NAME: Narmada Peddi

NJIT UCID: np863

Email Address: np863@njit.edu

04-12-2024

Professor: Yasser Abduallah

CS 634-104 Data Mining

Final Project Report

Implementation and Code Usage

Random Forest Algorithm

Abstract:

In this project, I implemented a Random Forest algorithm that outputs a 1/ 0 based on the given dataset.

Introduction:

Random Forest Algorithm is a popular technique used for both classification and regression tasks. It is an ensemble learning algorithm which involves construction of multiple decision trees.

To sum up I have carried out the following steps in the algorithm:

- Loading the dataset, split, transformed the dataset.
- Built the model using 10 decision trees, with entropy as criterion.
- Calculated the Performance Metrics and Defined a loop for cross validation with 10 folds.
- Calculated performance metrics for each fold.
- Printed the confusion matrix.
- Printed the average performance metrics of all 10 folds.

Core Concepts and Principles:

Ensemble Size:

The Random Forest algorithm operates by aggregating the predictions of multiple decision trees

Decision Trees:

A part of Random Forest, where each tree makes decision by recursively splitting the data based on feature values.

Bootstrap sampling:

It refers to randomly sampling data points from the training set with replacement to create multiple bootstrap samples.

Project Workflow:

The implementation structure is as follows:

Data Loading and Preprocessing:

The necessary libraries (numpy, pandas, sklearn), reads the dataset from a CSV file ("thedataset.csv"), and separates the features (X) and the target variable (Y)

Feature Scaling:

Features are scaled using StandardScaler from sklearn.preprocessing.

Model Training:

We initialize a Random Forest Classifier with n_estimators= 10, and criterion= 'entropy', indicating 10 decision trees and using entropy as the criterion for splitting.

Cross- Validation Loop:

It loops through 10 epochs, each time performing 10-fold cross-validation. For each loop:

- It calculates the confusion matrix and various performance metrics.
- It prints out the performance metrics for each epoch.

Averaging Metrics:

After all epochs, it calculates the average confusion matrix, accuracy, anmd performance metrics over all folds.

Displaying Results:

Printing out all the average confusion matrix, accuracy and performance metrics.

Screenshots

FOLD 1 PERFORMACE METRICS:

```
Hey, I am Random Forest Algorithm!

Epoch 1 Performance Metrics:
                        Metric     Value
0      True Positive Rate (TPR)  0.797203
1      True Negative Rate (TNR)  0.891051
2     False Positive Rate (FPR)  0.108949
3     False Negative Rate (FNR)  0.202797
4                     Precision  0.802817
5                      F1 Score  0.800000
6             Balanced Accuracy  0.844127
7   True Skill Statistics (TSS)  0.688253
8      Heidke Skill Score (HSS)  0.689323
9                   Brier Score  0.755613
10      Brier Skill Score (BSS) -2.022450
11                    Accuracy  0.857500

Execution Time for Epoch 1: 0.07 sec
```

FOLD 2 PERFORMANCE METRICS:

```
Epoch 2 Performance Metrics:
                        Metric     Value
0      True Positive Rate (TPR)  0.797203
1      True Negative Rate (TNR)  0.891051
2     False Positive Rate (FPR)  0.108949
3     False Negative Rate (FNR)  0.202797
4                     Precision  0.802817
5                      F1 Score  0.800000
6             Balanced Accuracy  0.844127
7   True Skill Statistics (TSS)  0.688253
8      Heidke Skill Score (HSS)  0.689323
9                   Brier Score  0.755613
10      Brier Skill Score (BSS) -2.022450
11                    Accuracy  0.857500

Execution Time for Epoch 2: 0.05 sec
```

AVERAGE PERFORMANCE METRICS:

```
Average Confusion Matrix:
          Predicted 0  Predicted 1
Actual 0        229.0         28.0
Actual 1         29.0        114.0

Average Accuracy: 0.8575000000000002

Average Performance Metrics:
                                  Metric     Value
0      Average True Positive Rate (TPR)  0.797203
1      Average True Negative Rate (TNR)  0.891051
2     Average False Positive Rate (FPR)  0.108949
3     Average False Negative Rate (FNR)  0.202797
4                   Average Precision  0.802817
5                    Average F1 Score  0.800000
6             Average Balanced Accuracy  0.844127
7   Average True Skill Statistics (TSS)  0.688253
8      Average Heidke Skill Score (HSS)  0.689323
9                 Average Brier Score  0.755612
10     Average Brier Skill Score (BSS) -2.022450
11                   Average Accuracy  0.857500
```

KNN ALGORITHM

Abstract:

In this project, I implemented a KNN algorithm that outputs a 1/ 0 based on the given dataset.

Introduction:

The KNN is an algorithm which relies on the similarity between the data points to make predictions.

To sum up I have carried out the following steps in the algorithm:

- Loading the dataset, split, transformed the dataset.
- Built the model with 7 nearest neighbours, and distance metric as Euclidean.
- Calculated the Performance Metrics and Defined a loop for cross validation with 10 folds.
- Calculated performance metrics for each fold.
- Printed the confusion matrix.
- Printed the average performance metrics of all 10 folds.

Core Concepts and Principles:

Nearest Neighbors:

The k data points closest to a given query point based on a chosen distance metric.

Distance Metric:

A function used to measure the similarity or dissimilarity between two data points.

K:

The number of nearest neighbours to consider while making predictions.

Lazy Learning:

K-NN is often referred to as a lazy learning algorithm because it doesn't learn a model during ther training phase.

Project Workflow:

The implementation structure is as follows:

Data Loading and Preprocessing:

The necessary libraries (numpy, pandas, sklearn), reads the dataset from a CSV file (“thedataset.csv”), and separates the features (X) and the target variable (Y)

Feature Scaling:

Features are scaled using StandardScaler from sklearn.preprocessing.

Model Training:

We initialize a K Neighbors Classifier with n_neighbors= 7, and metric= Euclidean.

Cross- Validation Loop:

It loops through 10 epochs, each time performing 10-fold cross-validation. For each loop:

- It calculates the confusion matrix and various performance metrics.
- It prints out the performance metrics for each epoch.

Averaging Metrics:

After all epochs, it calculates the average confusion matrix, accuracy, anmd performance metrics over all folds.

Displaying Results:

Printing out all the average confusion matrix, accuracy and performance metrics.

Screenshots

FOLD 1 PERFORMACE METRICS:

```
Fold 1 Performance Metrics:
                         Metric     Value
0      True Positive Rate (TPR)  0.622378
1      True Negative Rate (TNR)  0.887160
2     False Positive Rate (FPR)  0.112840
3     False Negative Rate (FNR)  0.377622
4                   Precision  0.754237
5                    F1 Score  0.681992
6            Balanced Accuracy  0.754769
7   True Skill Statistics (TSS)  0.509537
8      Heidke Skill Score (HSS)  0.530091
9                  Brier Score  0.671113
10      Brier Skill Score (BSS) -1.684450
11                    Accuracy  0.792500

Execution Time for Fold 1: 0.01 sec
```

AVERAGE PERFORMANCE METRICS:

```
Average Confusion Matrix:
          Predicted 0  Predicted 1
Actual 0          228           29
Actual 1           54           89

Average Accuracy: 0.7925000000000001

Average Performance Metrics:
                                Metric     Value
0      Average True Positive Rate (TPR)  0.622378
1      Average True Negative Rate (TNR)  0.887160
2     Average False Positive Rate (FPR)  0.112840
3     Average False Negative Rate (FNR)  0.377622
4                   Average Precision  0.754237
5                    Average F1 Score  0.681992
6            Average Balanced Accuracy  0.754769
7   Average True Skill Statistics (TSS)  0.509537
8      Average Heidke Skill Score (HSS)  0.530091
9                  Average Brier Score  0.671112
10      Average Brier Skill Score (BSS) -1.684450
11                    Average Accuracy  0.792500
```

LSTM

Abstract:

In this project, I implemented LSTM algorithm that outputs a 1/ 0 based on the given dataset.

Introduction:

LSTM is a type of recurrent neural network (RNN) architecture that is designed to capture long-term dependencies in sequential data.

To sum up I have carried out the following steps in the algorithm:

- Loading the datasets, asking for user prompts which includes datasets, minimum support and confidence.
- Defined functions that calculate the frequent item sets and generate association rules.
- Generated association rules based on Frequent Pattern Item sets.
- Printing the association rules along with their support and confidence.
- Displaying the execution time of the program at the end.

Core Concepts and Principles:

Memory cell:

The core component of LSTM is the memory cell, which contains a cell state that can store information over time. The cell state is modified by various operations called gates.

Hidden state:

It is a filtered version of the cell state that is passed along to the next time step and used to make predictions.

Peephole Connections:

This refers to connections that allow gates to have access to the cell state.

Project Workflow:

Data Loading and Preprocessing:

The necessary libraries (numpy, pandas, sklearn), reads the dataset from a CSV file (“thedataset.csv”), and separates the features (X) and the target variable (Y)

LSTMClassifierWrapper Class:

This class acts as a wrapper around the LSTM model. It has methods to create the model, train it, and make predictions.

Create Model:

This method defines the architecture of the LSTM model using TensorFlow's Keras API. It consists of an LSTM layer with 6 units followed by a dense layer with 1 unit and a sigmoid activation function.

Feature Scaling:

Features are scaled using StandardScaler from sklearn.preprocessing.

Model Training:

We initialize a K Neighbors Classifier with n_neighbors= 7, and metric= Euclidean.

Cross- Validation Loop:

It loops through 10 epochs, each time performing 10-fold cross-validation. For each loop:

- It calculates the confusion matrix and various performance metrics.
- It prints out the performance metrics for each epoch.

Averaging Metrics:

After all epochs, it calculates the average confusion matrix, accuracy, anmd performance metrics over all folds.

Displaying Results:

Printing out all the average confusion matrix, accuracy and performance metrics.

Screenshots

FOLD 1 PERFORMACE METRICS:

```
Hey, I am LSTM Algorithm!
Fold 1:
Epoch 1/10


/Users/honeypeddi/anaconda3/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `inpu
t_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(**kwargs)


12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 702us/step - accuracy: 0.2758 - loss: 0.7324
Epoch 2/10
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 770us/step - accuracy: 0.3306 - loss: 0.7212
Epoch 3/10
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 804us/step - accuracy: 0.3907 - loss: 0.7167
Epoch 4/10
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 768us/step - accuracy: 0.4911 - loss: 0.7037
Epoch 5/10
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 680us/step - accuracy: 0.5483 - loss: 0.6972
Epoch 6/10
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 692us/step - accuracy: 0.5832 - loss: 0.6871
Epoch 7/10
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 705us/step - accuracy: 0.6160 - loss: 0.6796
Epoch 8/10
```

AVERAGE PERFORMANCE METRICS:

```
Average Performance Metrics Across Folds:
+--------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+-------
--------------+-----------+----------+--------------------+--------------------+--------------------+-------------+-------
--------------+
| Epochs |      Time (s)      |      Accuracy      |        TPR         |        TNR         |         FPR        |
FNR         | Precision | F1 Score | Balanced Accuracy  |  True Skill Score  | Heidke Skill Score | Brier Score |  Brier S
kill Score  |
+--------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+-------
--------------+-----------+----------+--------------------+--------------------+--------------------+-------------+-------
--------------+
|  10.0  | 0.25738348960876467 | 0.8699999999999999 | 0.5506115833060913 | 0.9068284881520177 | 0.09317151184798243 | 0.449
3884166939086 |    nan    |   nan    | 0.7287200357290545 | 0.45744007145810883 | 0.4921675570305351 |    0.798    | -11.17
8017316017316 |
+--------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+-------
--------------+-----------+----------+--------------------+--------------------+--------------------+-------------+-------
--------------+
```