CAP6673 Homework 1

Hector Lopez ; Spring 2018

I.A Engineering the input: Preparing the datasets

The assignement is to prepare the datasets in a format required to be used with Weka. We will convert the dataset files, "fit.dat" and "test.dat" into ARFF format. One set will be used to build and evaluate prediction models, and the other to build and evaluate classification models. The "fit" data can also be seen as the training data. The training data is the data we will use to train the model.

Understanding the data

The dataset indicates different classes for each column value per row. The values are defined as follows in the same order :

1. Number of unique operators (NUMUORS)
2. Number of unique operands (NUMUANDS)
3. Total number of operators (TOTOTORS)
4. Total number of operands (TOTOPANDS)
5. McCabe's cyclomatic complexity (VG)
6. Number of logical operators (NLOGIC)
7. Lines of code (LOC)
8. Executable line of code (ELOC)
9. Number of faults (FAULTS)

The data represents software written in the ADA programming language in a way that can grade its complexity and also provide it proclivity to having faults. The data uses the number of operators, operands and McCabe's cyclomatic complexity, amongst other features to create a picture of the softwares complexity.

Set 1: [For Prediction]

The fit.dat file is a space seperated file. The .ARFF format we require for Weka takes the data as a comma-seperated file format. With the python code below we can convert the file quickly with minimal human error. We can make it into a function for future use.

```
import csv
#Local training data
#Open fit.dat file and replace each space with a comma
#this prepares it for the ARFF format
ConvertSpaceToCSV("fit.dat","fit.csv")
```

```
def ConvertSpaceToCSV(source_file,dest_file):
    with open(source_file, "r") as infile:
        reader = csv.reader(infile, delimiter=' ')
        with open(dest_file, "w", newline='') as outfile:
            writer = csv.writer(outfile, delimiter = ',')
            for row in reader:
                writer.writerow(row)
```

The fit.dat file goes from the space-seperated data set to a comma-seperated data set.

Space Seperated (fit.dat):

```
22 85 203 174 9 0 362 40 0
21 87 186 165 5 0 379 32 0
30 107 405 306 25 0 756 99 0
...
```

Comma Seperated (fit.csv):

```
22,85,203,174,9,0,362,40,0
21,87,186,165,5,0,379,32,0
30,107,405,306,25,0,756,99,0
...
```

The .arff file needs to have the attributes created for the Weka tool to identify each of the columns as a specific attribute. Lets add the following comments and attributes to the top of the csv file and save it as an .ARFF file format for Weka.

```
comments = """
% ADA Software Fault Training Data
% Author: Hector Lopez
% 9 attributes
% 188 instances
% fit.arff
"""

attributes ="""
@relation fit_data_frame

@attribute NUMUORS real
@attribute NUMUANDS real
@attribute TOTOTORS real
@attribute TOTOPANDS real
@attribute VG real
@attribute NLOGIC real
@attribute LOC real
@attribute ELOC real
@attribute FAULTS real

@data
"""

#Add comments and weka attribs to the csv file and save as arf
header = comments + attributes;
AddWekaFeatures("fit.csv","fit.arff",header);
```

```
def AddWekaFeatures(source_file,dest_file,header):
    infile = open(source_file, "r");
    intext = header + infile.read();
    outfile = open(dest_file, "w");
    outfile.write(intext);
```

The final .arff file is now in a Weka format :

```
% ADA Software Fault Training Data
% Author: Hector Lopez
% 9 attributes
% 188 instances
% fit.arff

@relation fit_data_frame

@attribute NUMUORS real
@attribute NUMUANDS real
@attribute TOTOTORS real
@attribute TOTOPANDS real
@attribute VG real
@attribute NLOGIC real
@attribute LOC real
@attribute ELOC real
@attribute FAULTS real

@data
22,85,203,174,9,0,362,40,0
21,87,186,165,5,0,379,32,0
30,107,405,306,25,0,756,99,0
6,5,19,6,2,0,160,9,0
21,47,168,148,7,0,352,29,0
...
```

The same methods to clean the test.dat dataset was used. But , the test.dat file is tab seperated not space seperated. We can replace the dillimiter we are using in our function and create a new one to condition our test.dat file.

```
#Open test.dat file and replace each tab with a comma
#this prepares it for the ARFF format
ConvertTabToCSV("test.dat","test.csv");

comments = """
% ADA Software Fault Test Data
% Author: Hector Lopez
% 9 attributes
% 94 instances
% test.arff
"""

header = comments + attributes

AddWekaFeatures("test.csv","test.arff",header);
```

```
def ConvertTabToCSV(source_file,dest_file):
    with open(source_file, "r") as infile:
        reader = csv.reader(infile, delimiter='\t')
        with open(dest_file, "w", newline='') as outfile:
            writer = csv.writer(outfile, delimiter = ',')
            for row in reader:
                writer.writerow(row)
```

The completed test data set is formatted as a .arff file ready to be imported into Weka.

```
% ADA Software Fault Test Data
% Author: Hector Lopez
% 9 attributes
% 94 instances
% test.arff

@relation test_data_frame

@attribute NUMUORS real
@attribute NUMUANDS real
@attribute TOTOTORS real
@attribute TOTOPANDS real
@attribute VG real
@attribute NLOGIC real
@attribute LOC real
@attribute ELOC real
@attribute FAULTS real

@data
6,12,127,45,10,0,641,55,0
5,5,41,12,1,0,407,17,0
23,28,95,66,4,2,241,20,0
5,5,35,20,1,0,254,14,0
...
```

Set 2 [For Classification]

Lets create an .ARFF file for classification in Weka. First we must change our fit.arff and test.arff file for classification by creating a way to identify the software modules. We need to add a column describing the class of each module: fault-prone (fp) or not fault-prone(nfp). Fault proneness is based on a threshold of number of faults. We will assume that modules with less than 2 faults are considered nfp, and modules with 2 or more faults are considered fp. We will make sure to not use the number of faults column as an independent variable while doing classification

I.B Modeling assignment: Prediction

We will attempt numerical prediction based on the data conditioned in the previous section. The data has a "FAULTS" attribute that we shall use as the dependent variable for our predictive modeling. The FAULTS indicate the number of faults recorded in each software program instance analyzed in the data set.

The numerical prediction methods used will be *Linear Regression* and *Decision Stump*. Both methods will use the k-fold Cross-Validation scheme to quantify the effectiveness of each model. We will then use a "test" data set to analyze the effectiveness with new data points.

Defining Terms

Cross Validation

Cross-validation is used to reduce overfitting bias by using the same data for training as for testing. It takes the entire testing data set and seperates it into "k" parts. All parts are used to train the model except one. The one set aside is used to test the model. This process repeats cycling through by changing the part used for testing. Each part is considered a "fold" . When there are "k" folds the process repeats "k" times. The performance of each fold is averaged across the itterations and provided as the performance of the entire prediction method.

Measurement Methods

The numerical analysis of the dataset can give us an indication of the data. There are many ways to view the behaviour of the training data and more importantly the resulting prediction data.
If we reserve some points in the training data and use our prediction method to "guess" the known point we can determine an error between our test point and our predicted point. The results of these errors can be viewed in different ways. Each analysis gives us a clue about the predictive models effectiveness.

- Mean Absolute Error (MAE)
    - similar to RMSE , but provides the error in a arithmetic distance to the correct value.
- Root Mean Square Error (RMSE)
    - Provides the error in terms of the average euclidean distance to the correct value for each point.
- Releative Absolute Error (RAE)
    - Using the absolute value of relative points around the predicted point and using its distanc as error
- Root Relative Squared Error (R^2)
    - The relative distance error but taking the square root of that distance to get the error.

Build Prediction Models

Import the "fit.arff" dataset into Weka and process the data using the Linear Regression classifier. The Linear Regression classifier can be adjusted to use M5, or Greedy attribute selection, as well as no attribute selection. Using the different configurations compare the outputs of each. Also use the Decision Stump cassifier and compare it along side the linear regression classifier outputs aswell. The datasets are tested using the cross-validation method and then tested using a "test" data set.

Using Weka 3.8 Explorer :

1. Select the "Preprocess Tab", and click on "Open File"
2. Import the "fit_pred.arff" file.
3. Select the "Classify" Tab and click on "choose" under classifier section.
4. Select the "Weka\Classifier\functions\LinearRegression" classifier. Use "Weka\Classifier\tree\DecisionStump" accordingly.
5. Double click on the text box that holds the classifier label, it should open a dialog box that allows the attributeSelection option to be changed from None, to M5 or Greedy.
6. Choose Testing method :
    1. Select "cross-entropy" with a k of 10
    2. Load test set file
        1. Select "supplied test set"
        2. Select "open file" in dialog window and navigate to "test_pred.arff"
        3. select close on dialog box
7. Select "Start" to get the results in the output window

Predictive Models

Linear Regression - Attribute Selection ; None

The linear regression algorithm with default attribute selection. Notice that the model uses 8 out of the 9 attributes.
The 9th Attribute is the "Faults" attribute that is the dependent variable.

```
FAULTS =
    -0.0517 * NUMUORS +
    0.0341 * NUMUANDS +
    -0.0026 * TOTOTORS +
    -0      * TOTOPANDS +
    -0.0372 * VG +
    0.2118 * NLOGIC +
    0.0018 * LOC +
    0.005  * ELOC +
    -0.309
```

Linear Regression - Greedy

The linear regression method that uses the "Greedy" approach. This approach tries to clear out outliers. When the outliers are removed there are less constraints to adhere to all the attributes. Notice that this method only uses 6 our of the 9 independent variables.

```
FAULTS =
    -0.0482 * NUMUORS +
    0.0336 * NUMUANDS +
    -0.0021 * TOTOTORS +
    -0.0337 * VG +
    0.2088 * NLOGIC +
    0.0019 * LOC +
    -0.3255
```

Linear Regression M5

The M5 linear regression technique tries to model 5 different models with the data and create a linear regression line for each one. The models are then used together to create an overall prediction curve.

```
FAULTS =
    -0.0516 * NUMUORS +
    0.0341 * NUMUANDS +
    -0.0027 * TOTOTORS +
    -0.0372 * VG +
    0.2119 * NLOGIC +
    0.0018 * LOC +
    0.005  * ELOC +
    -0.3091
```

Decision Stump

```
Classifications

NLOGIC <= 14.0 : 1.3806818181818181
NLOGIC > 14.0 : 15.333333333333334
NLOGIC is missing : 2.271276595744681
```

Results

The first set of data used to analyze the prediction functions could be considered our "training" data. This is the data that we use to build the models. When we say "build" the models this means analyzing the data set and choosing the attributes necessary to create an accurate prediction using the given algorithms such as different linear regression techniques or decision stump.

Training DataSet : Predictive Model Results

Using a 10-fold cross validation technique , analyze the models
that where built from the training data. The results show that with the training data we can create points that validate our prediction algorithms. Here are the results of that cross validation.

	LinR. None	LinR. Greedy	LinR. M5	DecisionStump
Corr. Coef.	0.7969	0.7961	0.7935	0.5941

MAE	1.6902	1.6939	1.7017	2.2173
RMSE	2.8362	2.8425	2.8612	3.7905
RAE	58.47%	58.60%	58.87%	76.70%
R^2	60.86%	60.99%	61.39%	81.3%

Test DataSet : Predictive Model Results

The "test" data is a set of data that was used to just test the trained models. Remember that we already built the models with the "training" data. So now instead of using a corss-validation technique to measure performance we are going to use this new set of "test" data that is from the same system but can be considered *new* data points. This is the closest to a real world performance measurement that we can get. Here are the results of our algorithms in the *wild*.

	LinR. None	**LinR. Greedy**	**LinR. M5**	**DecisionStump**
Corr. Coef.	0.829	0.8314	0.829	0.711
MAE	1.8377	1.8383	1.8376	2.2952
RMSE	3.7317	3.6895	3.7324	4.1928
RAE	58.64%	58.66%	58.64%	73.24%
R^2	63.68%	62.96%	63.7%	71.55%

Overall Performance

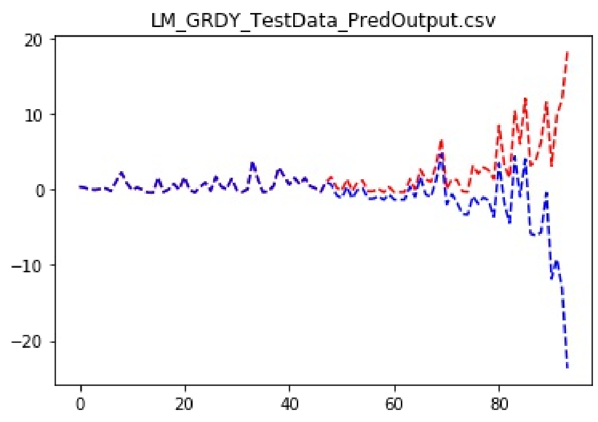
The results of the cross-validation and testing data show clear advantages when using the linear regression models over the decision stump method. The decision stump only uses one decision to split the data and classify it. The linear regression methods performances are very close to each other.

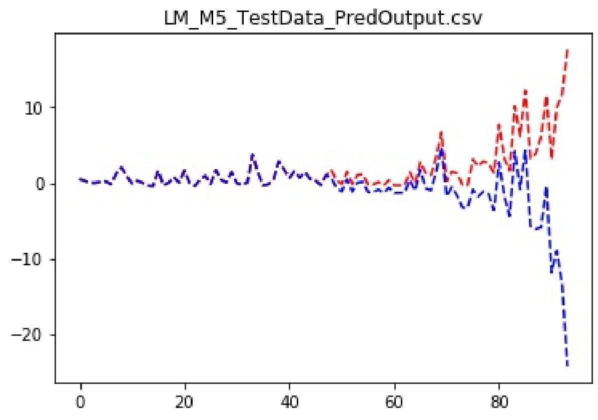
The Linear Regression "greedy" has better indicators than the other Linear Regression methods. Notice that *Greedy* has the lowest Root Mean Square error and has the least amount of independent variables. The *Greedy* method only uses 6 out of the 8 independent variables in the model. The reason this is beneficial is that with less independent variables in the formula we are able to not get into an *overfitting* condition.

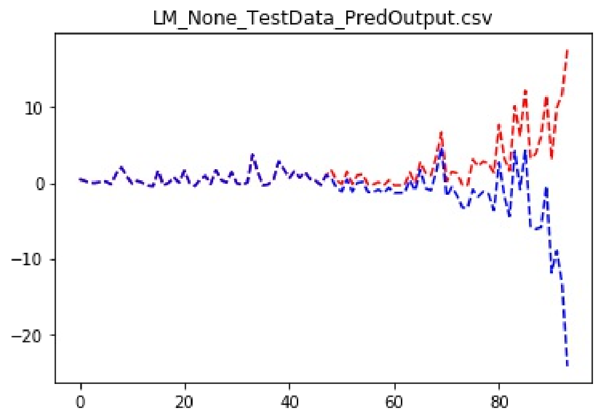
Reviewing the Output

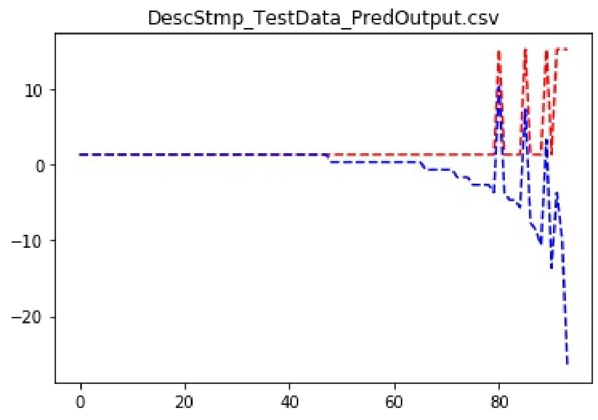
Using the weka tool we could output the values from our models as they are tested with the testing data.

The graphs below show the *predicted values in blue*, and the *training values in red*.









These graphs show something very interesting. The predicted values are diverging from the test data to the point where the last predicted value is nowhere near the real value. So just looking at the indicators the story was not as clear. Even though the indicators show improvements in error the real story is that non of these predictive models are sufficient for accurate predictions. They are either overfitted or they are not trained well enough.

References

(1). Witten, I., Frank, E. (2005), Data Mining Practical Machine Learning Tools and Techniques, 2nd edition, Elsevier Inc.

(2). G. Papageorgiou, P. Bouboulis and S. Theodoridis, "Robust Linear Regression Analysis— A Greedy Approach," in IEEE Transactions on Signal Processing, vol. 63, no. 15, pp. 3872-3887, Aug.1, 2015.

(3). Al-Abadi, Alaa. (2014). Modeling of stage–discharge relationship for Gharraf River, southern Iraq using backpropagation artificial neural networks, M5 decision trees, and Takagi–Sugeno inference system technique: a comparative study. Applied Water Science. 6. . 10.1007/s13201-014-0258-7.

Appendix

Source Code

I.A Processing Data (Python)

```
import csv

#Convert data files into csv .arff file
ConvertSpaceToCSV("fit.dat","fit_pred.arff")
ConvertTabToCSV("test.dat","test_pred.arff")

fit_comments = """
% ADA Software Fault Training Data
% Author: Hector Lopez
% 9 attributes
% 188 instances
% fit.arff
"""

test_comments = """
% ADA Software Fault Test Data
% Author: Hector Lopez
% 9 attributes
% 94 instances
% test.arff
"""

attributes ="""

@relation prediction_data_frame

@attribute NUMUORS real
@attribute NUMUANDS real
@attribute TOTOTORS real
@attribute TOTOPANDS real
@attribute VG real
@attribute NLOGIC real
@attribute LOC real
@attribute ELOC real
@attribute FAULTS real

@data
"""

#Add comments and weka attribs to the csv file and save as arf
AddWekaFeatures("fit_pred.arff",fit_comments + attributes);
AddWekaFeatures("test_pred.arff",test_comments + attributes);


print(open("fit_pred.arff","r",encoding="UTF8").read(500))
print("...")

print(open("test_pred.arff","r",encoding="UTF8").read(500))
print("...")

#Convert data files into csv .arff file
ConvertSpaceToCSV("fit.dat","fit_class.csv")
ConvertTabToCSV("test.dat","test_class.csv")

AddClassFeature("fit_class.csv","fit_class.arff")
AddClassFeature("test_class.csv","test_class.arff")

fit_comments = """
% ADA Software Fault Training Data
% Author: Hector Lopez
% 9 attributes
% 188 instances
% fit.arff
"""

test_comments = """
% ADA Software Fault Test Data
% Author: Hector Lopez
% 9 attributes
% 94 instances
% test.arff
"""
```

```
attributes ="""

@relation classification_data_frame

@attribute class {fp,nfp}
@attribute NUMUORS real
@attribute NUMUANDS real
@attribute TOTOTORS real
@attribute TOTOPANDS real
@attribute VG real
@attribute NLOGIC real
@attribute LOC real
@attribute ELOC real
@attribute FAULTS real


@data
"""


#Add comments and weka attribs to the csv file and save as arf
AddWekaFeatures("fit_class.arff",fit_comments + attributes);
AddWekaFeatures("test_class.arff",test_comments + attributes);

print(open("fit_class.arff","r",encoding="UTF8").read(500))
print("...")

print(open("test_class.arff","r",encoding="UTF8").read(500))
print("...")


def ConvertSpaceToCSV(source_file,dest_file):
    with open(source_file, "r") as infile:
        reader = csv.reader(infile, delimiter=' ')
        with open(dest_file, "w", newline='') as outfile:
            writer = csv.writer(outfile, delimiter = ',')
            for row in reader:
                writer.writerow(row)

def ConvertTabToCSV(source_file,dest_file):
    with open(source_file, "r") as infile:
        reader = csv.reader(infile, delimiter='\t')
        with open(dest_file, "w", newline='') as outfile:
            writer = csv.writer(outfile, delimiter = ',')
            for row in reader:
                writer.writerow(row)

def AddWekaFeatures(file,header):
    infile = open(file, "r");
    intext = header + infile.read();
    outfile = open(file, "w");
    outfile.write(intext);


#Add A nfp(non-fault-prone)/fp(fault-prone) class to each row
def AddClassFeature(source_file,dest_file) :
    with open(source_file, "r") as infile:
            reader = csv.reader(infile, delimiter=' ')
            with open(dest_file, "w", newline='') as outfile:
                writer = csv.writer(outfile, delimiter = ',')
                for row in reader:
                    vals=row[0].split(",")
                    idx=len(vals)-1
                    if(int(vals[idx])>=2):
                        vals.insert(0,"fp")
                    else:
                        vals.insert(0,"nfp")
                    writer.writerow(vals)
```

I.B Modeling Graphs (Python)

```
import matplotlib.pyplot as plt
import numpy as np

pred_files = ['LM_GRDY_TestData_PredOutput.csv',
              'LM_M5_TestData_PredOutput.csv',
              'LM_None_TestData_PredOutput.csv',
              'DescStmp_TestData_PredOutput.csv'
             ]

for f in pred_files :
    actuals = []
    predicted =[]
    with open(f, "r") as infile:
            reader = csv.reader(infile, delimiter=' ')
            for row in reader:
                    vals=row[0].split(",")
                    if(vals[1]=="actual"):
                        continue
                    else:
                        actuals.append(float(vals[2]))
                        predicted.append(float(vals[3]))
    t = np.arange(1, 1, len(actuals))
    plt.plot(actuals,'r--',predicted,'b--')
    plt.title(f)
    plt.show()
```

Training Data Results

Linear Regression - Attribute Selection ; None

```
=== Run information ===

Scheme:       weka.classifiers.functions.LinearRegression -S 1 -R 1.0E-8 -num-decimal-places 4
Relation:     prediction_data_frame
Instances:    188
Attributes:   9
            NUMUORS
            NUMUANDS
            TOTOTORS
            TOTOPANDS
            VG
            NLOGIC
            LOC
            ELOC
            FAULTS
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===


Linear Regression Model

FAULTS =

    -0.0517 * NUMUORS +
    0.0341 * NUMUANDS +
    -0.0026 * TOTOTORS +
    -0      * TOTOPANDS +
    -0.0372 * VG +
    0.2118 * NLOGIC +
    0.0018 * LOC +
    0.005  * ELOC +
    -0.309

Time taken to build model: 0 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.7969
Mean absolute error                      1.6902
Root mean squared error                  2.8362
Relative absolute error                 58.4755 %
Root relative squared error             60.8616 %
Total Number of Instances              188
```

Linear Regression - Greedy

```
=== Run information ===

Scheme:       weka.classifiers.functions.LinearRegression -S 2 -R 1.0E-8 -num-decimal-places 4
Relation:     prediction_data_frame
Instances:    188
Attributes:   9
            NUMUORS
            NUMUANDS
            TOTOTORS
            TOTOPANDS
            VG
            NLOGIC
            LOC
            ELOC
            FAULTS
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===


Linear Regression Model

FAULTS =

    -0.0482 * NUMUORS +
    0.0336 * NUMUANDS +
    -0.0021 * TOTOTORS +
    -0.0337 * VG +
    0.2088 * NLOGIC +
    0.0019 * LOC +
    -0.3255

Time taken to build model: 0 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.7961
Mean absolute error                      1.6939
Root mean squared error                  2.8425
Relative absolute error                 58.6027 %
Root relative squared error             60.9977 %
Total Number of Instances              188
```

Linear Regression M5

```
=== Run information ===

Scheme:       weka.classifiers.functions.LinearRegression -S 0 -R 1.0E-8 -num-decimal-places 4
Relation:     prediction_data_frame
Instances:    188
Attributes:   9
            NUMUORS
            NUMUANDS
            TOTOTORS
            TOTOPANDS
            VG
            NLOGIC
            LOC
            ELOC
            FAULTS
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===


Linear Regression Model

FAULTS =

    -0.0516 * NUMUORS +
    0.0341 * NUMUANDS +
    -0.0027 * TOTOTORS +
    -0.0372 * VG +
    0.2119 * NLOGIC +
    0.0018 * LOC +
    0.005  * ELOC +
    -0.3091

Time taken to build model: 0 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.7935
Mean absolute error                      1.7017
Root mean squared error                  2.8612
Relative absolute error                 58.8734 %
Root relative squared error             61.3972 %
Total Number of Instances              188
```

Decision Stump

```
=== Run information ===

Scheme:       weka.classifiers.trees.DecisionStump
Relation:     prediction_data_frame
Instances:    188
Attributes:   9
            NUMUORS
            NUMUANDS
            TOTOTORS
            TOTOPANDS
            VG
            NLOGIC
            LOC
            ELOC
            FAULTS
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

Decision Stump

Classifications

NLOGIC <= 14.0 : 1.3806818181818181
NLOGIC > 14.0 : 15.333333333333334
NLOGIC is missing : 2.271276595744681


Time taken to build model: 0 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.5941
Mean absolute error                      2.2173
Root mean squared error                  3.7905
Relative absolute error                 76.7091 %
Root relative squared error             81.3406 %
Total Number of Instances              188
```

Test Data Results

Linear Regression - Attribute Selection None

```
=== Run information ===

Scheme:       weka.classifiers.functions.LinearRegression -S 1 -R 1.0E-8 -num-decimal-places 4
Relation:     prediction_data_frame
Instances:    188
Attributes:   9
            NUMUORS
            NUMUANDS
            TOTOTORS
            TOTOPANDS
            VG
            NLOGIC
            LOC
            ELOC
            FAULTS
Test mode:    user supplied test set:  size unknown (reading incrementally)

=== Classifier model (full training set) ===


Linear Regression Model

FAULTS =

    -0.0517 * NUMUORS +
    0.0341 * NUMUANDS +
    -0.0026 * TOTOTORS +
    -0      * TOTOPANDS +
    -0.0372 * VG +
    0.2118 * NLOGIC +
    0.0018 * LOC +
    0.005  * ELOC +
    -0.309

Time taken to build model: 0.07 seconds

=== Evaluation on test set ===

Time taken to test model on supplied test set: 0 seconds

=== Summary ===

Correlation coefficient                  0.829
Mean absolute error                      1.8377
Root mean squared error                  3.7317
Relative absolute error                 58.6426 %
Root relative squared error             63.6881 %
Total Number of Instances               94
```

Linear Regression - Greedy

```
=== Run information ===

Scheme:       weka.classifiers.functions.LinearRegression -S 2 -R 1.0E-8 -num-decimal-places 4
Relation:     prediction_data_frame
Instances:    188
Attributes:   9
            NUMUORS
            NUMUANDS
            TOTOTORS
            TOTOPANDS
            VG
            NLOGIC
            LOC
            ELOC
            FAULTS
Test mode:    user supplied test set:  size unknown (reading incrementally)

=== Classifier model (full training set) ===


Linear Regression Model

FAULTS =

    -0.0482 * NUMUORS +
    0.0336 * NUMUANDS +
    -0.0021 * TOTOTORS +
    -0.0337 * VG +
    0.2088 * NLOGIC +
    0.0019 * LOC +
    -0.3255

Time taken to build model: 0.02 seconds

=== Evaluation on test set ===

Time taken to test model on supplied test set: 0 seconds

=== Summary ===

Correlation coefficient                  0.8314
Mean absolute error                      1.8383
Root mean squared error                  3.6895
Relative absolute error                 58.6625 %
Root relative squared error             62.968  %
Total Number of Instances               94
```

Linear Regression M5

```
=== Run information ===

Scheme:       weka.classifiers.functions.LinearRegression -S 0 -R 1.0E-8 -num-decimal-places 4
Relation:     prediction_data_frame
Instances:    188
Attributes:   9
            NUMUORS
            NUMUANDS
            TOTOTORS
            TOTOPANDS
            VG
            NLOGIC
            LOC
            ELOC
            FAULTS
Test mode:    user supplied test set:  size unknown (reading incrementally)

=== Classifier model (full training set) ===


Linear Regression Model

FAULTS =

    -0.0516 * NUMUORS +
    0.0341 * NUMUANDS +
    -0.0027 * TOTOTORS +
    -0.0372 * VG +
    0.2119 * NLOGIC +
    0.0018 * LOC +
    0.005  * ELOC +
    -0.3091

Time taken to build model: 0 seconds

=== Evaluation on test set ===

Time taken to test model on supplied test set: 0 seconds

=== Summary ===

Correlation coefficient                  0.829
Mean absolute error                      1.8376
Root mean squared error                  3.7324
Relative absolute error                 58.6423 %
Root relative squared error             63.7    %
Total Number of Instances               94
```

Decision Stump

```
=== Run information ===

Scheme:       weka.classifiers.trees.DecisionStump
Relation:     prediction_data_frame
Instances:    188
Attributes:   9
            NUMUORS
            NUMUANDS
            TOTOTORS
            TOTOPANDS
            VG
            NLOGIC
            LOC
            ELOC
            FAULTS
Test mode:    user supplied test set:  size unknown (reading incrementally)

=== Classifier model (full training set) ===

Decision Stump

Classifications

NLOGIC <= 14.0 : 1.3806818181818181
NLOGIC > 14.0 : 15.333333333333334
NLOGIC is missing : 2.271276595744681


Time taken to build model: 0 seconds

=== Evaluation on test set ===

Time taken to test model on supplied test set: 0 seconds

=== Summary ===

Correlation coefficient                  0.7111
Mean absolute error                      2.2952
Root mean squared error                  4.1928
Relative absolute error                 73.2439 %
Root relative squared error             71.5571 %
Total Number of Instances               94
```