Predictive analysis and modeling of disease classes using genetic microarray data

Hamdan, Ahmad
ahamdan16@ubishops.ca

Bill Morrisson, Talla
btalla18@ubishops.ca

Karimifar, Niloufar
nkarimifar19@ubishops.ca

Nourzad, Ali
anourzad19@ubishops.ca

Dept. of Computer Science
Bishop's University
Sherbrooke, Canada

Submitted to Dr. Layachi Bentabet

in conformity with the requirements for

the capstone research project(CS590)

and

the degree of Master of Science

Abstract:

Using machine learning approaches to develop a method that uses genetic microarray data to predict disease classes. This study aims to evaluate different machine learning techniques in prediction of disease.

The dataset is extracted from a DNA microarray which measures the expression levels of large numbers of genes simultaneously. Samples in the datasets represent patients; for each patient 7070 genes expressions (values) are measured in order to classify the patient's disease into one of the following cases: EPD, JPA, MED, MGL, RHB.

Keywords— disease prediction; Machine Learning Algorithm; genetic microarray.

I. INTRODUCTION

In recent years, machine-learning techniques such as classification trees and artificial neural networks (ANN) have been used as prediction, classification, and diagnosis tools. Machine-learning techniques are used in the medical approaches to help using an invasive method in prediction and detection of diseases.

We will start with an overview of the common tasks in a machine learning project. A predictive modeling machine learning project can be broken down into main processes:

Data Analysis & Preparation

- Define Problem: Investigate and characterize the problem in order to better understand the goals of the project.
- Analyze Data: Use descriptive statistics and visualization to better understand the data we have available.
- Prepare Data: Use data transforms in order to better expose the structure of the prediction problem to modeling algorithms.

ML Modeling

- Evaluate Algorithms: Design a test harness to evaluate a number of standard algorithms on the data and select the top few to investigate further.
- Improve Results: Use algorithm tuning and ensemble methods to get the most out of well-performing algorithms on the data.

- Present Results: Finalize the model, make predictions and present results.

II. DATA ANALYSIS & PREPARATION

A. Understand the Data: We must understand the data in order to get the best results. The Data has 24 features as:

```
SNO     object
S101     int64
S102     int64
S103     int64
S104     int64
S105     int64
S106     int64
S107     int64
S108     int64
S109     int64
S110     int64
S111     int64
S112     int64
S113     int64
S114     int64
S115     int64
S116     int64
S117     int64
S118     int64
S119     int64
S120     int64
S121     int64
S122     int64
S123     int64
dtype: object
```

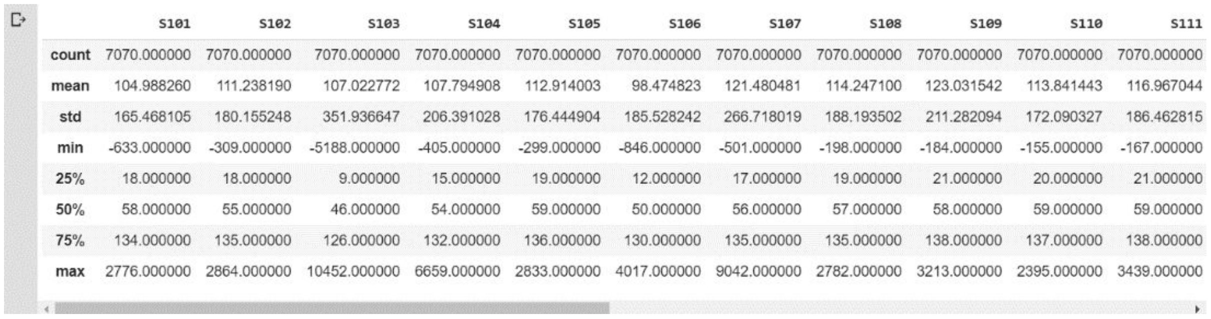
Note: S.. indicate to the name of the feature.

Here are the dimensions of our data:

```
(7070, 24)
```

__B. Descriptive Statistics__

We now have a better feeling for how different the attributes are. The min and max values as well as the means vary a lot. We are likely going to get better results by rescaling the data in some way

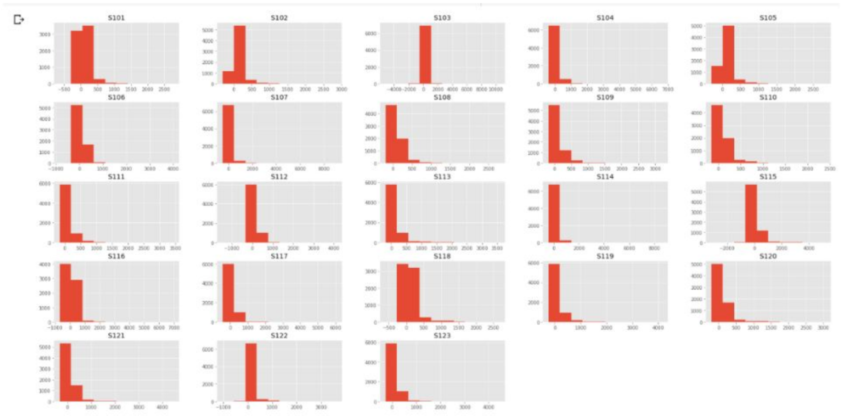


	S101	S102	S103	S104	S105	S106	S107	S108	S109	S110	S111
count	7070.000000	7070.000000	7070.000000	7070.000000	7070.000000	7070.000000	7070.000000	7070.000000	7070.000000	7070.000000	7070.000000
mean	104.988260	111.238190	107.022772	107.794908	112.914003	98.474823	121.480481	114.247100	123.031542	113.841443	116.967044
std	165.468105	180.155248	351.936647	206.391028	176.444904	185.528242	266.718019	188.193502	211.282094	172.090327	186.462815
min	-633.000000	-309.000000	-5188.000000	-405.000000	-299.000000	-846.000000	-501.000000	-198.000000	-184.000000	-155.000000	-167.000000
25%	18.000000	18.000000	9.000000	15.000000	19.000000	12.000000	17.000000	19.000000	21.000000	20.000000	21.000000
50%	58.000000	55.000000	46.000000	54.000000	59.000000	50.000000	56.000000	57.000000	58.000000	59.000000	59.000000
75%	134.000000	135.000000	126.000000	132.000000	136.000000	130.000000	135.000000	135.000000	138.000000	137.000000	138.000000
max	2776.000000	2864.000000	10452.000000	6659.000000	2833.000000	4017.000000	9042.000000	2782.000000	3213.000000	2395.000000	3439.000000

__C. Understand the Data With Visualization__

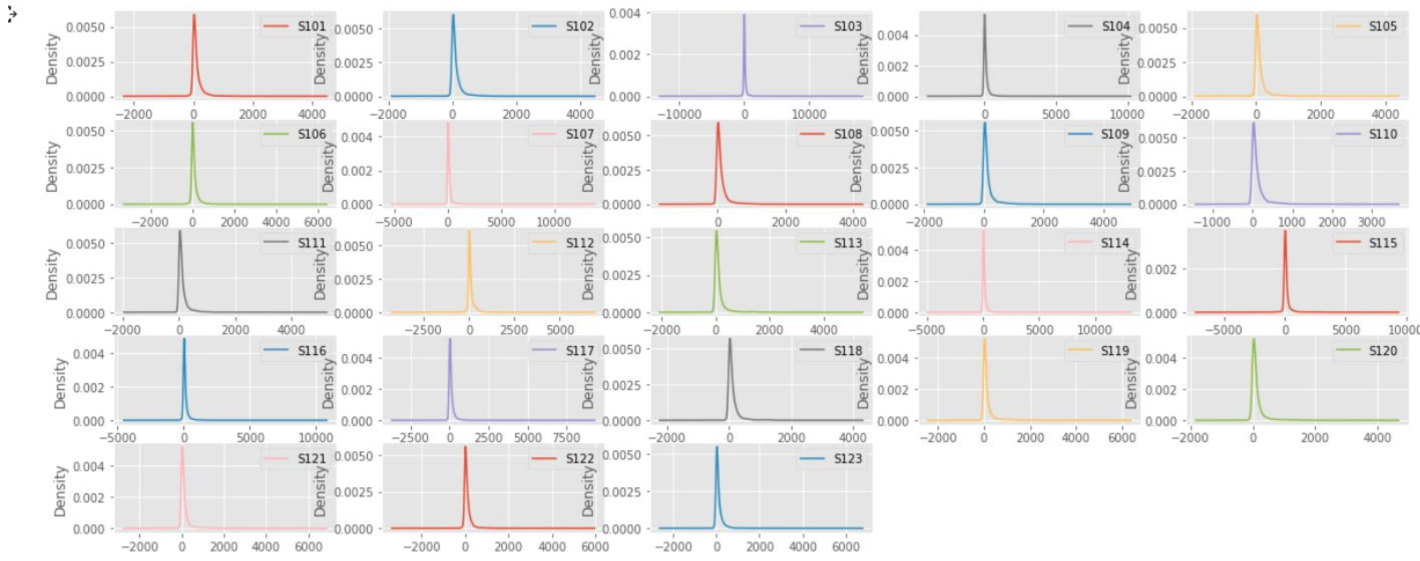
Below is Univariate Histograms for some of the attributes.

We can see that perhaps some attributes have a Gaussian or nearly Gaussian distribution. This is interesting because many machine learning techniques assume a Gaussian univariate distribution on the input variables. This is useful to note as we can use algorithms that can exploit the Gaussian distribution. Also, it also looks like some attributes may be skewed Gaussian distributions, which might be helpful later with transforms.

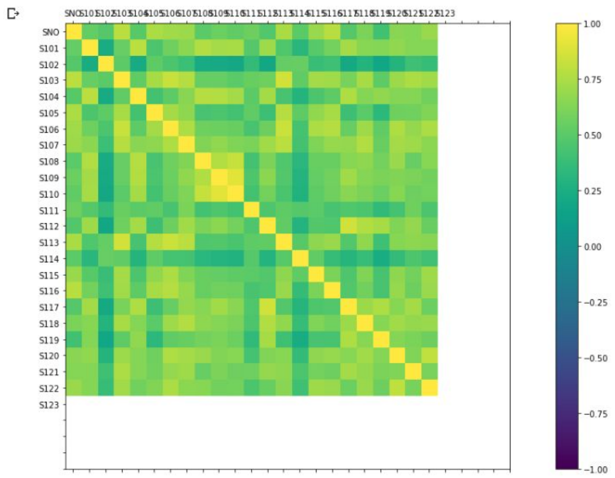


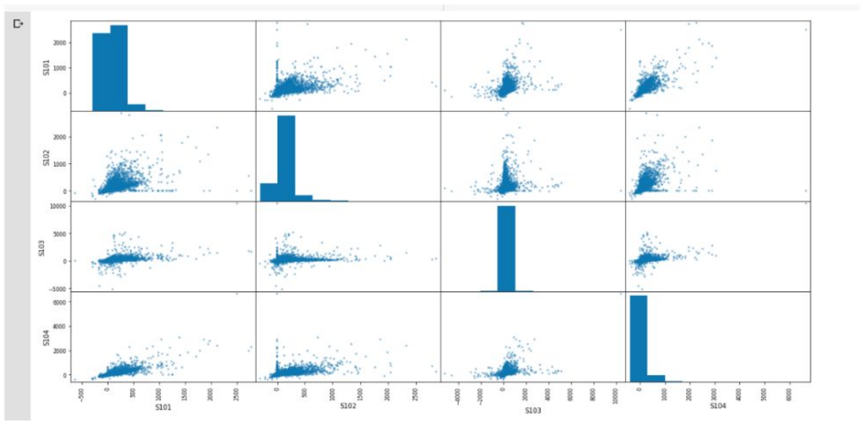
Density Plots

Density plots are another way of getting a quick idea of the distribution of each attribute.



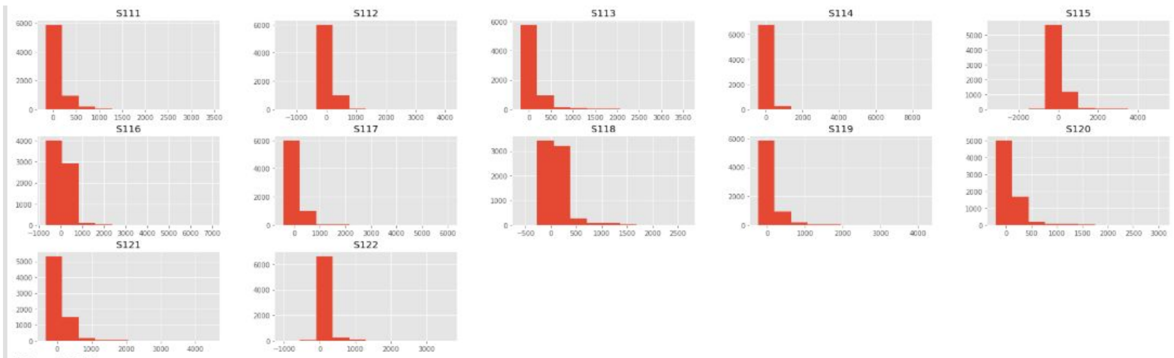
The Scatter Plot was created for each pair of attributes in the data for spotting structured relationships between variables, like whether we could summarize the relationship between two variables with a line.





D. Log transform certain features (skewed distribution)

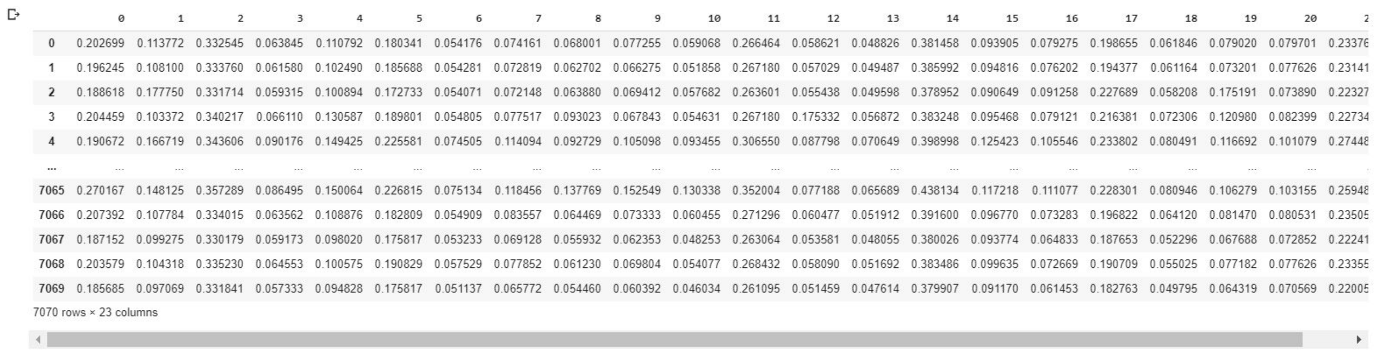
We will apply log transformation to those attributes who have a skewed distribution. After log transformation is applied, we notice that now we do not have any attribute with skew distribution as show below:



```
S101     4.876644
S102     4.754997
S103     6.715070
S104     9.080761
S105     4.490358
S106     6.401698
S107    10.531303
S108     4.695266
S109     5.361007
S110     4.109216
S111     5.060773
S112     4.742946
S113     4.671302
S114    11.793470
S115     4.428517
S116     9.260182
S117     9.956578
S118     4.089250
S119     5.421232
S120     4.763273
S121     5.951807
S122     5.386277
S123     6.580753
dtype: float64
```

E. Rescale Data

When the data is composed of attributes with varying scales, many machine learning algorithms can benefit from rescaling the attributes to all have the same scale. Often this is referred to as normalization and attributes are often rescaled into the range between 0 and 1. This is useful for optimization algorithms used in the core of machine learning algorithms like gradient descent. It is also useful for algorithms that weigh inputs like regression and neural networks and algorithms that use distance measures like k-Nearest Neighbors. We can rescale the data using scikit-learn using the MinMaxScaler class



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	2
0	0.202699	0.113772	0.332545	0.063845	0.110792	0.180341	0.054176	0.074161	0.068001	0.077255	0.059068	0.266464	0.058621	0.048826	0.381458	0.093905	0.079275	0.198655	0.061846	0.079020	0.079701	0.23376
1	0.196245	0.108100	0.333760	0.061580	0.102490	0.185688	0.054281	0.072819	0.062702	0.066275	0.051858	0.267180	0.057029	0.049487	0.385992	0.094816	0.076202	0.194377	0.061164	0.073201	0.077626	0.23141
2	0.188618	0.177750	0.331714	0.059315	0.100894	0.172733	0.054071	0.072148	0.063880	0.069412	0.057682	0.263601	0.055438	0.049598	0.378952	0.090649	0.091258	0.227689	0.058208	0.175191	0.073890	0.22327
3	0.204459	0.103372	0.340217	0.066110	0.130587	0.189801	0.054805	0.077517	0.093023	0.067843	0.054631	0.267180	0.175332	0.056872	0.383248	0.095468	0.079121	0.216381	0.072306	0.120980	0.082399	0.22734
4	0.190672	0.166719	0.343606	0.090176	0.149425	0.225581	0.074505	0.114094	0.092729	0.105098	0.093455	0.306550	0.087798	0.070649	0.398998	0.125423	0.105546	0.233802	0.080491	0.116692	0.101079	0.27448
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
7065	0.270167	0.148125	0.357289	0.086495	0.150064	0.226815	0.075134	0.118456	0.137769	0.152549	0.130338	0.352004	0.077188	0.065689	0.438134	0.117218	0.111077	0.228301	0.080946	0.106279	0.103155	0.25948
7066	0.207392	0.107784	0.334015	0.063562	0.108876	0.182809	0.054909	0.083557	0.064469	0.073333	0.060455	0.271296	0.060477	0.051912	0.391600	0.096770	0.073283	0.196822	0.064120	0.081470	0.080531	0.23505
7067	0.187152	0.099275	0.330179	0.059173	0.098020	0.175817	0.053233	0.069128	0.055932	0.062353	0.048253	0.263064	0.053581	0.048055	0.380026	0.093774	0.064833	0.187653	0.052296	0.067688	0.072852	0.22241
7068	0.203579	0.104318	0.335230	0.064553	0.100575	0.190829	0.057529	0.077852	0.061230	0.069804	0.054077	0.268432	0.058090	0.051692	0.383486	0.099635	0.072669	0.190709	0.055025	0.077182	0.077626	0.23355
7069	0.185685	0.097069	0.331841	0.057333	0.094828	0.175817	0.051137	0.065772	0.054460	0.060392	0.046034	0.261095	0.051459	0.047614	0.379907	0.091170	0.061453	0.182763	0.049795	0.064319	0.070569	0.22005

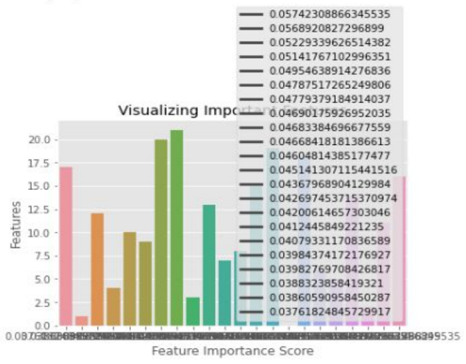
7070 rows × 23 columns

G. Standardize Data

When the data is composed of attributes with varying scales, many machine learning algorithms can benefit from rescaling the attributes to all have the same scale. Often this is referred to as normalization and attributes are often rescaled into the range between 0 and 1. This is useful for optimization algorithms used in the core of machine learning algorithms like gradient descent. It is also useful for algorithms that weigh inputs like regression and neural networks and algorithms that use distance measures like k-Nearest Neighbors.

F. Finding Important Features Using RandomForestClassifierWe have created a random forests model & used the feature importance variable to see feature importance scores. Then we visualized these scores as below:

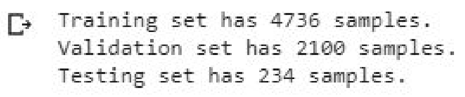
```
Finding Important Features
```



```
Importance of Features:

 16    0.057423
11    0.056892
2     0.052293
14    0.051418
5     0.049546
6     0.047875
18    0.047794
0     0.046902
19    0.046834
15    0.046684
8     0.046048
7     0.045141
13    0.043680
3     0.042697
21    0.042006
20    0.041245
9     0.040793
10    0.039844
4     0.039828
12    0.038832
1     0.038606
17    0.037618
```

H. Create a Validation & Test Dataset

We will split the loaded dataset into two, 67% of which we will use to train our models and 33% that we will hold back as a validation & test dataset.



```
Training set has 4736 samples.
Validation set has 2100 samples.
Testing set has 234 samples.
```

L. Build Models:

We don't know which algorithms would be good on this problem or what configurations to use. We get an idea from the plots that some of the classes are partially linearly separable in some dimensions, so we are expecting generally good results. Let's evaluate six different algorithms:

- Logistic Regression (LR).
- Linear Discriminant Analysis (LDA).
- k-Nearest Neighbors (KNN).
- Classification and Regression Trees (CART).
- Gaussian Naive Bayes (NB).
- Support Vector Machines (SVM).

This list is a good mixture of simple linear (LR and LDA), nonlinear (KNN, CART, NB and SVM) algorithms. We reset the random number seed before each run to ensure that the evaluation of each algorithm is performed using exactly the same data splits.

M. Spot-Check Classification Algorithms:

Spot-checking is a way of discovering which algorithms perform well on our machine learning problem. We cannot know which algorithms are best suited to our problem beforehand. We must try a number of methods and focus attention on those that prove themselves the most promising. We will discover six machine learning algorithms that we can use when spot-checking our classification problem in Python with scikit-learn:

1. How to spot-check machine learning algorithms on a classification problem.
2. How to spot-check two linear classification algorithms.
3. How to spot-check four nonlinear classification algorithms.

We are going to take a look at six classification algorithms that we can spot-check on our dataset. Starting with two linear machine learning algorithms:

1. **Logistic Regression.** Logistic regression assumes a Gaussian distribution for the numeric input variables and can model binary classification problems. we can construct a logistic regression model using the LogisticRegression class1.
2. **Linear Discriminant Analysis.** Linear Discriminant Analysis or LDA is a statistical technique for binary and multiclass classification. It too assumes a Gaussian distribution for the numerical input variables. We can construct an LDA model using the LinearDiscriminantAnalysis class

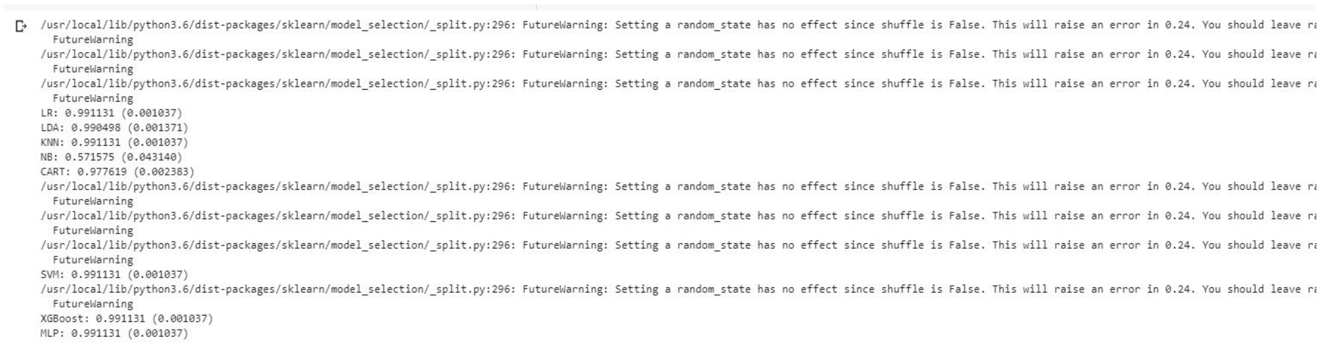
Then looking at **four nonlinear machine learning algorithms**:

- **k-Nearest Neighbors:**The k-Nearest Neighbors algorithm (or KNN) uses a distance metric to find the k most similar instances in the training data for a new instance and takes the mean outcome of the neighbors as the prediction. You can construct a KNN model using the KNeighborsClassifier class

- **Naive Bayes:** Naive Bayes calculates the probability of each class and the conditional probability of each class given each input value. These probabilities are estimated for new data and multiplied together, assuming that they are all independent (a simple or naive assumption). When working with real-valued data, a Gaussian distribution is assumed to easily estimate the probabilities for input variables using the Gaussian Probability Density Function. We can construct a Naive Bayes model using the GaussianNB class

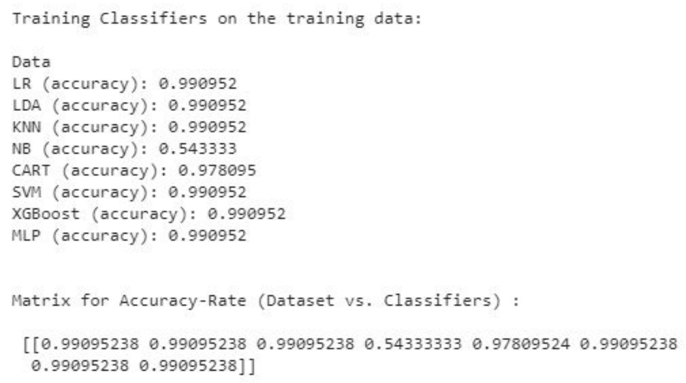
- **Classification and Regression Trees:** Classification and Regression Trees (CART or just decision trees) construct a binary tree from the training data. Split points are chosen greedily by evaluating each attribute and each value of each attribute in the training data in order to minimize a cost function (like the Gini index). We can construct a CART model using the DecisionTreeClassifier class

- **Support Vector Machines:** Support Vector Machines (or SVM) seek a line that best separates two classes. Those data instances that are closest to the line that best separates the classes are called support vectors and influence where the line is placed. SVM has been extended to support multiple classes. Of particular importance is the use of different kernel functions via the kernel parameter. A powerful Radial Basis Function is used by default. We can construct an SVM model using the SVC class



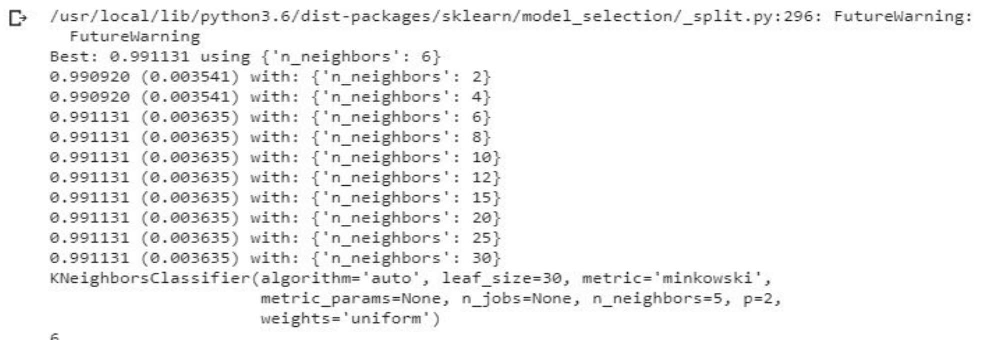
```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning: Setting a random_state has no effect since shuffle is False. This will raise an error in 0.24. You should leave ra
  FutureWarning
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning: Setting a random_state has no effect since shuffle is False. This will raise an error in 0.24. You should leave ra
  FutureWarning
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning: Setting a random_state has no effect since shuffle is False. This will raise an error in 0.24. You should leave ra
  FutureWarning
LR: 0.991131 (0.001037)
LDA: 0.990498 (0.001371)
KNN: 0.991131 (0.001037)
NB: 0.571575 (0.043140)
CART: 0.977619 (0.002383)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning: Setting a random_state has no effect since shuffle is False. This will raise an error in 0.24. You should leave ra
  FutureWarning
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning: Setting a random_state has no effect since shuffle is False. This will raise an error in 0.24. You should leave ra
  FutureWarning
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning: Setting a random_state has no effect since shuffle is False. This will raise an error in 0.24. You should leave ra
  FutureWarning
SVM: 0.991131 (0.001037)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning: Setting a random_state has no effect since shuffle is False. This will raise an error in 0.24. You should leave ra
  FutureWarning
XGBoost: 0.991131 (0.001037)
MLP: 0.991131 (0.001037)
```

Training Classifiers on the training data:



```
Training Classifiers on the training data:

Data
LR (accuracy): 0.990952
LDA (accuracy): 0.990952
KNN (accuracy): 0.990952
NB (accuracy): 0.543333
CART (accuracy): 0.978095
SVM (accuracy): 0.990952
XGBoost (accuracy): 0.990952
MLP (accuracy): 0.990952


Matrix for Accuracy-Rate (Dataset vs. Classifiers) :

 [[0.99095238 0.99095238 0.99095238 0.54333333 0.97809524 0.99095238
  0.99095238 0.99095238]]
```

Tuning KNN:



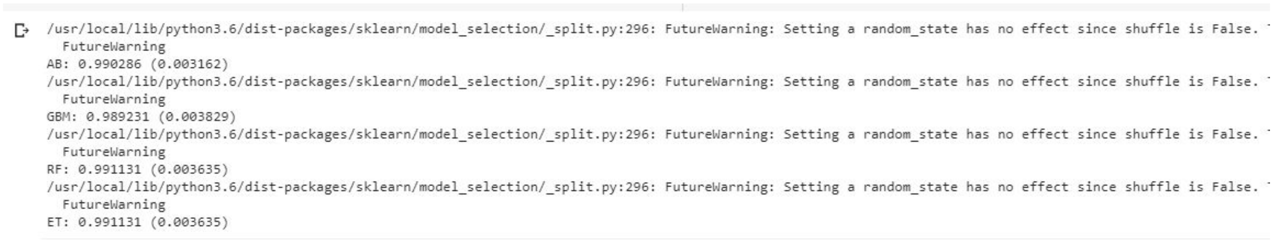
```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning:
  FutureWarning
Best: 0.991131 using {'n_neighbors': 6}
0.990920 (0.003541) with: {'n_neighbors': 2}
0.990920 (0.003541) with: {'n_neighbors': 4}
0.991131 (0.003635) with: {'n_neighbors': 6}
0.991131 (0.003635) with: {'n_neighbors': 8}
0.991131 (0.003635) with: {'n_neighbors': 10}
0.991131 (0.003635) with: {'n_neighbors': 12}
0.991131 (0.003635) with: {'n_neighbors': 15}
0.991131 (0.003635) with: {'n_neighbors': 20}
0.991131 (0.003635) with: {'n_neighbors': 25}
0.991131 (0.003635) with: {'n_neighbors': 30}
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
6
```

Ensemble Methods

Another way that we can improve the performance of algorithms on this problem is by using ensemble methods. In this section we will evaluate four different ensemble machine learning algorithms, two boosting and two bagging methods:

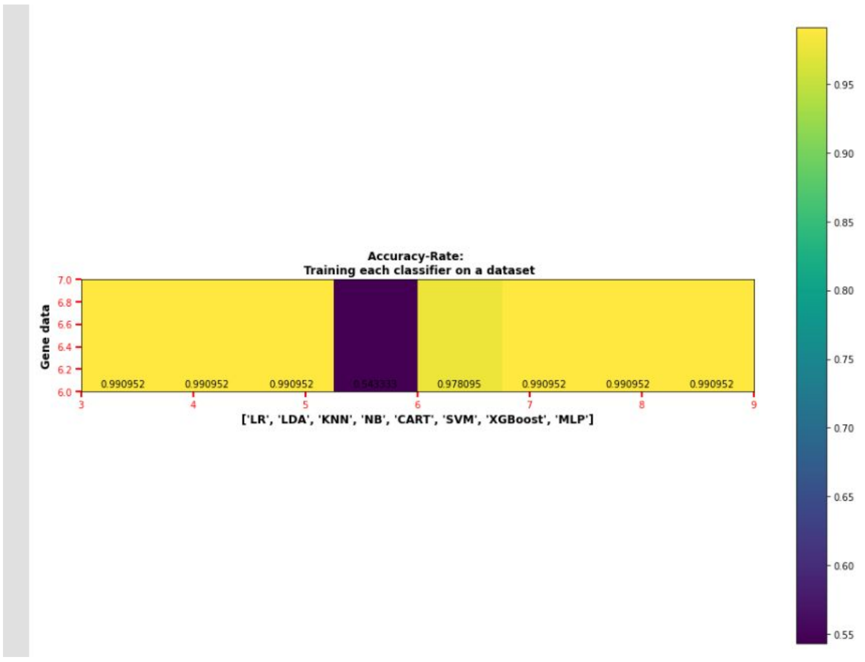
- Boosting Methods: AdaBoost (AB) and Gradient Boosting (GBM).
- Bagging Methods: Random Forests (RF) and Extra Trees (ET).

We will use the same test harness as before, 10-fold cross validation. No data standardization is used in this case because all four ensemble algorithms are based on decision trees that are less sensitive to data distributions.



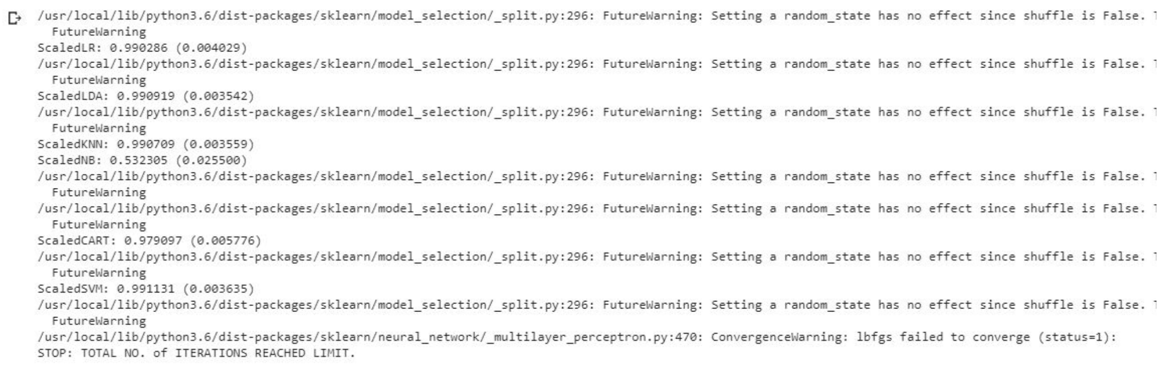
```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning: Setting a random_state has no effect since shuffle is False.
  FutureWarning
AB: 0.990286 (0.003162)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning: Setting a random_state has no effect since shuffle is False.
  FutureWarning
GBM: 0.989231 (0.003829)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning: Setting a random_state has no effect since shuffle is False.
  FutureWarning
RF: 0.991131 (0.003635)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning: Setting a random_state has no effect since shuffle is False.
  FutureWarning
ET: 0.991131 (0.003635)
```

Plotting Accuracy Matrix (Dataset vs. Classifiers)



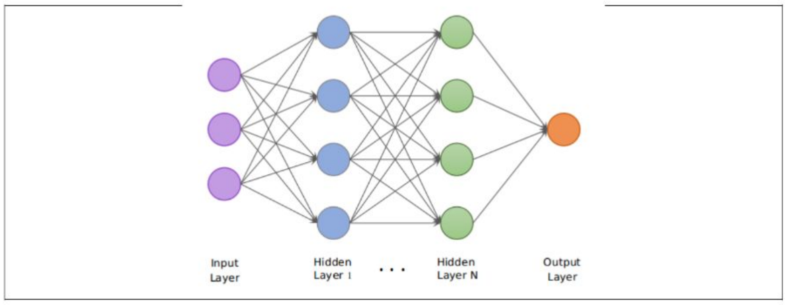
Evaluate Algorithms: Standardize Data

We suspect that the differing distributions of the raw data may be negatively impacting the skill of some of the algorithms. Let's evaluate the same algorithms with a standardized copy of the dataset. This is where the data is transformed such that each attribute has a mean value of zero and a standard deviation of one. We also need to avoid data leakage when we transform the data. A good way to avoid leakage is to use pipelines that standardize the data and build the model for each fold in the cross validation test harness. That way we can get a fair estimation of how each model with standardized data might perform on unseen data.



```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning: Setting a random_state has no effect since shuffle is False.
  FutureWarning
ScaledLR: 0.990286 (0.004029)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning: Setting a random_state has no effect since shuffle is False.
  FutureWarning
ScaledLDA: 0.990919 (0.003542)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning: Setting a random_state has no effect since shuffle is False.
  FutureWarning
ScaledKNN: 0.990709 (0.003559)
ScaledNB: 0.532305 (0.025500)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning: Setting a random_state has no effect since shuffle is False.
  FutureWarning
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning: Setting a random_state has no effect since shuffle is False.
  FutureWarning
ScaledCART: 0.979097 (0.005776)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning: Setting a random_state has no effect since shuffle is False.
  FutureWarning
ScaledSVM: 0.991131 (0.003635)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning: Setting a random_state has no effect since shuffle is False.
  FutureWarning
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:470: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

III. Deep learning modeling (fully connected neural network)



A. Setting up the Architecture

The first thing we have to do is to set up the architecture.

B. Filling in the best numbers

Now that we've got our architecture specified, we need to find the best numbers for it. Before we start our training, we have
to configure the model by:

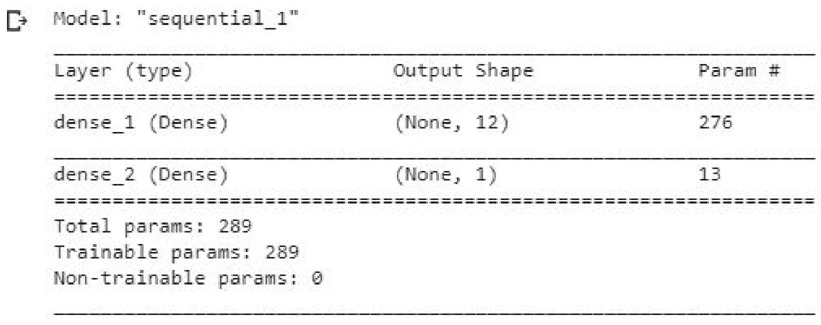
- Telling it which algorithm we want to use to do the optimization
- Telling it what loss function to use
- Telling it what other metrics we want to track apart from the loss function

We will use stochastic gradient descent optimizer for the compiler & the loss function for outputs that take the values 1 or 0
is called binary cross entropy.

Before training a model, we need to configure the learning process, which is done via the compile method. It receives three arguments:

- An optimizer. This could be the string identifier of an existing optimizer (such as sgf, rmsprop or adagrad).
- A loss function. This is the objective that the model will try to minimize. It can be the string identifier of an existing loss function (such as categorical_crossentropy or mse).
- A list of metrics. For any classification problem we will want to set this to metrics=['accuracy']. A metric could be the string identifier of an existing metric or a custom metric function.

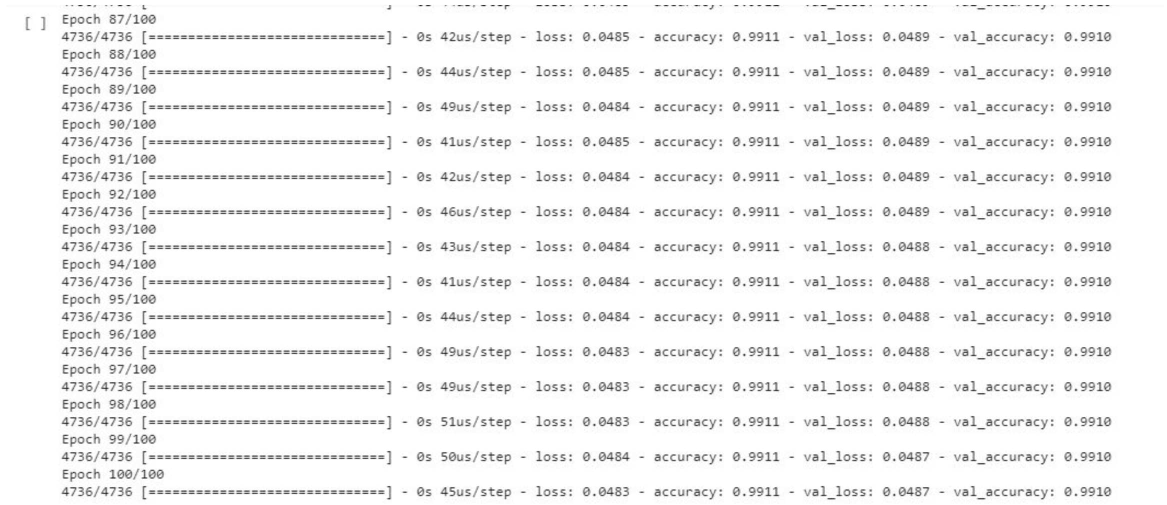
we're done with specifying our architecture! To see a summary of the full architecture:



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 12)                276       
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 13        
=================================================================
Total params: 289
Trainable params: 289
Non-trainable params: 0
_________________________________________________________________
```

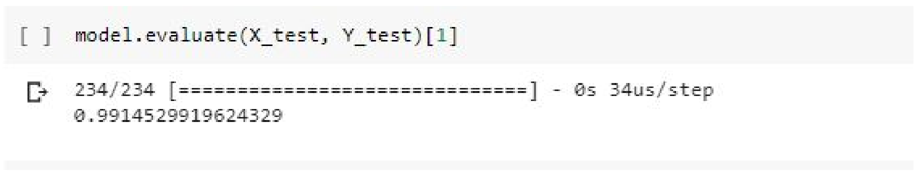
C. Training the model

We can now see that the model is training! By looking at the numbers, we should be able to see the loss decrease and the accuracy increase over time.

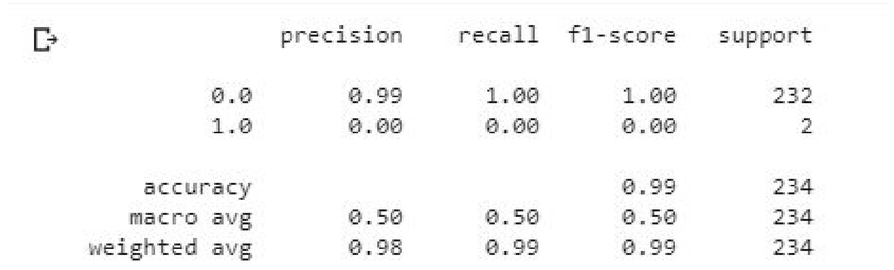


```
Epoch 87/100
4736/4736 [==============================] - 0s 42us/step - loss: 0.0485 - accuracy: 0.9911 - val_loss: 0.0489 - val_accuracy: 0.9910
Epoch 88/100
4736/4736 [==============================] - 0s 44us/step - loss: 0.0485 - accuracy: 0.9911 - val_loss: 0.0489 - val_accuracy: 0.9910
Epoch 89/100
4736/4736 [==============================] - 0s 49us/step - loss: 0.0484 - accuracy: 0.9911 - val_loss: 0.0489 - val_accuracy: 0.9910
Epoch 90/100
4736/4736 [==============================] - 0s 41us/step - loss: 0.0485 - accuracy: 0.9911 - val_loss: 0.0489 - val_accuracy: 0.9910
Epoch 91/100
4736/4736 [==============================] - 0s 42us/step - loss: 0.0484 - accuracy: 0.9911 - val_loss: 0.0489 - val_accuracy: 0.9910
Epoch 92/100
4736/4736 [==============================] - 0s 46us/step - loss: 0.0484 - accuracy: 0.9911 - val_loss: 0.0489 - val_accuracy: 0.9910
Epoch 93/100
4736/4736 [==============================] - 0s 43us/step - loss: 0.0484 - accuracy: 0.9911 - val_loss: 0.0488 - val_accuracy: 0.9910
Epoch 94/100
4736/4736 [==============================] - 0s 41us/step - loss: 0.0484 - accuracy: 0.9911 - val_loss: 0.0488 - val_accuracy: 0.9910
Epoch 95/100
4736/4736 [==============================] - 0s 44us/step - loss: 0.0484 - accuracy: 0.9911 - val_loss: 0.0488 - val_accuracy: 0.9910
Epoch 96/100
4736/4736 [==============================] - 0s 49us/step - loss: 0.0483 - accuracy: 0.9911 - val_loss: 0.0488 - val_accuracy: 0.9910
Epoch 97/100
4736/4736 [==============================] - 0s 49us/step - loss: 0.0483 - accuracy: 0.9911 - val_loss: 0.0488 - val_accuracy: 0.9910
Epoch 98/100
4736/4736 [==============================] - 0s 51us/step - loss: 0.0483 - accuracy: 0.9911 - val_loss: 0.0488 - val_accuracy: 0.9910
Epoch 99/100
4736/4736 [==============================] - 0s 50us/step - loss: 0.0484 - accuracy: 0.9911 - val_loss: 0.0487 - val_accuracy: 0.9910
Epoch 100/100
4736/4736 [==============================] - 0s 45us/step - loss: 0.0483 - accuracy: 0.9911 - val_loss: 0.0487 - val_accuracy: 0.9910
```

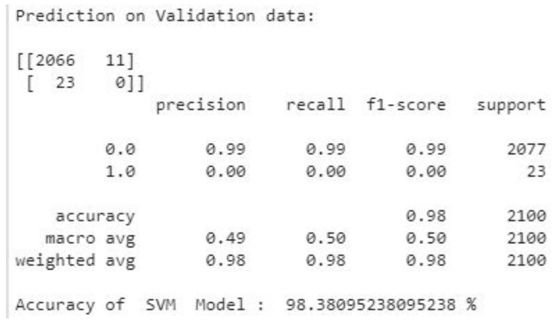
Evaluating our data on the test set:



```
[ ]  model.evaluate(X_test, Y_test)[1]

234/234 [==============================] - 0s 34us/step
0.9914529919624329
```



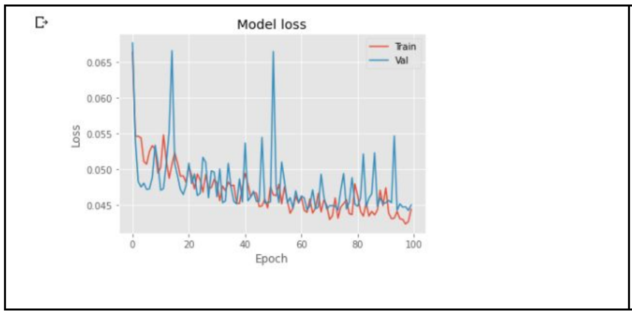
```
              precision    recall  f1-score   support

         0.0       0.99      1.00      1.00       232
         1.0       0.00      0.00      0.00         2

    accuracy                           0.99       234
   macro avg       0.50      0.50      0.50       234
weighted avg       0.98      0.99      0.99       234
```

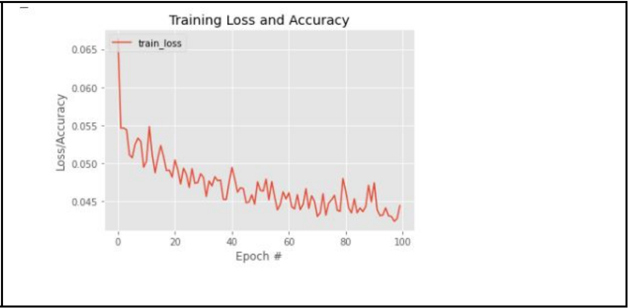


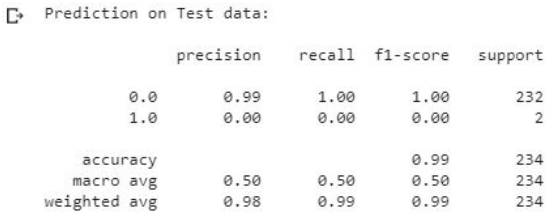
```
Prediction on Validation data:

[[2066   11]
 [  23    0]]
              precision    recall  f1-score   support

         0.0       0.99      0.99      0.99      2077
         1.0       0.00      0.00      0.00        23

    accuracy                           0.98      2100
   macro avg       0.49      0.50      0.50      2100
weighted avg       0.98      0.98      0.98      2100

Accuracy of  SVM  Model :  98.38095238095238 %
```

D. Visualizing Loss and Accuracy

How do we know if our model is currently over fitting?







```
Prediction on Test data:

              precision    recall  f1-score   support

         0.0       0.99      1.00      1.00       232
         1.0       0.00      0.00      0.00         2

    accuracy                           0.99       234
   macro avg       0.50      0.50      0.50       234
weighted avg       0.98      0.99      0.99       234
```

REFERENCES:

- Li Zhang, Qiao-ying LI, Yun-you Duan, Guo-zhen Yan, Yi-lin Yang and Rui-jing Yang. Artificial neural network aided non-invasive, BMC Medical Informatics and Decision Making. 2012; 12:55.
- IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM). 2012; 771-778
- Piatetsky-Shapiro, G. (1996). Advances in knowledge discovery and data mining (Vol. 21). U. M. Fayyad, P. Smyth, & R. Uthurusamy (Eds.). Menlo Park: AAAI press.
- 11] Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., ... & Zhou, Z. H. (2008). Top 10 algorithms in data mining. Knowledge and information systems, 14(1), 1-37.
- zgomi, H., Ghasemi Mahsayeh, M., Mohammadi, M., & Moradi, M. (2014).A Method for Finding Similar Documents Relying on Adding Repetition of Symbols in Length Based Filtering. Indian Journal of Scientific Research, 2(1), 81-84
- Hand, D. J. (2007). Principles of data mining. Drug safety, 30(7), 621-622