ML Library Implementation Project

By: Madhav Sood(IMT2021009), Kalyan Ram (IMT2021023), Nilay Kamat (IMT2021096), Shlok Agrawal (IMT2021103)

Code Base:

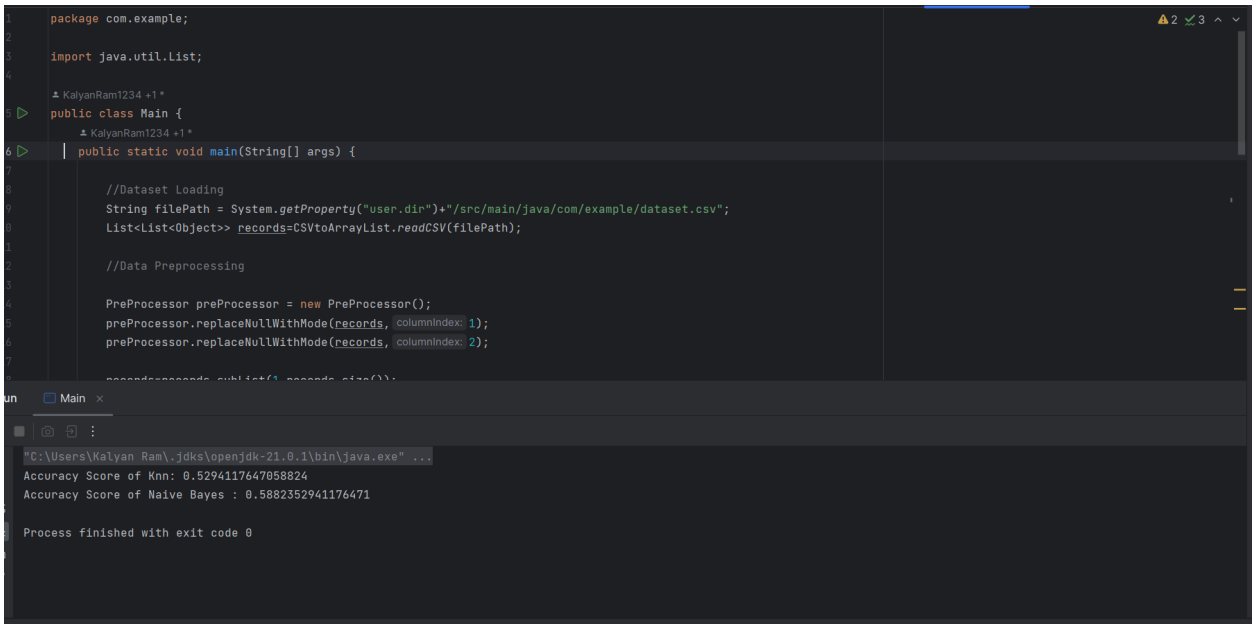
1. **Main.java**
    a. Made a sample weather dataset and showcased a sample run of the various methods implemented below.
    b. Loaded the csv file using CSVtoArrayList class.
    c. Using the PreProcessor class, we replace null values with mode, label encoded columns with string values, and normalize some of the columns.
    d. Then we split the data into train and test.
    e. Trained the dataset on both the models , i.e using the knnClassifer and the NaiveBayesClassifier class with the fit method which implements the MLModel interface.
    f. Made predictions on the test data and the accuracy scores were calculated for each of the models.
2. **MlModel.java**
    a. MlModel interface:
    Defines abstract methods fit(List<List<Double>> X_train, List<Double> y_train), predict(List<List<Double>> X_test), calculateAccuracy(List<Double> yTrue, List<Double> yPred).
3. **knnClassifier.java**
    a. public void fit(List<List<Double>> X_train, List<Double> y_train) -
    b. private Double _predict(List<Double> instance) - Will predict the class label for a particular data point.
    c. public List<Double> predict(List<List<Double>> X_test) - Uses the _predict method to return a list of class labels based on the test data
    d. private double euclideanDistance(List<Double> instance1, List<Double> instance2) - Finds the euclidean Distance between 2 data points
    e. private static class Neighbor - Stores the distance and the label
    f. public double calculateAccuracy(List<Double> yTrue, List<Double> yPred) - Calculates the accuracy of the model based on the number of correctly predicted labels
4. **NaiveBayesClassifier.java**
    a. public void fit(List<List<Double>> X, List<Double> y) - Finds the probabilities of the features and the class labels.
    b. public List<Double> predict(List<List<Double>> X_test) - Returns a list of predictions based on the test data and the trained probabilities.

c. public double calculateAccuracy(List<Double> yTrue, List<Double> yPred) - Calculates the accuracy of the model based on the number of correctly predicted labels
d. public double calculateClassProbability(Double targetClass, List<Double> y) - Find the probabilities of the y labels.
e. public double calculateFeatureProbability(Integer column,Double featureval, Double targetClass, List<List<Double>> X, List<Double> y) - Finds the probabilities of the features per column

5. **CSVtoArrayList.java**
   a. readCSV(String filePath):
   Reads data stored in each line of CSV file and stores it in a List of List of Objects after splitting using “,” delimiter.
6. **PreProcessor.java**
   a. replaceNullWithMode(List<List<Object>> data, int columnIndex):
   Replaces all null values in the column with the mode of that column.
   b. normalizeColumn(List<List<Object>> data, int columnIndex):
   Normalizes the values of the specific column.
   c. labelEncodeColumn(List<List<Object>> data, int columnIndex):
   Performs label encoding in the specific column.
   d. splitData(List<List<Object>> data, double splitRatio, int y_col):
   Randomly splits the data into training and testing data in the given splitRatio.
7. **SplittedData.java**
   a. Helper class to return the split data as x_train, y_train, x_test and y_test.

Github Link : [ML Project]

Sample Run:



Implementation of Functional and Non-Functional Requirements:

1. **Functional Requirements:**
    a. Data loading from CSV files.
    b. Data cleaning, including handling missing/null values.
    c. Data scaling, and encoding.
    d. Error handling for robustness.
    e. Training with labeled data(distance metric, and neighbors for KNN) and classification of data points.
    f. Model evaluation and persistence.
2. **Non-Functional Requirements:**
    a. Usability : Easy to use User Interface and is well-documented for new users.
    b. Efficiency : Provides efficient model training and inference on the provided dataset.
    c. Safety and Error Handling : The library provides informative error messages that help the user identify the errors in their code.
    d. Scalability : The library can handle large datasets.

Test Driven Development(TDD):

The process we followed for TDD involves the following steps:

Write a Test: First, write a test for a small unit of functionality (such as a method).

Write Code: Implement the code necessary to make the test pass. This code may be just enough to fulfill the requirements of the test.

Run Tests: Run all the tests, including the new one. If the new test passes and all previous tests still pass, then the code is likely correct.

Refactor (if needed): Refactor the code to improve its structure or efficiency, ensuring that all tests continue to pass.

Test Cases and Runs:

CSVtoArrayListTest:

Reading and Verifying the size of the dataset: assertEquals(82, records.size());

knnClassifierTest:

For fit() method: Checking size of training data: assertEquals(6, knn1.getTrainingData().size());

For predict() method: Verifying class label of first data point of testData: assertEquals(0.0, y_pred.get(0));

For calculateAccuracy() method: Checking accuracy calculated: assertEquals(accuracy, 0.66, 0.01);

NaiveBayesClassifier :

For calculateClassProbability() : Checking the class probability : assertEquals(0.5, prob, DELTA);

For calculateFeatureProbability() method : Checking the feature probability for each column : assertEquals(0.375, naiveBayesClassifier.calculateFeatureProbability(1, 15.0, 1.0, XNew, YNew), DELTA);

For calculateAccuracy() method : Checking accuracy calculated : assertEquals(0.75, naiveBayesClassifier.calculateAccuracy(YTrue, YPred), DELTA);

PreProcessorTest:

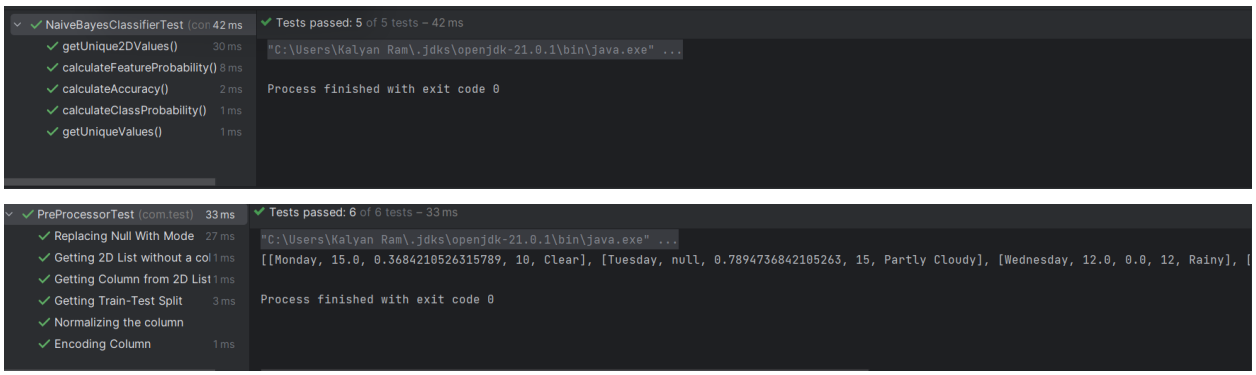
replaceNullWithMode: verifying for null value in row of column 1: assertEquals(12.0, (double) data.get(1).get(1),DELTA);

normalizeColumn(): verifying normalized value for a particular cell (normalizing column 2): assertEquals(0.789, (Double) data.get(1).get(2), DELTA);

labelEncodeColumn(): label encoding first column and verifying value for a particular row: assertEquals(1.0, (Double) data.get(1).get(0), DELTA);

splitData(): verifying size of split dataset according to ratio provided: assertEquals(4,newData.get_x_train().size());

Screenshots of Result of Running the Testcases:



```java
class CSVtoArrayListTest {

    KalyanRam1234
    @Test
    void readCSV() {

        String filePath = System.getProperty("user.dir")+"/src/main/java/com/example/dataset.csv";
        List<List<Object>> records=CSVtoArrayList.readCSV(filePath);
        assertEquals( expected: 82, records.size()); // Assuming 30 rows in the test CSV

        // Check values for a specific row (e.g., the first row)
        List<Object> firstRow = records.get(1);
        assertEquals( expected: "Monday", firstRow.get(0)); // Assuming the first column is "Day"
        assertEquals( expected: "15", firstRow.get(1));    // Assuming the second column is "Temperature (Celsius)"
        assertEquals( expected: "65", firstRow.get(2));    // Assuming the third column is "Humidity (%)"
        assertEquals( expected: "10", firstRow.get(3));    // Assuming the fourth column is "Wind Speed (km/h)"
        assertEquals( expected: "Clear", firstRow.get(4));
    }
}
```

un CSVtoArrayListTest ×

CSVtoArrayListTest (com.test) 30 ms
readCSV() 30 ms

Tests passed: 1 of 1 test – 30 ms

knnClassifierTest (com.test) 34 ms
predict() 32 ms
fit() 1 ms
calculateAccuracy() 1 ms

Tests passed: 3 of 3 tests – 34 ms

```
"C:\Users\Kalyan Ram\.jdks\openjdk-21.0.1\bin\java.exe" ...

Process finished with exit code 0
```