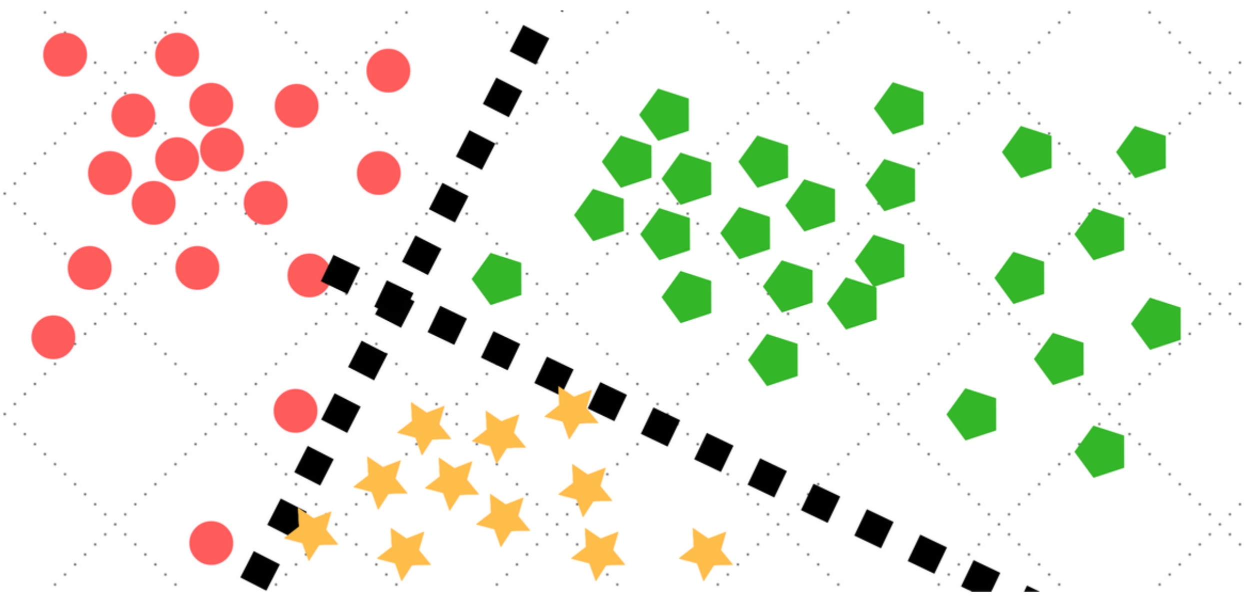
CS 634 -DATA MINING

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.

- **Decision Tree :**

Decision trees build classification or regression models in the form of a tree structure. 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

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 :

Make Moons Dataset :

The make moons Dataset is for classification and will generate a swirl pattern, or two moons.

We can control how noisy the moon shapes are and the number of samples to generate.

This test problem is suitable for algorithms that are capable of learning nonlinear class boundaries.

Dataset Preparation :

Dataset preparation is done using Gaussian Distribution and is separated based on the separable factor .

Gaussian distribution is the healthy-studied probability distribution. It is for nonstop-valued random variables.

Setup and Data Processing

```
[37] import numpy as np
     from sklearn.datasets import make_moons


     import matplotlib.pyplot as plt

     def data(n_samples = 100, seed = 123, separable = True):

         np.random.seed(seed)

         if separable == True: # separable
             # genrate points using Gaussian distributions
             p1 = np.random.multivariate_normal([0, 0], [[1, .95],[.95, 1]], n_samples)
             p2 = np.random.multivariate_normal([1, 4], [[1, .65],[.65, 1]], n_samples)
             X = np.vstack((p1, p2)).astype(np.float32)
             Y = np.hstack((np.ones(n_samples), np.zeros(n_samples)))

         else: # nonseparable
             p1 = np.random.multivariate_normal([0, 0], [[1, .55],[.55, 1]], n_samples)
             p2 = np.random.multivariate_normal([1, 2.5], [[1, .55],[.55, 1]], n_samples)
             X = np.vstack((p1, p2)).astype(np.float32)
             Y = np.hstack((np.ones(n_samples), np.zeros(n_samples)))
         return X,Y
```

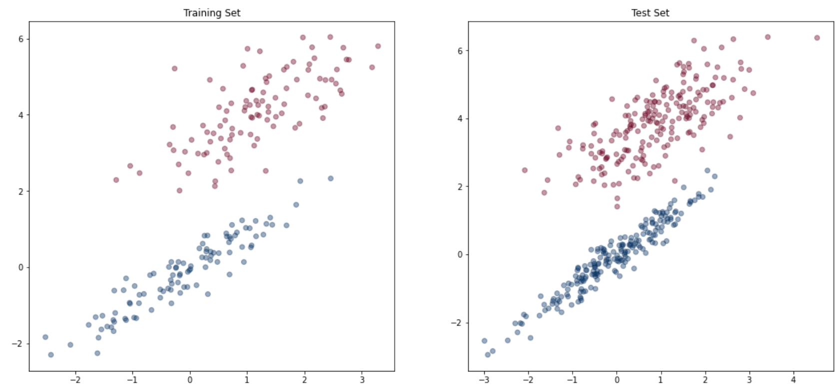
Separable Factor : True

Separating testing and training datasets

Plotting testing and training datasets to see the distribution

```
X_train, Y_train = data(n_samples = 100, seed = 123)
X_test,  Y_test  = data(n_samples = 200, seed = 321)

plt.subplots(figsize=(18, 8))
plt.subplot(1,2,1)
plt.scatter(X_train[:, 0], X_train[:, 1], c = Y_train, alpha = .4, cmap="RdBu")
plt.title("Training Set")
plt.subplot(1,2,2)
plt.scatter(X_test[:, 0],  X_test[:, 1],  c = Y_test,  alpha = .4, cmap="RdBu")
plt.title("Test Set")
plt.show()
```



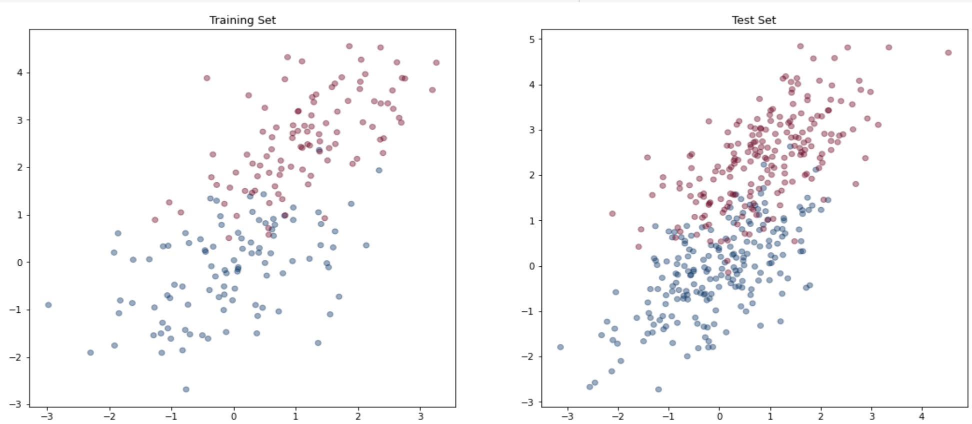
Separable Factor : False

Separating testing and training datasets

Plotting testing and training datasets to see the distribution

```
X_train, Y_train = data(n_samples = 100, seed = 123 , separable=False)
X_test,  Y_test  = data(n_samples = 200, seed = 321,separable=False)

plt.subplots(figsize=(18, 8))
plt.subplot(1,2,1)
plt.scatter(X_train[:, 0], X_train[:, 1], c = Y_train, alpha = .4, cmap="RdBu")
plt.title("Training Set")
plt.subplot(1,2,2)
plt.scatter(X_test[:, 0],  X_test[:, 1],  c = Y_test,  alpha = .4, cmap="RdBu")
plt.title("Test Set")
plt.show()
```



Code Implementation :

Importing all required Libraries for the program .

Importing all required libraries

```
[38] 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 Dropout
     from keras.layers import *
     from sklearn.metrics import mean_squared_error
     from keras import metrics
     from tensorflow import keras
     from tensorflow.keras import layers
     import warnings
     tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
     from sklearn.metrics import accuracy_score
     from sklearn.metrics import confusion_matrix
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.naive_bayes import GaussianNB
     from sklearn.tree import DecisionTreeClassifier
```

LSTM Code Implementation :

LSTM implementation :

```
fp = 0
fn = 0
tp = 0
tn = 0
for i in range(1,11):
    model1 = keras.Sequential()
    model1.add(layers.LSTM(units = 5, return_sequences = True,input_shape= (X_train.shape[1],1)))
    model1.add(layers.LSTM(units = 5, return_sequences = True))
    model1.add(Dropout(0.2))
    model1.add(Dense(units = 1))
    model1.compile(optimizer = 'adam', loss = 'mean_squared_error', metrics=[['accuracy' , 'mse']])
    model1.fit(X_train, Y_train)
    X_test1=X_test.reshape(2,1)
    prediction_rm=model1.predict(X_test1)
    for X_test1, prediction_rm in zip(X_test1, prediction_rm):
      if prediction_rm.any() == X_test1.any():
          if prediction_rm.any() == 1:
              tp += 1
          else:
              tn += 1
      else:
          if prediction_rm.any() == 1:
              fp += 1
          else:
              fn += 1
    our_confusion_matrix = [
        [tn, fp],
        [fn, tp]
    ]
    print("iteration :", i)
    print(our_confusion_matrix)
```

Output :

```
7/7 [==============================] - 4s 6ms/step - loss: 0.4717 - accuracy: 0.5000 - mse: 0.4717
iteration : 1
[[0, 0], [0, 2]]
7/7 [==============================] - 4s 6ms/step - loss: 0.4988 - accuracy: 0.5000 - mse: 0.4988
iteration : 2
[[0, 0], [0, 4]]
7/7 [==============================] - 4s 6ms/step - loss: 0.4908 - accuracy: 0.5000 - mse: 0.4908
iteration : 3
[[0, 0], [0, 6]]
7/7 [==============================] - 4s 6ms/step - loss: 0.4820 - accuracy: 0.5000 - mse: 0.4820
iteration : 4
[[0, 0], [0, 8]]
7/7 [==============================] - 4s 6ms/step - loss: 0.4934 - accuracy: 0.5000 - mse: 0.4934
iteration : 5
[[0, 0], [0, 10]]
7/7 [==============================] - 4s 6ms/step - loss: 0.4818 - accuracy: 0.5000 - mse: 0.4818
iteration : 6
[[0, 0], [0, 12]]
7/7 [==============================] - 4s 6ms/step - loss: 0.4932 - accuracy: 0.5000 - mse: 0.4932
iteration : 7
[[0, 0], [0, 14]]
7/7 [==============================] - 5s 6ms/step - loss: 0.5037 - accuracy: 0.5000 - mse: 0.5037
iteration : 8
[[0, 0], [0, 16]]
7/7 [==============================] - 4s 6ms/step - loss: 0.5307 - accuracy: 0.5000 - mse: 0.5307
iteration : 9
[[0, 0], [0, 18]]
7/7 [==============================] - 5s 6ms/step - loss: 0.5008 - accuracy: 0.5000 - mse: 0.5008
iteration : 10
[[0, 0], [0, 20]]
```

Accuracy achieved : 50%

Implementation Issues :

- Implementing Confusion matrix as the inbuilt function is not available for Deep Learning Algorithms.

Random Forest Code Implementation :

```
total_score = 0.0
for i in range(1,11):
    model2 = RandomForestClassifier()
    model2.fit(X_train,Y_train)
    Y_logr_sck=model2.predict(X_test)
    accuracy_logr_sck = accuracy_score(Y_test.flatten(), Y_logr_sck)
    print("iteration :", i)
    print('Accuracy of The Random Forest :',accuracy_logr_sck)
    print('Confusion matrix:\n', confusion_matrix(Y_test.flatten(), Y_logr_sck))
    total_score = total_score + accuracy_logr_sck
print("mean score :", (total_score/10)*100)
```

Output :

```
iteration : 1
Accuracy of The Random Forest : 0.9925
Confusion matrix:
 [[198   2]
 [  1 199]]
iteration : 2
Accuracy of The Random Forest : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 3
Accuracy of The Random Forest : 0.9925
Confusion matrix:
 [[198   2]
 [  1 199]]
iteration : 4
Accuracy of The Random Forest : 0.9875
Confusion matrix:
 [[197   3]
 [  2 198]]
iteration : 5
Accuracy of The Random Forest : 0.9925
Confusion matrix:
 [[198   2]
 [  1 199]]
iteration : 6
Accuracy of The Random Forest : 0.985
Confusion matrix:
 [[198   2]
 [  4 196]]
iteration : 7
Accuracy of The Random Forest : 0.9925
Confusion matrix:
 [[198   2]
 [  1 199]]
iteration : 8
Accuracy of The Random Forest : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 9
Accuracy of The Random Forest : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 10
Accuracy of The Random Forest : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
mean score : 99.02499999999999
```

Accuracy achieved : 99.02%

Naive Bayes Implementation :

```
total_score = 0.0
for i in range(1,11):
    model3 = GaussianNB()
    model3.fit(X_train, Y_train)
    Y_perc_sck = model3.predict(X_test)
    accuracy_perc_sck = accuracy_score(Y_test.flatten(), Y_perc_sck)
    print("iteration :", i)
    print('Accuracy of The Naive bayes :',accuracy_logr_sck)
    print('Confusion matrix:\n', confusion_matrix(Y_test.flatten(), Y_logr_sck))
    total_score = total_score + accuracy_logr_sck
print("mean score :", (total_score/10)*100)
```

Output :

```
iteration : 1
Accuracy of The Naive bayes : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 2
Accuracy of The Naive bayes : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 3
Accuracy of The Naive bayes : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 4
Accuracy of The Naive bayes : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 5
Accuracy of The Naive bayes : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 6
Accuracy of The Naive bayes : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 7
Accuracy of The Naive bayes : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 8
Accuracy of The Naive bayes : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 9
Accuracy of The Naive bayes : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 10
Accuracy of The Naive bayes : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
mean score : 99.0
```

Accuracy achieved : 99%

Decision Tree Implementation :

```
total_score = 0.0
for i in range(1,11):
    model4 = DecisionTreeClassifier()
    model4.fit(X_train, Y_train)
    Y_perc_sck = model4.predict(X_test)
    accuracy_perc_sck = accuracy_score(Y_test.flatten(), Y_perc_sck)
    print("iteration :", i)
    print('Accuracy of The Naive bayes :',accuracy_logr_sck)
    print('Confusion matrix:\n', confusion_matrix(Y_test.flatten(), Y_logr_sck))
    total_score = total_score + accuracy_logr_sck
print("mean score :", (total_score/10)*100)
```

Output :

```
iteration : 1
Accuracy of The Naive bayes : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 2
Accuracy of The Naive bayes : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 3
Accuracy of The Naive bayes : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 4
Accuracy of The Naive bayes : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 5
Accuracy of The Naive bayes : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 6
Accuracy of The Naive bayes : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 7
Accuracy of The Naive bayes : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 8
Accuracy of The Naive bayes : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 9
Accuracy of The Naive bayes : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
iteration : 10
Accuracy of The Naive bayes : 0.99
Confusion matrix:
 [[198   2]
 [  2 198]]
mean score : 99.0
```

Accuracy achieved : 99%