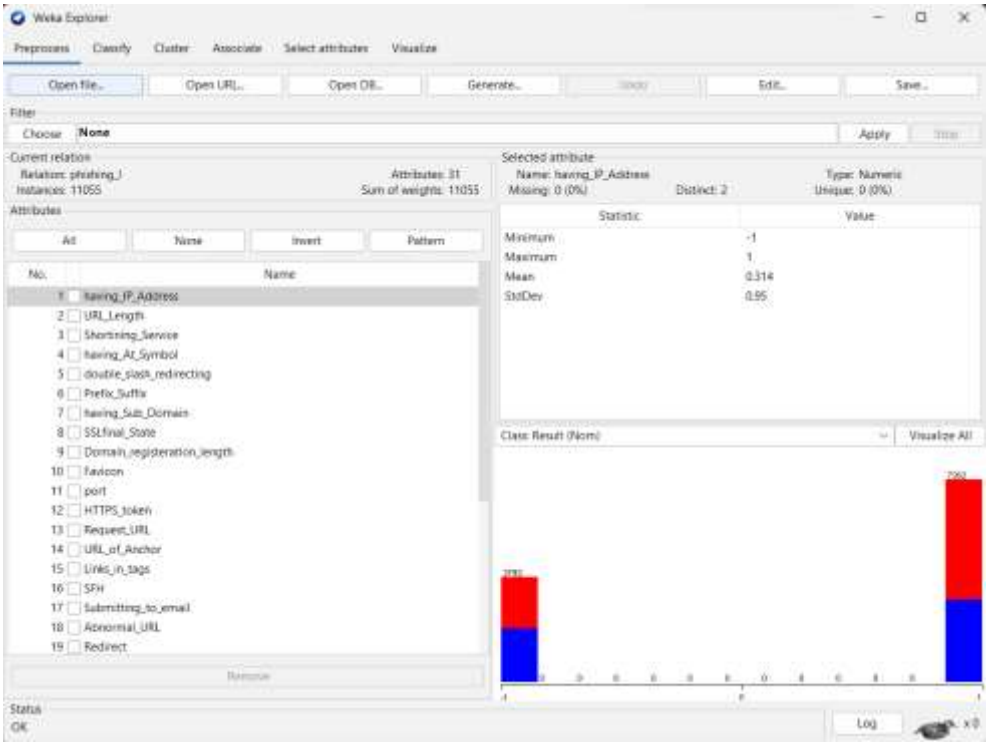
EXERCISE 1:

Imported csv data



Under classify selected, test option as cross validation 10 folds

1) Random Forest (Trees)
    a. Log

```
12:28:19: Weka Explorer
12:28:19: (c) 1999-2022 The University of Waikato, Hamilton, New Zealand
12:28:19: web: https://www.cs.waikato.ac.nz/~ml/weka/
12:28:19: Started on Wednesday, 23 February 2022
12:28:48: Base relation is now phishing_l (11055 instances)
12:31:22: Started weka.classifiers.trees.RandomForest
12:31:22: Command: weka.classifiers.trees.RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M
    1.0 -V 0.001 -S 1
12:31:43: Finished weka.classifiers.trees.RandomForest
```

    b. Output

```
=== Run information ===

Scheme:       weka.classifiers.trees.RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V
    0.001 -S 1
Relation:     phishing_l
Instances:    11055

Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===
```

```
RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 2.21 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       10751               97.2501 %
Incorrectly Classified Instances       304                2.7499 %
Kappa statistic                          0.9442
Mean absolute error                      0.0494
Root mean squared error                  0.1432
Relative absolute error                 10.0172 %
Root relative squared error             28.8286 %
Total Number of Instances            11055

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.961    0.019    0.976      0.961    0.969      0.944    0.996     0.995     Legitimate
                 0.981    0.039    0.970      0.981    0.975      0.944    0.996     0.996     Phishing
Weighted Avg.    0.973    0.030    0.973      0.973    0.972      0.944    0.996     0.995

=== Confusion Matrix ===

    a    b   <-- classified as
 4708  190 |    a = Legitimate
  114 6043 |    b = Phishing
```

2) LMT (Trees)

a. Log

```
12:38:04: Started weka.classifiers.trees.LMT
12:38:04: Command: weka.classifiers.trees.LMT -I -1 -M 15 -W 0.0
12:39:13: Finished weka.classifiers.trees.LMT
```

b. Output

```
=== Run information ===

Scheme:       weka.classifiers.trees.LMT -I -1 -M 15 -W 0.0
Relation:     phishing_l
Instances:    11055
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===


Time taken to build model: 6.89 seconds

=== Stratified cross-validation ===
=== Summary ===

```

```
Correctly Classified Instances       10715               96.9245 %
Incorrectly Classified Instances       340                3.0755 %
Kappa statistic                          0.9376
Mean absolute error                      0.0348
Root mean squared error                  0.1603
Relative absolute error                  7.0539 %
Root relative squared error             32.2655 %
Total Number of Instances            11055

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.958    0.022    0.972      0.958    0.965      0.938    0.990     0.986     Legitimate
                 0.978    0.042    0.967      0.978    0.973      0.938    0.990     0.990     Phishing
Weighted Avg.    0.969    0.033    0.969      0.969    0.969      0.938    0.990     0.988

=== Confusion Matrix ===

    a    b   <-- classified as
 4693  205 |    a = Legitimate
  135 6022 |    b = Phishing
```

3) Logistic (Function)

a. Log

```
12:39:26: Started weka.classifiers.functions.Logistic
12:39:26: Command: weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4
12:39:28: Finished weka.classifiers.functions.Logistic
```

b. Output

```
=== Run information ===

Scheme:       weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4
Relation:     phishing_l
Instances:    11055
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

Logistic Regression with ridge parameter of 1.0E-8
Coefficients...

Time taken to build model: 0.3 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       10252               92.7363 %
Incorrectly Classified Instances       803                7.2637 %
Kappa statistic                          0.8524
Mean absolute error                      0.1078
Root mean squared error                  0.2332
```

```
Relative absolute error                 21.8369 %
Root relative squared error             46.9471 %
Total Number of Instances           11055

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.905    0.055    0.929      0.905    0.917      0.853    0.979     0.977     Legitimate
                 0.945    0.095    0.926      0.945    0.935      0.853    0.979     0.982     Phishing
Weighted Avg.    0.927    0.077    0.927      0.927    0.927      0.853    0.979     0.980

=== Confusion Matrix ===

    a    b   <-- classified as
 4434  464 |    a = Legitimate
  339 5818 |    b = Phishing
```

4) SGD(Function)

a. Log

```
12:39:44: Started weka.classifiers.functions.SGD
12:39:44: Command: weka.classifiers.functions.SGD -F 0 -L 0.01 -R 1.0E-4 -E 500 -C 0.001 -S 1
12:39:48: Finished weka.classifiers.functions.SGD
```

b. Output

```
=== Run information ===

Scheme:       weka.classifiers.functions.SGD -F 0 -L 0.01 -R 1.0E-4 -E 500 -C 0.001 -S 1
Relation:     phishing_l
Instances:    11055
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

Loss function: Hinge loss (SVM)

Result =
Time taken to build model: 0.49 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       10276               92.9534 %
Incorrectly Classified Instances       779                7.0466 %
Kappa statistic                          0.8568
Mean absolute error                      0.0705
Root mean squared error                  0.2655
Relative absolute error                 14.2783 %
Root relative squared error             53.4385 %
Total Number of Instances            11055
```

```
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.905    0.051    0.934      0.905    0.919      0.857    0.927     0.887     Legitimate
                 0.949    0.095    0.926      0.949    0.938      0.857    0.927     0.907     Phishing
Weighted Avg.    0.930    0.075    0.930      0.930    0.929      0.857    0.927     0.899

=== Confusion Matrix ===

    a    b   <-- classified as
 4433  465 |    a = Legitimate
  314 5843 |    b = Phishing
```

5) Naïve Bayes (Bayes)

a. Log

```
12:39:57: Started weka.classifiers.bayes.NaiveBayes
12:39:57: Command: weka.classifiers.bayes.NaiveBayes
12:39:57: Finished weka.classifiers.bayes.NaiveBayes
```

b. Output

```
=== Run information ===

Scheme:       weka.classifiers.bayes.NaiveBayes
Relation:     phishing_l
Instances:    11055
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===


Time taken to build model: 0.06 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       10027               90.701  %
Incorrectly Classified Instances      1028                9.299  %
Kappa statistic                          0.8111
Mean absolute error                      0.1197
Root mean squared error                  0.274
Relative absolute error                 24.2585 %
Root relative squared error             55.1609 %
Total Number of Instances            11055

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.884    0.074    0.904      0.884    0.894      0.811    0.962     0.956     Legitimate
```

```
                 0.926    0.116    0.909      0.926    0.917      0.811    0.962     0.968
Phishing
Weighted Avg.    0.907    0.098    0.907      0.907    0.907      0.811    0.962     0.963

=== Confusion Matrix ===

     a    b   <-- classified as
  4328  570 |    a = Legitimate
   458 5699 |    b = Phishing
```

EXERCISE 2:

Changes done in "**phishing_sklearn.py",** added f1 score, recall, precision code

```
from warnings import simplefilter

import numpy as np
import pandas as pd
import sklearn
from numpy import genfromtxt
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (accuracy_score, confusion_matrix, f1_score,
                             precision_score, recall_score)
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler

simplefilter(action='ignore', category=FutureWarning)

#################################################################

feature=genfromtxt('phishing.csv',delimiter=',',usecols=(i for i in
range(0,30)),skip_header=1)
target=genfromtxt('phishing.csv',delimiter=',',usecols=(-1),skip_header=1)
sc = StandardScaler()
sc.fit(feature)
target_label = LabelEncoder().fit_transform(target)
feature_std = sc.transform(feature)
test_size_val=0.33
x_train, x_test, y_train, y_test = train_test_split(feature_std, target_label,
test_size=test_size_val, random_state=1)

print("Begin with test_size"+str(test_size_val)+":_____________________________________")
#####################################################
## print stats
precision_scores_list = []
accuracy_scores_list = []

def print_stats_metrics(y_test, y_pred):
    print('Accuracy: %.2f' % accuracy_score(y_test,   y_pred) )
    accuracy_scores_list.append(accuracy_score(y_test,   y_pred) )
    confmat = confusion_matrix(y_true=y_test, y_pred=y_pred)

    print('F1 score: %.2f' % f1_score(y_true=y_test,y_pred=y_pred))
    print('Precision: %.2f' % precision_score(y_true=y_test,y_pred=y_pred))
    print('Recall: %.2f' % recall_score(y_true=y_test,y_pred=y_pred))

    print ("confusion matrix")
    print(confmat)
```

```
    print (pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=True))

########################Logistic Regression##############################
print("########################Logistic Regression##############################")
clfLog = LogisticRegression()
clfLog.fit(x_train,y_train)
predictions = clfLog.predict(x_test)
print_stats_metrics(y_test, predictions)

########################Random Forest##############################
print("########################Random Forest##############################")
clfRandForest = RandomForestClassifier()
clfRandForest.fit(x_train,y_train)
predictions = clfRandForest.predict(x_test)
print_stats_metrics(y_test, predictions)
########################Decision Tree######################
print("########################Decision Tree######################")
clfDT = DecisionTreeRegressor()
clfDT.fit(x_train,y_train)
predictions = clfDT.predict(x_test)
print_stats_metrics(y_test, predictions)
########################Naive Bayes######################
print("########################Naive Bayes######################")
clfNB = GaussianNB()
clfNB.fit(x_train,y_train)
predictions = clfNB.predict(x_test)
print_stats_metrics(y_test, predictions)
```

At test_size=0.33

```
########################Logistic Regression##############################
Accuracy: 0.92
F1 score: 0.93
Precision: 0.92
Recall: 0.94
confusion matrix
[[1476  156]
 [ 121 1896]]
Predicted     0     1   All
True
0          1476   156  1632
1           121  1896  2017
All        1597  2052  3649
########################Random Forest##############################
Accuracy: 0.97
F1 score: 0.97
Precision: 0.96
Recall: 0.98
confusion matrix
[[1557   75]
 [  42 1975]]
Predicted     0     1   All
True
0          1557    75  1632
1            42  1975  2017
All        1599  2050  3649
########################Decision Tree######################
Accuracy: 0.96
F1 score: 0.96
Precision: 0.96
Recall: 0.96
confusion matrix
```

```
[[1549   83]
 [  76 1941]]
Predicted   0.0   1.0   All
True
0          1549    83  1632
1            76  1941  2017
All        1625  2024  3649
######################Naive Bayes#####################
Accuracy: 0.62
F1 score: 0.48
Precision: 1.00
Recall: 0.31
confusion matrix
[[1629    3]
 [1383  634]]
Predicted     0    1   All
True
0          1629    3  1632
1          1383  634  2017
All        3012  637  3649
```

At test_size=0.50

```
Begin with test_size 0.50:_____________________________
######################Logistic Regression#############################
Accuracy: 0.92
F1 score: 0.93
Precision: 0.92
Recall: 0.94
confusion matrix
[[2216  248]
 [ 176 2888]]
Predicted     0     1   All
True
0          2216   248  2464
1           176  2888  3064
All        2392  3136  5528
######################Random Forest#############################
Accuracy: 0.97
F1 score: 0.97
Precision: 0.96
Recall: 0.98
confusion matrix
[[2331  133]
 [  59 3005]]
Predicted     0     1   All
True
0          2331   133  2464
1            59  3005  3064
All        2390  3138  5528
######################Decision Tree#####################
Accuracy: 0.95
F1 score: 0.95
Precision: 0.95
Recall: 0.96
confusion matrix
[[2308  156]
 [ 125 2939]]
Predicted   0.0   1.0   All
True
0          2308   156  2464
```

```
1           125  2939  3064
All        2433  3095  5528
########################Naive Bayes########################
Accuracy: 0.61
F1 score: 0.47
Precision: 0.99
Recall: 0.30
confusion matrix
[[2459    5]
 [2134  930]]
Predicted     0    1   All
True
0          2459    5  2464
1          2134  930  3064
All        4593  935  5528
```

At test_size=0.20

```
Begin with test_size0.20:_____________________________
########################Logistic Regression##############################
Accuracy: 0.92
F1 score: 0.93
Precision: 0.92
Recall: 0.94
confusion matrix
[[ 895  101]
 [  68 1147]]
Predicted    0     1   All
True
0          895   101   996
1           68  1147  1215
All        963  1248  2211
########################Random Forest##############################
Accuracy: 0.98
F1 score: 0.98
Precision: 0.97
Recall: 0.98
confusion matrix
[[ 963   33]
 [  22 1193]]
Predicted    0     1   All
True
0          963    33   996
1           22  1193  1215
All        985  1226  2211
########################Decision Tree########################
Accuracy: 0.96
F1 score: 0.96
Precision: 0.97
Recall: 0.96
confusion matrix
[[ 955   41]
 [  46 1169]]
Predicted   0.0   1.0   All
True
0           955    41   996
1            46  1169  1215
All        1001  1210  2211
########################Naive Bayes########################
Accuracy: 0.62
F1 score: 0.47
Precision: 0.99
```

```
Recall: 0.31
confusion matrix
[[993   3]
 [843 372]]
Predicted     0    1   All
True
0           993    3   996
1           843  372  1215
All        1836  375  2211
```

EXERCISE 3:

Learning rate: Itis a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function.

Epoch: It's a term used in machine learning and indicates the number of passes of the entire training dataset the algorithm has been completed.

Batch Size: It's a term used in machine learning and refers to the number of training examples utilized in one iteration.

Coded F1 scores, precision, recall

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import numpy as np
import pandas as pd
from numpy import genfromtxt
from sklearn import datasets
from sklearn.model_selection import train_test_split
import sklearn
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score, f1_score
import pandas as pd
import matplotlib.pyplot as plt

################################################################

learning_rate = 0.01
#n_epochs = 5000
n_epochs = 100
batch_size = 10000

def convertOneHot(data):
    y_onehot=[0]*len(data)
    for i,j in enumerate(data):
        y_onehot[i]=[0]*(data.max()+1)
        y_onehot[i][j]=1
    return y_onehot

################################################################
feature=genfromtxt('phishing.csv',delimiter=',',usecols=(i for i in
range(0,31)),skip_header=1)
```

```
target=genfromtxt('phishing.csv',delimiter=',',usecols=(-1),skip_header=1)
sc = StandardScaler()
sc.fit(feature)
target_label = LabelEncoder().fit_transform(target)
target_onehot = convertOneHot(target_label)
feature_std = sc.transform(feature)
x_train, x_test, y_train_onehot, y_test_onehot = train_test_split(feature_std,
target_onehot, test_size=0.30, random_state=0)
A=x_train.shape[1]
B=len(y_train_onehot[0])
print(len(y_test_onehot[0]))
print(B)
print("Begin:_____")
######################################################################

def plot_metric_per_epoch():
    x_epochs = []
    y_epochs = []
    for i, val in enumerate(accuracy_scores_list):
        x_epochs.append(i)
        y_epochs.append(val)

    plt.scatter(x_epochs, y_epochs,s=50,c='lightgreen', marker='s', label='score')
    plt.xlabel('epoch')
    plt.ylabel('score')
    plt.title('Score per epoch')
    plt.legend()
    plt.grid()
    plt.show()

################################################
## print stats
precision_scores_list = []
accuracy_scores_list = []

def print_stats_metrics(y_test, y_pred):
    print('Accuracy: %.2f' % accuracy_score(y_test,   y_pred) )
    #Accuracy: 0.84
    accuracy_scores_list.append(accuracy_score(y_test,   y_pred) )
    confmat = confusion_matrix(y_true=y_test, y_pred=y_pred)

    print('F1 score: %.2f' % f1_score(y_true=y_test,y_pred=y_pred))
    print('Precision: %.2f' % precision_score(y_true=y_test,y_pred=y_pred))
    print('Recall: %.2f' % recall_score(y_true=y_test,y_pred=y_pred))

    print ("confusion matrix")
    print(confmat)
    print (pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'],
margins=True))

#############################################################
def layer(input, weight_shape, bias_shape):
    weight_stddev = (2.0/weight_shape[0])**0.5
    w_init = tf.random_normal_initializer(stddev=weight_stddev)
    bias_init = tf.constant_initializer(value=0)
    W = tf.get_variable("W", weight_shape, initializer=w_init)
    b = tf.get_variable("b", bias_shape, initializer=bias_init)
    return tf.nn.relu(tf.matmul(input, W) + b)
#############################################################
def inference_deep_layers(x_tf, A, B):
    with tf.variable_scope("hidden_1"):
        hidden_1 = layer(x_tf, [A, 15],[15])
    with tf.variable_scope("hidden_2"):
        hidden_2 = layer(hidden_1, [15, 5],[5])
    with tf.variable_scope("output"):
```

```
        output = layer(hidden_2, [5, B], [B])
    return output
################################################################
def loss_deep(output, y_tf):
    xentropy = tf.nn.softmax_cross_entropy_with_logits(logits=output, labels=y_tf)
    loss = tf.reduce_mean(xentropy)
    return loss
#############################################################

def training(cost):
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    train_op = optimizer.minimize(cost)
    return train_op

#############################################################
def evaluate(output, y_tf):
    correct_prediction = tf.equal(tf.argmax(output,1), tf.argmax(y_tf,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
    return accuracy
################################################################

x_tf = tf.placeholder("float",[None,A])
y_tf = tf.placeholder("float",[None,B])
################################################################
output = inference_deep_layers(x_tf,A,B)
cost = loss_deep(output,y_tf)
train_op=training(cost)
eval_op=evaluate(output,y_tf)
################################################################
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
################################################################
y_p_metrics = tf.argmax(output,1)
################################################################
num_samples_train_set=x_train.shape[0]
num_batches = int(num_samples_train_set/batch_size)

################################################################

for i in range(n_epochs):
    print("epoch %s out of %s"%(i,n_epochs))
    for batch_n in range(num_batches):
        sta = batch_n*batch_size
        end = sta+batch_size
        sess.run(train_op,feed_dict={x_tf:x_train[sta:end],y_tf:y_train_onehot[sta:end]})
    print ("-------------------------------------------------------------------------------
-")
    print ("Accuracy score")
    #result = sess.run(eval_op,feed_dict={x_tf:x_test,y_tf:y_test_onehot})
    result, y_result_metrics = sess.run([eval_op, y_p_metrics], feed_dict={x_tf: x_test,
 y_tf: y_test_onehot})
    print("Run {},{}".format(i,result))
    y_true = np.argmax(y_test_onehot,1)
    print_stats_metrics(y_true, y_result_metrics)
    if i==n_epochs-1:
        plot_metric_per_epoch()
```

At different epoch, batch size and learning rate

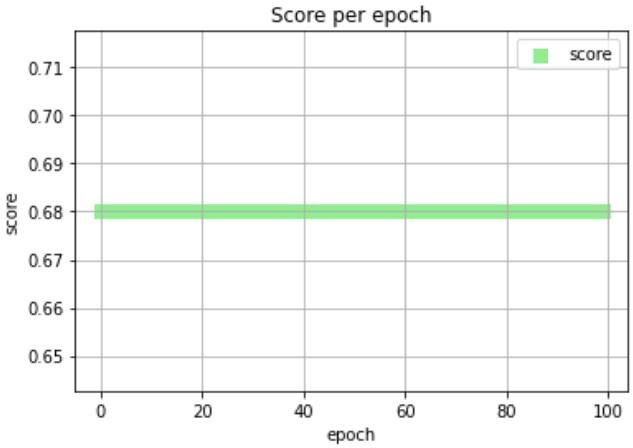
1) **At**

```
1.learning_rate = 0.01
```

```
2.n_epochs = 100
3.batch_size = 10000
```

Output:

```
1.  Accuracy score
2.  Run 99,0.4727163016796112
3.  Accuracy: 0.47
4.  F1 score: 0.09
5.  Precision: 0.85
6.  Recall: 0.05
7.  confusion matrix
8.  [[1483   15]
9.   [1734   85]]
10. Predicted     0    1   All
11. True
12. 0          1483   15  1498
13. 1          1734   85  1819
14. All        3217  100  3317
15.
16.
```



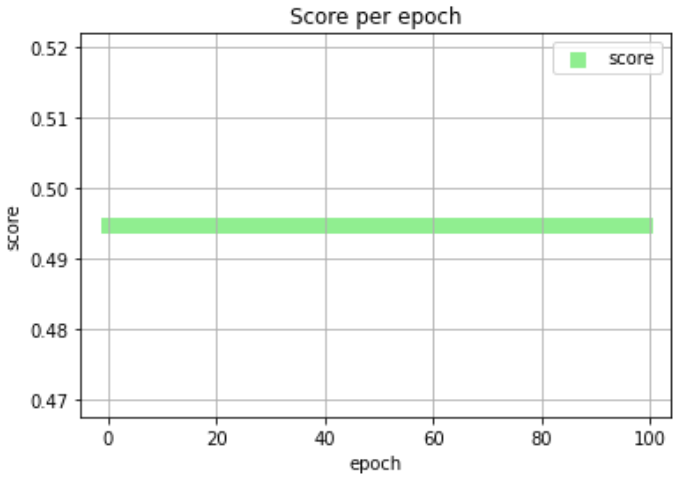
2) At

```
1.  learning_rate = 0.006
2.  n_epochs = 100
3.  batch_size = 10000
```

Output:

```
1.  Accuracy: 0.41
2.  F1 score: 0.20
3.  Precision: 0.39
```

```
Recall: 0.13
confusion matrix
[[1127  371]
 [1579  240]]
Predicted     0    1   All
True
0          1127  371  1498
1          1579  240  1819
All        2706  611  3317
```

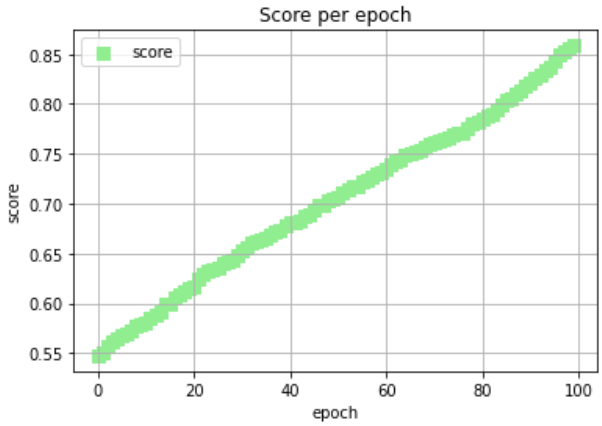


3) At

```
learning_rate = 0.006
n_epochs = 100
batch_size = 1000
```

Output:

```
Accuracy: 0.95
F1 score: 0.95
Precision: 0.94
Recall: 0.96
confusion matrix
[[1390  108]
 [  65 1754]]
Predicted     0     1   All
True
0          1390   108  1498
1            65  1754  1819
All        1455  1862  3317
```

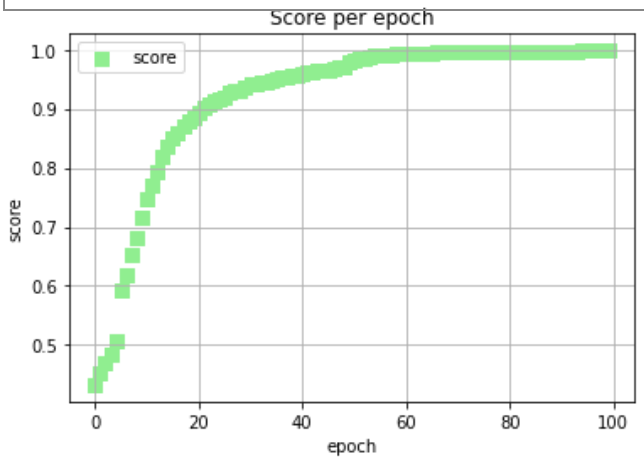


4) At

```
learning_rate = 0.006
n_epochs = 100
batch_size = 128
```

Output:

```
Accuracy: 0.99
F1 score: 0.99
Precision: 0.99
Recall: 1.00
confusion matrix
[[1474   24]
 [   0 1819]]
Predicted     0     1   All
True
0          1474    24  1498
1             0  1819  1819
All        1474  1843  3317
```



Results:

At first we saw that how our dataset looks like, Then we installed weka and opened the CSV data set. Using the Explorer window and. In the classifier tab, we set the classification as different sets. Logistic regression, Naïve Bayes, Random Forest, etc.

Will logged the data by each of them, The output contains the time taken by the model. The confusion matrix the F1 score, precision, accuracy and recall, for the data set over the trained model.

Time taken by model:

- Random Forest: 2.21 s
- LMT: 6.89 s
- Logistic: 0.3
- SGD: 0.49
- Naïve bayes: 0.06

Mean Abs Error:

- Random Forest: 0.049
- LMT: 0.034
- Logistic: 0.1078
- SGD: 0.0705
- Naïve bayes: 0.1197

Precision:

- Random Forest: 0.970
- LMT: 0.967
- Logistic: 0.926
- SGD: 0.930
- Naïve bayes: 0.909

On getting such results we can finally discuss which phising detector performed better in comparison to others, we can see performance on basis of various parameters that Random Forest performed better in comparison to others as it have high precision score and lower Mean Abs Error, but time taken is much higher compared to rest algorithms

In sklearn we implemented the classifier model on the basis of different algorithms like Logistic Regression, Decision Tree, Random Forest, and Naïve Bayes

On the basis of various parameters and accuracy we can see that which model performed better
Accuracy

- Logistic Regression:0.92
- Random Forest:0.97
- Decision Tree:0.96
- Naïve Bayes:0.62

F1 score

- Logistic Regression:0.93

Random Forest:0.97
Decision Tree:0.96
Naïve Bayes:0.48

Here also we can see the Random Forest worked better in comparison to other

Now, in terms of NN layer, we saw the highest accuracy came out to be 0.99 on slower learning rate and smaller batch size which was even better than earlier attempts.
So we can say that Neural Networks worked better in comparison to others.
We can also say that Neural Networks have the ability to learn and model non-linear and complex relationships , which is really important because in real-life, many of the relationships between inputs and outputs are non-linear as well as complex, so it performed better.