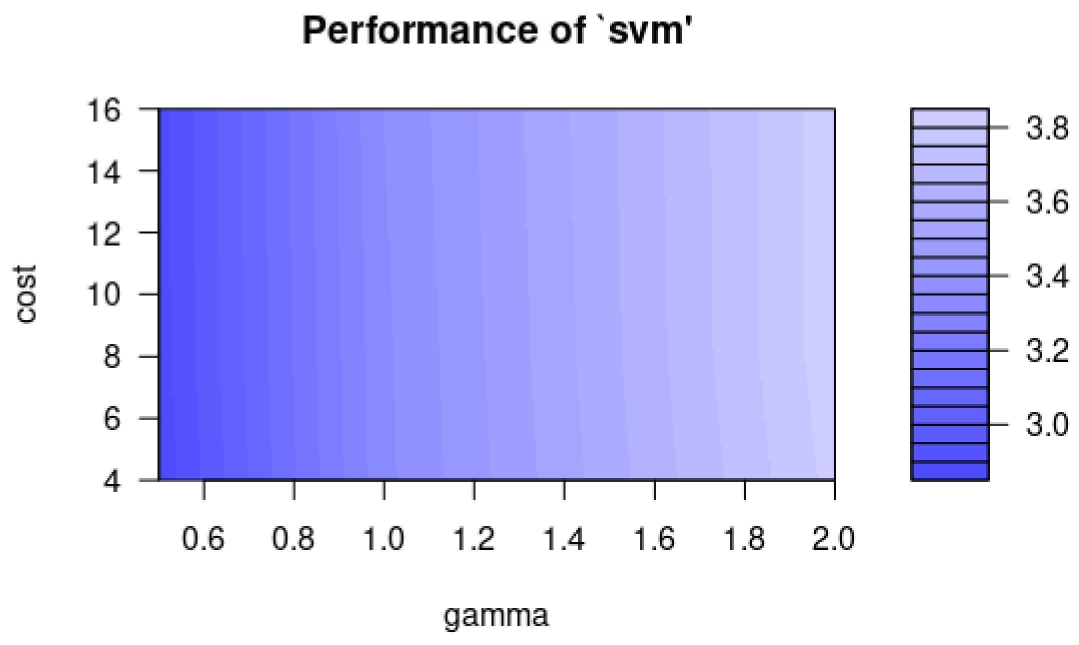


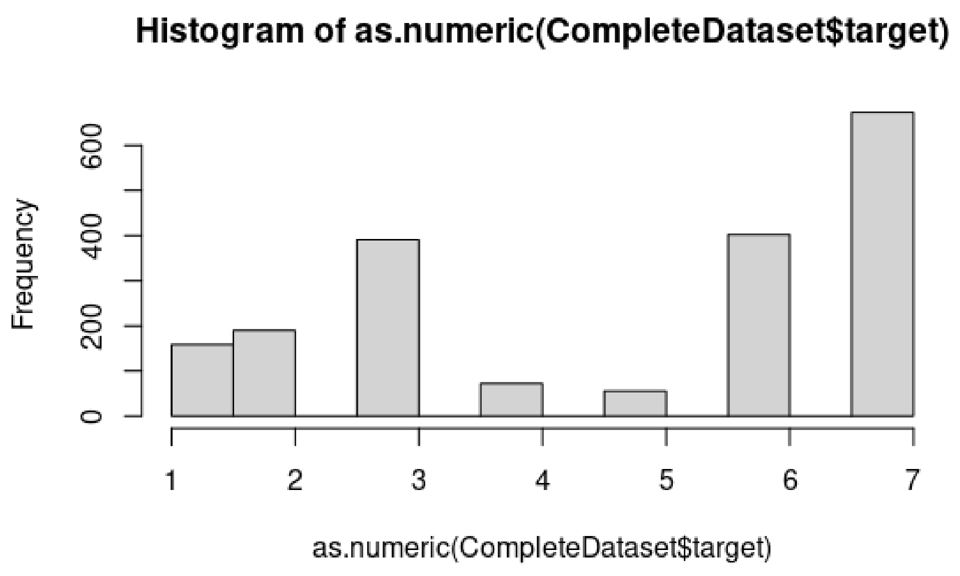
Figure 7: tuning gamma in SVM

➔ by apply this final model on the test data we get **acc = 0.9964286**

2- Decision Tree Model using R:

➔ we use the same basic analysis used in SVM,

➔ we compact the labels in one column again



➔ we look at the distribution of samples per classes the first approach, we just split the data 80% train and 20% test and then we create a tree model and train it on a training data using cross validation of 10 folds , and then we apply this model1 on the test dataset :

we get **Accuracy1= 60.6**

which is not good

➔ **the second approach**, before building the tree model we apply PCA on the data to see the most important variables on the classes

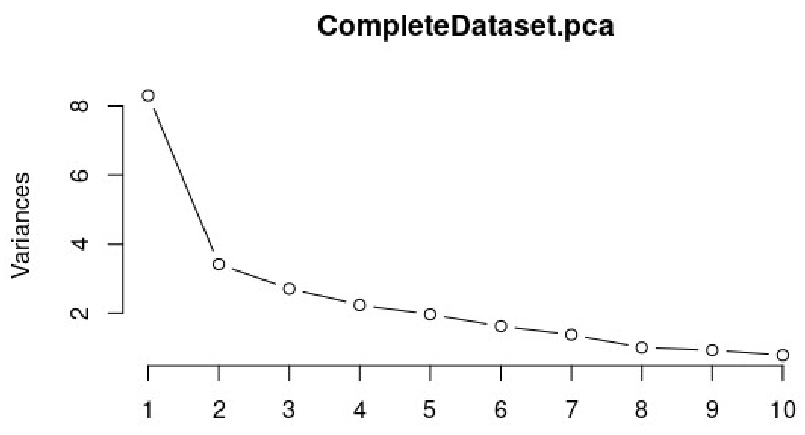
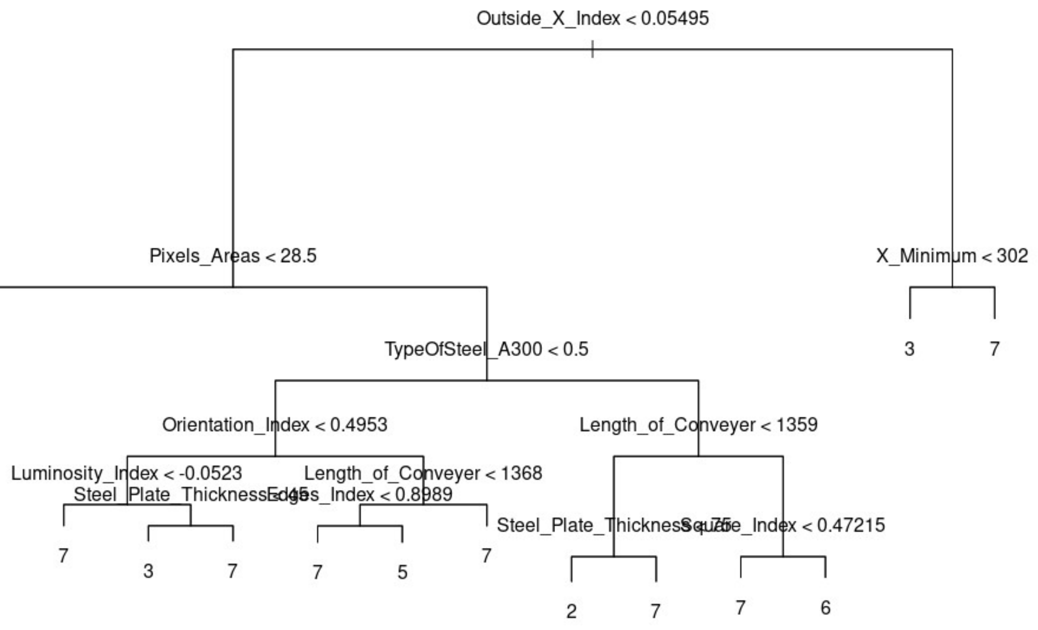


Figure 8: the variance of the PCs

we can see in the figure 8 and **summary(CompleteDataset.pca)** that the first 10 pcs can explain 90% of the data , by using these pcs and then build a tree model2

we get **Accuracy2= 53.1**

➔ **the third approach,** we create the tree model states in figure 9



Outside_X_Index < 0.05495
Pixels_Areas < 28.5
X_Minimum < 302
3
7
TypeOfSteel_A300 < 0.5
4
Orientation_Index < 0.4953
Length_of_Conveyer < 1359
Luminosity_Index < -0.0523
Length_of_Conveyer < 1368
Steel_Plate_Thickness < 45
Edges_Index < 0.8989
Steel_Plate_Thickness < 75
Square_Index < 0.47215
7
3
7
7
5
7
2
7
7
6

Figure 9: Model 3

and then we train the model using cross validation based on missed classification after that we evaluate the model3 in the test data

we get **Accuracy3= 0.6666667**

which is better than than the first two approaches.

➔ In this four approach we use cross validation based on prune miss classification error to train the tree Model4 illustrated in the figure 10

we get **Accuracy4= 0.6575964**

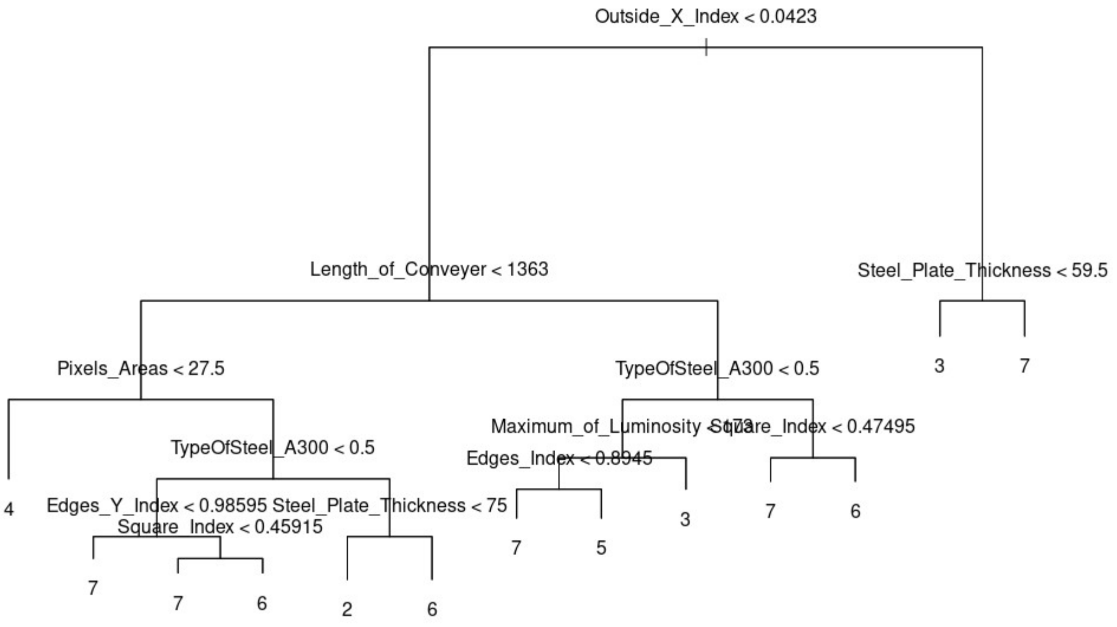


Figure 10: Tree Model 4

comparing the four approaches together, the best acc we get is **66.6% which is not good**

Table 3: compare the acc in the 4 tree models

Without_PCA	With_PCA	CV_full_tree	CV_prune_tree
60.6%	**53.1 %**	**66.6%**	**65.75%**

3. Neural Network using python

we follow a bit different approach in the neural network, first we use python instead of R.

- ➔ by checking missing values and duplicates, there is no missing or duplicates.
- ➔ by looking at the data type of the variables, it can be seen that all of them (including the labels) are numeric (float64 or int64)
- ➔ split the input variables in a data frame and the labels in another one.
- ➔ The input variables are normalized using MinMax to be in the range [0,1]
- ➔ by checking the distribution of the samples per classes, it can be seen in figure11 that some classes have a few values such as stains and dirtiness, which means that the data is not balanced .

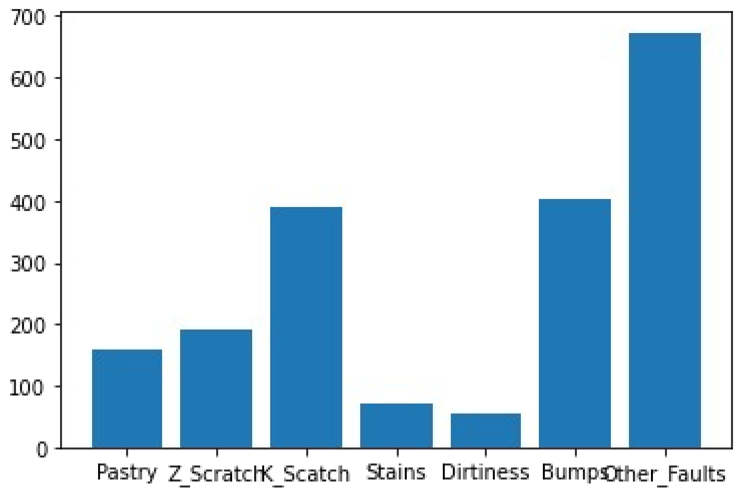
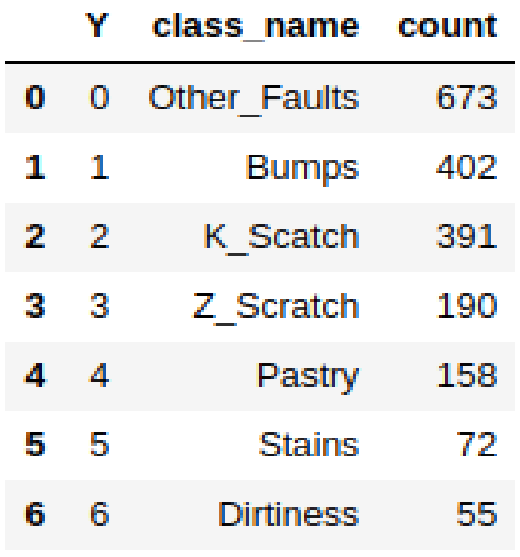


Figure 11: distribution of the samples per classes

therefore we create more samples for these classes to make the balanced data.

- ➔ To do this, first we compact the labels from one_hot encoding into one column Y such that the classes is encoded from 0 to 6 as shown in the Figure 12



	Y	class_name	count
0	0	Other_Faults	673
1	1	Bumps	402
2	2	K_Scatch	391
3	3	Z_Scratch	190
4	4	Pastry	158
5	5	Stains	72
6	6	Dirtiness	55

Figure 12: matching the classes in one column

➔ then we create more samples to get a new balanced dataset of the size 4711 rows (27 variables, and 1 target)

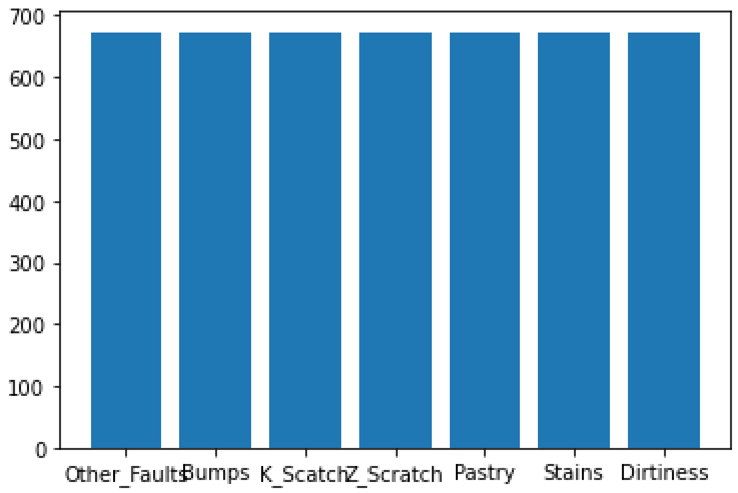


Figure 13: distribution of the new samples per classes

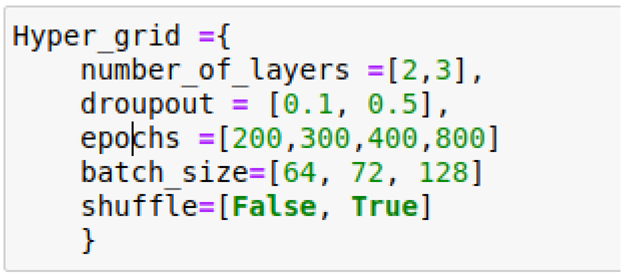
➔ now we create one-hot encoding of the label column Y.

➔ at the end of this data preparation, we get input data X (4711,27), and labels y_one_hot (4711,7) which will be used in the modeling part

➔ Split the data : train 60%, validation 20%, test 20%

```
size train (2826, 27)
size val (942, 27)
size test (943, 27)
```

➔ Modeling: we create the neural network and we try many values of a grid of hyper parameters and we fit the model into the training data for each combination of hyper_grid, and we check the loss and accuracy in the validation data.



```
Hyper_grid ={
    number_of_layers =[2,3],
    droupout = [0.1, 0.5],
    epochs =[200,300,400,800]
    batch_size=[64, 72, 128]
    shuffle=[False, True]
    }
```

Figure 14: hyper-parameters grid

➔ The best combination of HP is

```
number_of_layers =3,
drop-out = 0.1
epochs =300
batch_size= 128
shuffle= False
```

which results the loss and accuracy in the train and validation shown in figure15:

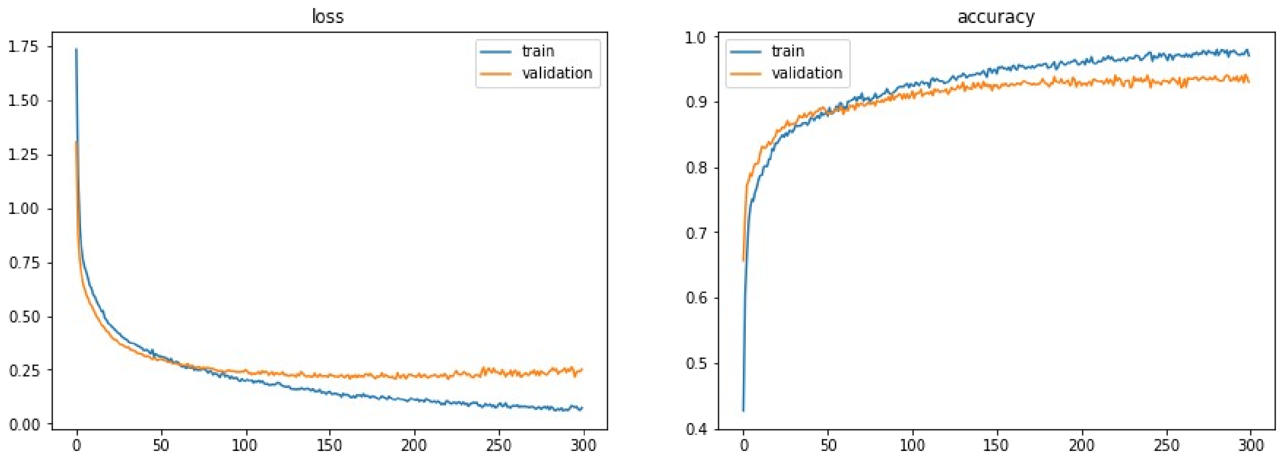


Figure 15: loss and acc of the best HP

➔ by using these HP, we create our final model and we fit it into the training data, and finally we apply it on the test data to predict the classes,

➔ by computing the accuracy in the test data we get:

acc = 92.47%

Compare the methods:

	SVM	Decision Tree	Neural Network
Acc	**99,64%**	**66.6%**	**92.47%**

by comparing the three methods we can see clearly that the support vector machine SVM performs much better than the the other two as it achieves an accuracy of 99%, on the other hand, Decision tree with this analysis does not perform well on this data.

Sources :

1- Research Center of Sciences and Communication:

http://www.semeion.it/wordpress/

2- The dataset on UCI

https://archive.ics.uci.edu/ml/datasets/Steel+Plates+Faults

3- Support Vector Machine (SVM)

https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496

4- Decision Tree :

https://www.saedsayad.com/decision_tree.htm

5- Neural Network

https://towardsdatascience.com/understanding-neural-networks-19020b758230