*Name:Vinay Basavaraja*

*Course :Data Mining- CS634*

FINAL PROJECT

Git-Hub: <https://github.com/vinayb004/Data_mining_Final>

**1. Option Selected:**

1st option is selected and the following algorithms are

likely to be implemented.

* Random Forest
* Svm
* lstm

**2. Software and libraries used:**

* Jupyter Notebook
* Python
* Numpy
* Pandas
* SciKit Learn

**3.Hardware Configuration:**

* Operating System: Windows
* Processor: intel core i-7
* RAM: 16GB

**4.Screenshots**

I have attached all my output in pdf file

**5. Source Code**

#!/usr/bin/env python  
# coding: utf-8  
  
# In[147]:  
  
  
import numpy as np   
import pandas as pd   
import seaborn as sns  
sns.set(color\_codes=True)  
import matplotlib.pyplot as plt  
get\_ipython().run\_line\_magic('matplotlib', 'inline')  
import collections  
import math  
  
  
# In[148]:  
  
  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler   
from sklearn.ensemble import RandomForestClassifier  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.model\_selection import KFold  
from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay  
from sklearn.metrics import accuracy\_score as acs  
from sklearn.svm import SVC  
  
  
# In[172]:  
  
  
from prettytable import PrettyTable  
from keras.models import Sequential  
from keras.layers import Dense  
from keras.layers import LSTM  
from keras.layers.embeddings import Embedding  
from keras.preprocessing import sequence  
from keras.callbacks import EarlyStopping  
from keras.layers.core import Dense,Activation,Dropout  
import warnings  
warnings.filterwarnings("ignore")  
  
  
# # Pre-Processing of the data  
  
# ## Dataset Used (Breast Cancer Diagnostic)  
  
# In[157]:  
  
  
data=pd.read\_csv("data.csv")  
data.drop(["Unnamed: 32","id"],axis=1,inplace=True)  
display(data.head(20))  
  
  
# In[158]:  
  
  
data.shape  
  
  
# In[216]:  
  
  
X=data.iloc[:,1:32]  
y=data.iloc[:,0]  
  
  
# In[217]:  
  
  
scaler=StandardScaler()  
X=scaler.fit\_transform(X)  
  
  
# # Function To Calcuate Metrices   
  
# # ML Models  
  
# In[218]:  
  
  
def data\_met(tn, fp, fn, tp):  
 result = []  
 result.append(tn)  
 result.append(fp)  
 result.append(fn)  
 result.append(tp)  
 tpr = tp/(tp+fn)  
 result.append(tpr)  
 tnr = tn/(tn+fp)  
 result.append(tnr)  
 fpr = fp/(tn+fp)  
 result.append(fpr)  
 fnr = fn/(tp+fn)  
 result.append(fnr)  
 recall = tp/(tp+fn)  
 result.append(recall)  
 precision = tp/(tp+fp)  
 result.append(precision)  
 f1 = (2\*tp)/(2\*tp+fp+fn)  
 result.append(f1)  
 acc = (tp+tn)/(tp+fp+fn+tn)  
 result.append(acc)  
 err = (fp+fn)/(tp+fp+fn+tn)  
 result.append(err)  
 bacc = (tpr+tnr)/2  
 result.append(bacc)  
 tss = tp/(tp+fn) - fp/(fp+tn)  
 result.append(tss)  
 hss = 2\*(tp\*tn - fp\*fn)/((tp+fn)\*(fn+tn) + (tp+fp)\*(fp+tn))  
 result.append(hss)  
 return np.array(result)  
  
  
# In[233]:  
  
  
cross\_validation\_folds = KFold(n\_splits=10,shuffle=True, random\_state=3030)  
fold = 0  
for train\_index, test\_index in cross\_validation\_folds.split(X, y):  
 fold += 1  
 print("Fold", str(fold))  
 X\_train, X\_test = X[train\_index], X[test\_index]  
 y\_train, y\_test = y[train\_index], y[test\_index]  
 #Random Forrest  
 print("\tRandom Forest model result:")  
 rf = RandomForestClassifier(max\_depth=5, random\_state=0)  
 rf.fit(X\_train, y\_train)  
 y\_pred\_rf = rf.predict(X\_test)  
 tn, fp, fn, tp = confusion\_matrix(y\_test, y\_pred\_rf).ravel()  
 rf\_result = data\_met(tn, fp, fn, tp)  
 print("\t\tTrue negative:", rf\_result[0])  
 print("\t\tFalse positive:", rf\_result[1])  
 print("\t\tFalse negative:", rf\_result[2])  
 print("\t\tTrue positive:", rf\_result[3])  
 print("\t\tTrue positive rate:", rf\_result[4])  
 print("\t\tTrue negative rate:", rf\_result[5])  
 print("\t\tFalse positive rate:", rf\_result[6])  
 print("\t\tFalse negative rate:", rf\_result[7])  
 print("\t\tRecall:", rf\_result[8])  
 print("\t\tPrecision:", rf\_result[9])  
 print("\t\tF1:", rf\_result[10])  
 print("\t\tAccuracy:", rf\_result[11])  
 print("\t\tError Rate:", rf\_result[12])   
 print("\t\tBalance Accuracy:", rf\_result[13])  
 print("\t\tTrue skill statistics:", rf\_result[14])  
 print("\t\tHeidke skill score:", rf\_result[15])  
 #SVM  
 print("\tSVM model result:")  
 svc = SVC(gamma='auto')  
 svc.fit(X\_train, y\_train)  
 y\_pred\_svc = svc.predict(X\_test)  
 tn, fp, fn, tp = confusion\_matrix(y\_test, y\_pred\_svc).ravel()  
 svc\_result = data\_met(tn, fp, fn, tp)  
 print("\t\tTrue negative:", svc\_result[0])  
 print("\t\tFalse positive:", svc\_result[1])  
 print("\t\tFalse negative:", svc\_result[2])  
 print("\t\tTrue positive:", svc\_result[3])  
 print("\t\tTrue positive rate:", svc\_result[4])  
 print("\t\tTrue negative rate:", svc\_result[5])  
 print("\t\tFalse positive rate:", svc\_result[6])  
 print("\t\tFalse negative rate:", svc\_result[7])  
 print("\t\tRecall:", svc\_result[8])  
 print("\t\tPrecision:", svc\_result[9])  
 print("\t\tF1:", svc\_result[10])  
 print("\t\tAccuracy:", svc\_result[11])  
 print("\t\tError Rate:", svc\_result[12])   
 print("\t\tBalance Accuracy:", svc\_result[13])  
 print("\t\tTrue skill statistics:", svc\_result[14])  
 print("\t\tHeidke skill score:", svc\_result[15])  
 print("\n\n")  
 ##lstm  
 #print (X[:5])  
 #print (y[:5],set(y))  
 #print (y[:5],set(y))  
   
   
 X\_bar = X.copy()  
 y\_bar = np.array([1 if k == 'B' else 0 for k in list(y)])  
   
 X\_train\_fold = np.array(X\_bar[train\_index])  
 y\_train\_fold = np.array(y\_bar[train\_index]).reshape(-1,1).astype(int)  
 X\_val\_fold = np.array(X\_bar[test\_index])  
 y\_val\_fold = np.array(y\_bar[test\_index]).reshape(-1,1).astype(int)  
   
 X\_train\_fold = np.reshape(X\_train\_fold,(X\_train\_fold.shape[0], 1, X\_train\_fold.shape[1])).astype(int)  
 X\_val\_fold = np.reshape(X\_val\_fold, (X\_val\_fold.shape[0], 1, X\_val\_fold.shape[1])).astype(int)  
   
 model = Sequential()  
  
 model.add(LSTM(60, return\_sequences=True, input\_shape=(1,30)))  
 model.add(Dropout(0.2))  
 model.add(Dense(1))  
 model.add(Activation('Softmax'))  
 model.compile(loss='binary\_crossentropy', optimizer='Adam', metrics=['accuracy'])  
 history = model.fit(X\_train\_fold, y\_train\_fold, epochs=5, batch\_size=20, validation\_data=(X\_val\_fold, y\_val\_fold),  
 verbose=1, shuffle=False)  
   
 ## Report LSTM Result  
 y\_pred\_lstm = model.predict(X\_val\_fold)  
 y\_pred\_lstm = np.array([t[0][0] for t in y\_pred\_lstm]).ravel()  
 y\_val\_fold = np.array(y\_bar[test\_index]).ravel()  
 tn, fp, fn, tp = confusion\_matrix(y\_val\_fold, y\_pred\_lstm).ravel()  
 lstm\_result = data\_met(tn, fp, fn, tp)  
 print("LSTM:\n\n")  
 print("\t\tTrue negative:", lstm\_result[0])  
 print("\t\tFalse positive:", lstm\_result[1])  
 print("\t\tFalse negative:", lstm\_result[2])  
 print("\t\tTrue positive:", lstm\_result[3])  
 print("\t\tTrue positive rate:", lstm\_result[4])  
 print("\t\tTrue negative rate:", lstm\_result[5])  
 print("\t\tFalse positive rate:", lstm\_result[6])  
 print("\t\tFalse negative rate:", lstm\_result[7])  
 print("\t\tRecall:", lstm\_result[8])  
 print("\t\tPrecision:", lstm\_result[9])  
 print("\t\tF1:", lstm\_result[10])  
 print("\t\tAccuracy:", lstm\_result[11])  
 print("\t\tError Rate:", lstm\_result[12])   
 print("\t\tBalance Accuracy:", lstm\_result[13])  
 print("\t\tTrue skill statistics:", lstm\_result[14])  
 print("\t\tHeidke skill score:", lstm\_result[15])  
   
   
 table=PrettyTable()  
 table.field\_names = ['Model','TN','FP','FN','TP','TPR','TNR','FPR','FNR','recall','precision','F1','accuracy','ER','BA','TSS','HSS']  
 table.add\_row(['RandomForest',rf\_result[0],rf\_result[1],rf\_result[2],rf\_result[3],rf\_result[4],rf\_result[5],rf\_result[6],rf\_result[7],rf\_result[8],rf\_result[9],rf\_result[10],rf\_result[11],rf\_result[12],rf\_result[13],rf\_result[14],rf\_result[15]])  
 table.add\_row(['SVM',svc\_result[0],svc\_result[1],svc\_result[2],svc\_result[3],svc\_result[4],svc\_result[5],svc\_result[6],svc\_result[7],svc\_result[8],svc\_result[9],svc\_result[10],svc\_result[11],svc\_result[12],svc\_result[13],svc\_result[14],svc\_result[15]])  
 table.add\_row(['lstm',lstm\_result[0],lstm\_result[1],lstm\_result[2],lstm\_result[3],lstm\_result[4],lstm\_result[5],lstm\_result[6],lstm\_result[7],lstm\_result[8],lstm\_result[9],svc\_result[10],lstm\_result[11],lstm\_result[12],lstm\_result[13],lstm\_result[14],lstm\_result[15]])  
 print(table)  
 print("\n\n")