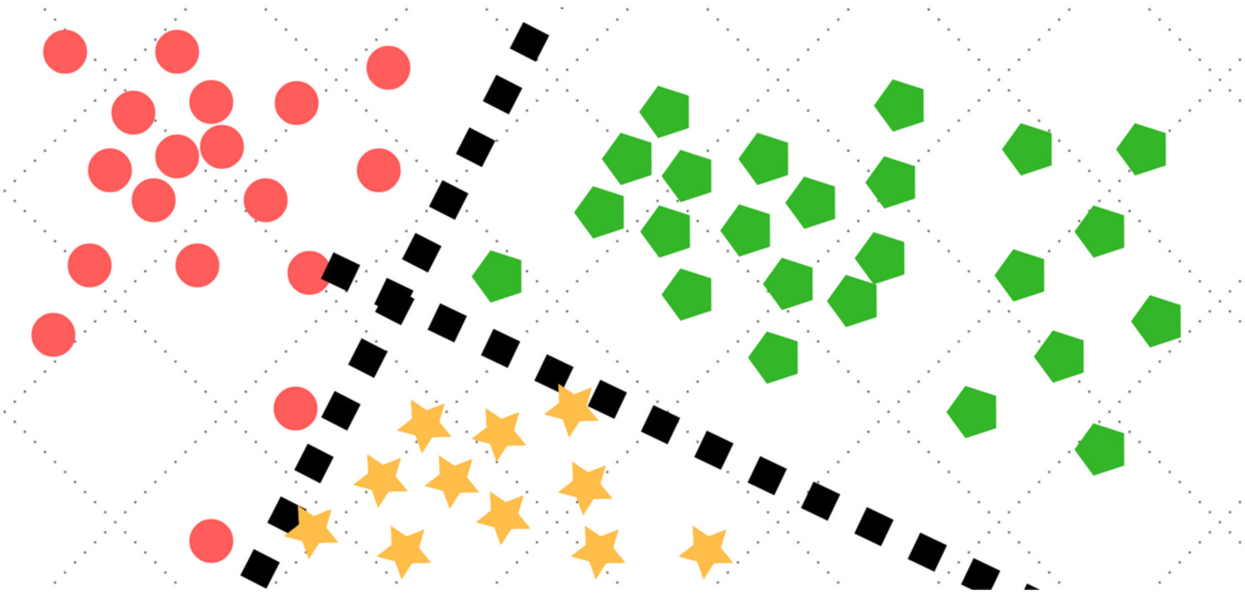
CS 634 -DATA MINING

SUPERVISED DATA MINING

CLASSIFICATION



GARIMA MATHUR

UCID - gm47

Email id - gm47@njit.edu

GitHub - github.com/Garima-Mathur

Introduction :

Supervised classification is the technique most often used for the quantitative analysis of remote sensing image data. At its core is the concept of segmenting the spectral domain into regions that can be associated with the ground cover classes of interest to a particular application. In practice those regions may sometimes overlap. A variety of algorithms are available for the task

Algorithms Implemented :

- ### LSTM (Long Short Term Memory) :

LSTM is a type of recurrent neural network but is better than traditional recurrent neural networks in terms of memory. Having a good hold over memorizing certain patterns LSTMs perform fairly better. As with every other NN, LSTM can have multiple hidden layers and as it passes through every layer, the relevant information is kept and all the irrelevant information gets discarded in every single cell.

- ### Random Forest :

The random forest is a classification algorithm consisting of many decision trees. It uses bagging and features randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

- ### Naive Bayes :

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

Requirements :

Software Configuration :

Google Colab

Jupyter Notebook 6.4.3 Anaconda
version Python 3.8
NumPy, pandas ,scikit learn,keras,tensor flow ,matplotlib

Hardware Configuration :

Operating System: Windows 10

Processor: Intel(R) Core (TM) i5-7200U CPU @ 2.50GHz 2.70 GHz RAM: 8GB

How to Run the Application :

Prerequisites :

- Python 3 and Jupyter Notebook 6.4.3 installed in the system.
- Alternatively we can use Google Colab

Dataset Used :

Diabetes Prediction Dataset

This dataset uses various parameters to determine if a person could be diabetic or not.

Dataset Preparation :

Dataset Preparation :

```
[80] df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
[81] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
[82] df.isnull().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
[83] X=df.iloc[:,0:7]
     Y=df.iloc[:,8]
```

```
[104] X.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288

```
[105] Y.head()
```

```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

Code Implementation :

Importing all required Libraries for the program .

Importing all Libraries :

```
[121] import numpy as np
      import pandas as pd
      import warnings
      warnings.filterwarnings('ignore')
      from sklearn import metrics
      from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
      from sklearn.model_selection import train_test_split
      import math
      import keras
      from keras.models import Sequential
      from tensorflow.keras import layers
      from keras.layers import Dense
      from keras.layers import LSTM
      from keras.layers import *
      from sklearn.metrics import roc_curve ,auc
      import matplotlib.pyplot as plt
```

Implementing the Calculate function to evaluate all Metrics :

```
def calculate(confusion_matrix):
    l1={}
    True_positive=confusion_matrix[0,0]
    False_Positive=confusion_matrix[0,1]
    False_Negative=confusion_matrix[1,0]
    True_Negative=confusion_matrix[1,1]
    TP=True_positive
    FP=False_Positive
    FN=False_Negative
    TN=True_Negative
    True_positive_rate= TP/(TP + FN)
    True_negative_rate= TN/(TN + FP)
    False_positive_rate= FP/(TN + FP)
    False_negative_rate= FN/(TP + FN)
    Recall= TP/(TP + FN)
    Precision= TP/(TP + FP)
    F1_measure= (2 * TP)/(2 * TP + FP + FN)
    Accuracy=(TP + TN)/(TP + FP + FN + TN)
    Error_rate= (FP + FN)/(TP + FP + FN + TN)
    Balanced_Accuracy= 1/2 *((TP/(TP+FN) + (TN/(TN+FP))))
    True_Skill_Statistics = ((TP/(TP+FN))-(FP/(FP+TN)))
    Heidke_Skill_Score=2*((TP * TN) - (FP * FN))/((TP + FN)*(FN+TN)+(TP+FP)*(FP+TN))
    l1 = {'False negative':FN, 'False positive': FP, 'True positive': TP, 'True negative': TN,
          'True positive rate': True_positive_rate, 'True negative rate': True_negative_rate, 'False positive rate': False_positive_rate, 'False negative rate': False_negative_rate,
          'Recall': Recall, 'Precision': Precision, 'F1 measure': F1_measure, 'Error rate': Error_rate,'Accuracy': Accuracy,'Balanced Accuracy':Balanced_Accuracy,'True Skill Statistics':True_Skill_Stat
          'Heidke Skill Score':Heidke_Skill_Score}
    result = pd.DataFrame([l1])
    return result
```

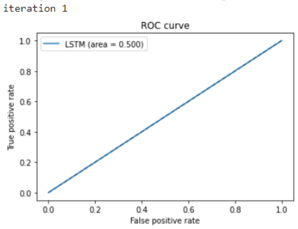
LSTM Code Implementation :

Implementing LSTM :

```
[129] for i in range(1,11):
        X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.2)
        model2 = Sequential()
        model2.add(LSTM(units = 64, return_sequences = True,input_shape= (X_train.shape[1],1)))
        model2.add(LSTM(units = 64, return_sequences = True))
        model2.add(Dense(units = 1,activation='softmax'))
        model2.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics=['accuracy'])
        model2.fit(X_train,y_train)
        prediction_rm=model2.predict(X_test)
        prediction_value=prediction_rm[:,0]
        print("iteration",i)
        y_pred_keras = prediction_value.ravel()
        fpr_keras, tpr_keras, thresholds_keras = roc_curve(y_test, y_pred_keras)
        auc_keras = auc(fpr_keras, tpr_keras)
        plt.plot([0, 1], [0, 1], 'k--')
        plt.plot(fpr_keras, tpr_keras, label='LSTM (area = {:.3f})'.format(auc_keras))
        plt.xlabel('False positive rate')
        plt.ylabel('True positive rate')
        plt.title('ROC curve')
        plt.legend(loc='best')
        plt.show()
        n_lstm=tf.math.confusion_matrix(y_test,prediction_value)
        confusion_matrix_lstm=n_lstm.numpy()
        final_value=calculate(confusion_matrix_lstm)
        display(final_value)
    print("Mean Accuracy of LSTM on data: ")
    model2.evaluate(X_test,y_test)
```

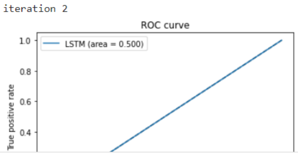
Output :

```
20/20 [==============================] - 5s 13ms/step - loss: 10.2076 - accuracy: 0.3306
iteration 1
```



	False negative	False positive	True positive	True negative	True positive rate	True negative rate	False positive rate	False negative rate	Recall	Precision	F1 measure	Error rate	Accuracy	Balanced Accuracy	True Skill Statistics	Heidke Skill Score
0	0	89	0	65	NaN	0.422078	0.577922	NaN	NaN	0.0	0.0	0.577922	0.422078	NaN	NaN	0.0

```
20/20 [==============================] - 4s 13ms/step - loss: 10.2324 - accuracy: 0.3290
iteration 2
```



Overall Accuracy achieved : 34.42%

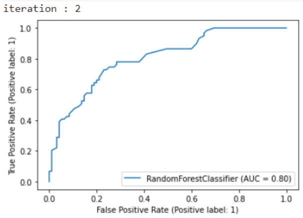
Random Forest Code Implementation :

Implementing Random Forest :

```
[115] from sklearn.ensemble import RandomForestClassifier
      total_score = 0.0
      for i in range(1,11):
          X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.2)
          model3 = RandomForestClassifier()
          model3.fit(X_train,y_train)
          prediction_rm=model3.predict(X_test)
          print("iteration :", i)
          metrics.plot_roc_curve(model3, X_test, y_test)
          plt.show()
          confusion_matrix_Random=confusion_matrix(y_test, prediction_rm)
          final_value=calculate(confusion_matrix_Random)
          display(final_value)
          total_score = total_score + accuracy_score(prediction_rm,y_test)
      print("Mean Accuracy of Random Forest on data: " , (total_score/10)*100)
```

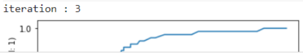
Output :

```
iteration : 2
```



	False negative	False positive	True positive	True negative	True positive rate	True negative rate	False positive rate	False negative rate	Recall	Precision	F1 measure	Error rate	Accuracy	Balanced Accuracy	True Skill Statistics	Heidke Skill Score
0	25	15	80	34	0.761905	0.693878	0.306122	0.238095	0.761905	0.842105	0.8	0.25974	0.74026	0.727891	0.455782	0.432258

```
iteration : 3
```



Overall Accuracy achieved : 75.19%

Naive Bayes Implementation :

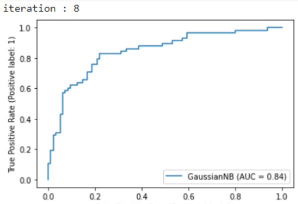
Implementing Naive Bayes :

```
[128] from sklearn.naive_bayes import GaussianNB
      total_score = 0.0
      for i in range(1,11):
          X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.2)
          model1 = GaussianNB()
          model1.fit(X_train,y_train)
          prediction_value=model1.predict(X_test)
          confusion_matrix_naive=confusion_matrix(y_test, prediction_value)
          print("iteration :", i)
          metrics.plot_roc_curve(model1, X_test, y_test)
          plt.show()
          plt.show()
          final_value=calculate(confusion_matrix_naive)
          display(final_value)
          total_score = total_score + accuracy_score(prediction_value,y_test)
      print(" Mean Accuracy of Naive Bayes :", (total_score/10)*100)
```

Output :

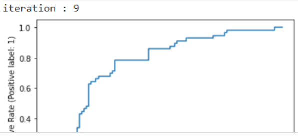
	negative	positive	positive	negative	rate	rate	rate	rate			measure	rate		Accuracy	Statistics	Score
0	28	13	76	37	0.730769	0.74	0.26	0.269231	0.730769	0.853933	0.787565	0.266234	0.733766	0.735385	0.470769	0.436753

iteration : 8



	False negative	False positive	True positive	True negative	True positive rate	True negative rate	False positive rate	False negative rate	Recall	Precision	F1 measure	Error rate	Accuracy	Balanced Accuracy	True Skill Statistics	Heidke Skill Score
0	20	14	82	38	0.803922	0.730769	0.269231	0.196078	0.803922	0.854167	0.828283	0.220779	0.779221	0.767345	0.534691	0.519985

iteration : 9



Overall Accuracy achieved : 75.58%