Simplified Predictive Modeling In Java

1 Introduction

The goal of this tutorial is to provide example code for predictive modeling in Java, using three custom-built classes: `Data`, `Plot`, and `LearningMachine`. These classes are designed so that the user does not have to worry about the details of the code, but instead can focus on learning the basics of predictive modeling.

2 The Data Class

In this example, we will be using the `ICU_data.csv` dataset. Each row of the CSV contains data on a single patient. First, we read the data into the `Data` class:

```
String file = "ICU_Data.csv";
Data d = new Data(file);
```

The `Data` constructor reads in a file location or a URL and stores the data in a `String[][]` object. After storing the first row of the CSV as the instance variable `String[][] headers`, it then randomly splits the data into training/testing/validation sets and stores them as the instance variables `String[][] train`, `String[][] test`, and `String[][] valid`.

Let's take a look at the first few headers for our dataset.

```
for (int i = 1; i < 6; i++) {
    System.out.print(d.headers[i] + " ");
}
```

This gives us an output of:

```
Age DiasABP FiO2 GCS Gender
```

The last column of our dataset is the binary classification variable we want to predict - whether or not the patient died in the ICU.

```
System.out.println(d.headers[d.headers.length - 1]);
```

This gives us an output of:

```
In-hospital_death
```

The `Data` class also has three methods for obtaining a subset of the data according to a specific filter: `getRows`, `getCols`, and `convertToDouble`. For example, suppose we want to only look at the patients in the training data who are male. We can use the `getRows` method:

```
int genderIndex = Arrays.asList(d.headers).indexOf("Gender");
String[][] male = d.getRows(d.train, genderIndex, "M");
String[][] female = d.getRows(d.train, genderIndex, "F");
```

If we wanted the GCS (Glasgow Coma Scale) score as well as the age of all the patients in the training data, we can use the `getCols` method:

```
int ageIndex = Arrays.asList(d.headers).indexOf("Age");
int GCSIndex = Arrays.asList(d.headers).indexOf("GCS");
String[][] twoCols = d.getCols(d.train, new int[] {ageIndex, GCSIndex});
```

Finally, suppose we just wanted the age of the all the training data patients, as well as separate age data for each gender. Since we know `age` is a numerical variable, we can use the `convertToDouble` method, which gives us a `double[]` rather than a `String[]`.

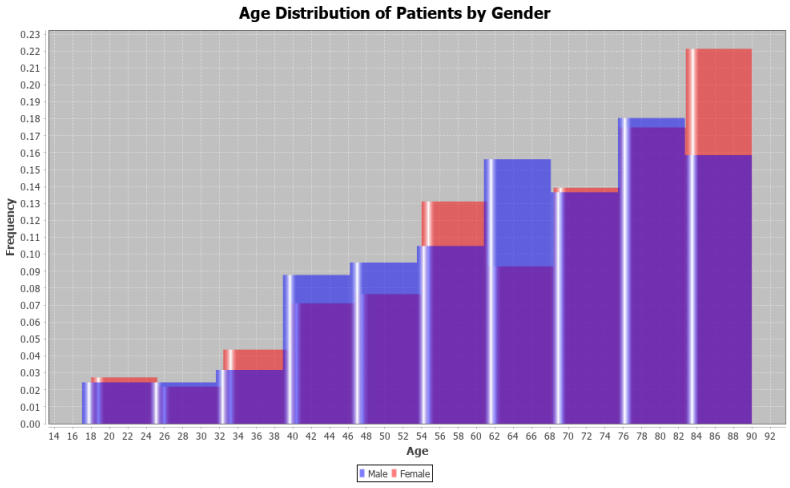
```
double[] ages = d.convertToDouble(d.train, ageIndex);
double[] agesM = d.convertToDouble(male, ageIndex);
double[] agesF = d.convertToDouble(female, ageIndex);
```

3 The Plot Class

The `Plot` class contains two methods: `plotHistogram` and `plotRates`. The class uses the `JFreeChart` Java library to generate graphs of our data. For example, suppose we want to visualize the `agesM` and `agesF` arrays in two histgrams:

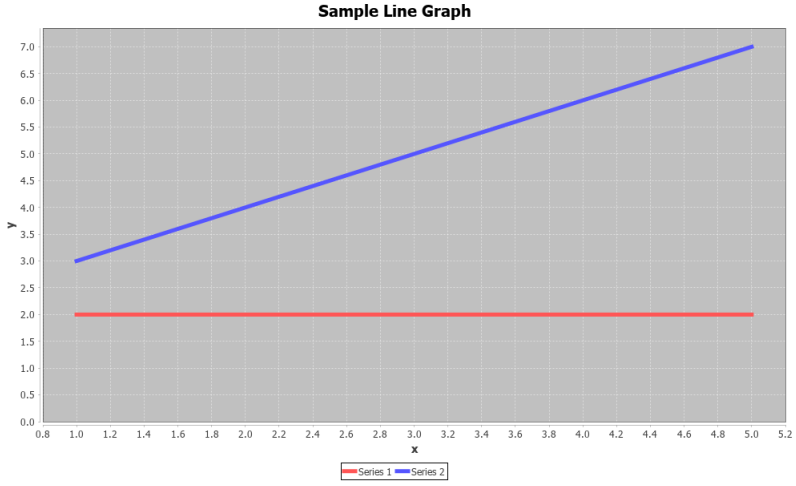
```
double[][] allAges = {agesM, agesF};
String[] legend = {"Male", "Female"};
Plot.plotHistogram(allAges, legend, "ICU_agesG.png",
        "Age Distribution of Patients by Gender", "Age", "Frequency");
```

This will produce the following output:



The `plotRates` method also can plot multiple line charts on the same graph. Below is a simple example on how to call the `plotRates` method:

```
double[] sampleX = {1, 5};
double[] sampleY1 = {2, 2};
double[] sampleY2 = {3, 7};
double[][] allY = {sampleY1, sampleY2};
String[] seriesNames = {"Series 1", "Series 2"};
Plot.plotRates(sampleX, allY, seriesNames, "sample_line_graph.png",
                "Sample Line Graph", "x", "y");
```



4 The LearningMachine Class

The `LearningMachine` class uses the `Weka` library, which contains tools for data mining and machine learning algorithms. `LearningMachine` has the instance variable `fit`, which is the `Weka` classifier for the model. The `LearningMachine` constructor has the following parameters:

- `String[] preds`: the names of the variables the model should use
- `Data d`: the `Data` object that `trainData` comes from
- `String[][] trainData`: the data used to build the model

For example, if we want to build a classifier using `Age` and `GCS` as predictor variables, with the goal to predict `In-hospital_death`:

```
String[] preds = {"Age", "GCS", "In-hospital_death"};
LearningMachine model = new LearningMachine(preds, d, d.train);
System.out.println(model.fit);
```

This outputs the coefficients of our model, as well as the odds ratio:

```
Logistic Regression with ridge parameter of 1.0E-8
Coefficients...
               Class
Variable           0
====================
Age          -0.0359
GCS           0.2404
Intercept    -0.1952


Odds Ratios...
               Class
Variable           0
====================
Age           0.9647
GCS           1.2717
```

`LearningMachine` has two `predict` methods, both with essentially the same function. One returns the model's predicted probability of death for a sample patient, while the other does this for multiple patients.

Suppose we have a 45-year-old patient with a GCS score of 7. To see what the model predicts the probability of this patient dying in the ICU, we have:

```
double prob = model.predict(new String[] {"45", "7"}, preds);
System.out.println(prob);
```

This gives us an output of:

```
0.46771888210572526
```

Note that the first parameter in `predict` is a `String[]` object, even though both of our predictor variables are of type `double`. This is to allow categorical variables to be used in other potential models.

Now, suppose we want to see what the model predicts on our entire validation set. We can use the other `predict` method as follows:

```
double probs[] = model.predict(d.valid, preds);
System.out.println("Validation set size: " + d.valid.length);
System.out.println("probs[] array length: " + probs.length);
```

Our output is:

```
Validation set size: 166
probs[] array length: 166
```

We see that both arrays are the same length - for the patient in row `i` of the validation set, our model's predicted probability for their death in the ICU is `probs[i]`.