**EECS 738**

**Final Project Progress Report**

**& Preliminary Results**

**Su Yan**

**Wei Fei**

**Jiahui Wang**

**Introduction**

This project studies on the problem of the high dropout rate on MOOC learning platforms. To predict whether a user will drop a course, we conducted our preliminary experiments on the provided training and testing data set in order to find the suitable classifiers to use. In phase one of project, we show the exploratory analysis of the classification accuracy of many classifiers. The accuracy of our prediction results in this phase may be poor. But the most important thing is our findings and plans for next steps (described in the Future Work section).

**Problem Definition**

In the given training data and testing data, we pad the first row with indices of attributes to make sure Weka can recognize the data. We define the second column as the class attribute which represents whether or not the student will dropout from the course. Also, we replaced 0s and 1s with ‘No’ and ‘Yes’ correspondingly to make it a nominal class. The rest 50 columns are the 50 features. The first problem we have is to determine which classifier to use.

**Experimental Evaluation**

**- Methodology**

To compare which classifier can perform the best in predicting the testing set, an intuitive way is to compare the prediction performance of each classifier in a k-fold cross-validation on the training set. Here we use the accuracy of a classifier (the percentage of correctly classified instances) in a 10-fold cross-validation to measure the performance. (See Appendix 1 for Java code) For simplicity, we use the default parameters for all classifiers. This measurement has been conducted on most classifiers available in the original Weka package. The result is listed below:

|  |  |
| --- | --- |
| Classifier Name | Accuracy (with default parameters) |
| RandomForest | 87.91% |
| LMT (Logistic Model Trees) | 87.90% |
| Logistic | 87.65% |
| JRip | 87.63% |
| MultilayerPerceptron | 87.26% |
| PART | 87.25% |
| REPTree | 87.11% |
| SMO | 87.08% |
| DecisionTable | 86.95% |
| HoeffdingTree | 86.95% |
| SGD | 86.87% |
| OneR | 85.90% |
| J48 | 85.29% |
|  |  |
| NaiveBayes | 84.89% |
| LWL | 84.52% |
| DecisionStump | 84.48% |
| BayesNet | 81.41% |
| VotedPerceptron | 81.36% |
| IBk | 81.33% |
| RandomTree | 81.09% |
| ZeroR | 79.29% |
| NaiveBayesMultinomial | 70.69% |
|  |  |
| KStar | (No result. Computing time too long.) |

From the above table, we observe that the performance of many classifiers are close. Because we haven’t performed feature selection and parameter optimization, we decide that we are interested in all classifiers with their accuracy above 85%. However, currently only three classifiers with highest accuracy, LMT, Logistic and RandomForest are selected and discussed in this report. But it’s possible for us to select different classifiers in the next phase of project.

**- Preliminary Results**

|  |  |  |  |
| --- | --- | --- | --- |
|  | LMT | Logistic | RandomForest |
| Correctly Classified Instances (Accuracy) | 63574 (87.90%) | 63393 (87.65%) | 63581 (87.91%) |
| MCC | 0.235 | 0.592 | 0.601 |
| ROC Area | 0.617 | 0.881 | 0.881 |
| Training error  (Mean Absolute Error) | 0.1837 | 0.189 | 0.1905 |
| Standard deviation (Root Mean Squared Error) | 0.3041 | 0.3068 | 0.3058 |
| Confusion matrixes  (a = 0, b = 1) | a b  8623 6354  2461 54888 | a b  8459 6518  2422 54927 | a b  8708 6269  2525 54824 |

**Discussion on Preliminary Results**

The selected three classifiers with default parameter values show the results of accuracy, MCCs and ROC Areas.

From the second row of the table, it indicates that the Logistic and RandomForest classifiers will be more suitable for the training data set, since the ROC Area of LMT is closer to 0.5, which means random guess; and the value of MCC is lower a lot than other two classifiers.

From the third and forth rows of the table, all three classifiers have similar training errors and standard deviations. The confusion matrices (last row) of three classifiers are also similar to each other. Overall, these three classifiers we selected has no significant differences with their default parameters in terms of their prediction performance.

As required by the project description, a sample prediction on the given testing set has been performed with the Random Forest classifier and its default parameters. The prediction results can be found in Appendix 2.

**Future Work**

Here we have a list of things to explore for the next stage of project.

* **Model Selection**

To reduce the chance of over-fitting, we use 10-fold cross-validation when measuring the accuracy of each model. However, cross-validation does not completely overcome the over-fitting problem in model selection. The cross-validation error does not have a negligible variance, especially if the size of the dataset is small. This means that if we have many hyper-parameters to tune for a model, or many models we can choose, the cross-validation can still be over-fitted as the model can exploit the random variations of the training and testing sets so that we may still end up with a model that performs poorly.

The solution to the above problem is regularization, which penalize the model on over-fitting. Common strategies includes the LASSO and Ridge Regression. As we are only going to use the already compiled classifier packages in Weka, how to select models with built-in regularization becomes crucial to overcome the over-fitting. For example, the Random Forest provides a parameter that governs the number of features that are randomly chosen to grow each tree. In next stage, we will expand our knowledge on each model to find models that can ‘control’ themselves from over-fitting the training data.

* **Feature Normalization**

From our observation, the weights of features in the training set greatly differs from each other. The value of a feature can be as small as 0.0001; and as large as 1.45E+06. These extremely separated numeric values have negative impacts on the model trained from the training set. Feature normalization has two main goals:

* + Avoid extreme large/small feature weights for the purpose of numerical stability of the model.
  + Ensure quick convergence of optimization algorithms.

The way Weka does normalization is using its ‘**Normalize**’ filter. Also, **batch filtering** need to be performed to ensure the testing set will be normalized in the same way as the training set. In next stage, we need to determine if feature normalization is necessary; and whether to normalize each feature separately or normalize all feature together.

* **Parameter Optimization**

The most straightforward way to optimize parameters for a model is to achieve the best performance and avoid over-fitting at the same time. There are two types of parameters. One is the parameters that penalize the model to control over-fitting (e.g. parameter ridge in classifier Logistic). The others are the actual parameters need to be tuned to achieve the best performance on predicting the validation sets.

We plan to use a meta-classifier in Weka named ‘MultiSearch’ to help optimize the parameter. ‘MultiSearch’ is similar to ‘GridSearch’, which is another meta-classifier for exploring all combinations of two parameters and return the combination that performs the best. The advantage of the ‘MultiSearch’ classifier is that it allows the optimization of an arbitrary number of parameters with built-in cross-validation. With the –seach option specified, we can easily test all desired combinations of parameters of a classifier. An example of testing parameter combinations for the ‘Logistic’ classifier is attached in Appendix 3. In this example, we use 10-fold cross-validation to test the accuracy of Logistic by setting the ‘ridge’ parameter from to and the ‘maxIts’ parameter from 1 to 15. The result indicates that Logistic has the highest accuracy with the ‘ridge’ set to and ‘maxIts’ set to 12.

This approach also have problems. As the number of parameters increases, the computation cost of exploiting all desired parameter value combinations also dramatically increases. One example of such cases is the classifier ‘MultilayerPreceptron’. With five parameters that we are interested in, it takes several days to run through all parameter combinations. Thus, optimizing the algorithm of parameter optimization becomes an important job for the next stage of project.

* Attribute Selection

Our first approach for attribute is to use R language scripts to test if a feature is important for a model. We can then check all features one by one automatically. For example, to find all important features for a Linear Regression model, we import the training data and assign all feature columns as matrix and the label column as matrix. Next, we calculate the value of beta and standard deviation. After calculating all intermediate parameters, we can use P-value to test whether a regressor is important or not.

Here is the sample implementation for the p-value test.

*>*

*> // if it is the ath regressor*

*>*

*>*

If p value is greater than 0.05, which means fail to reject null hypothesis, the regressor is unimportant.

The example above is only for the linear regression model. We still need to explore the implementation of p-value test for other models that we are interested in.

The second approach is to determine important attributes through expriments. Weka has three filters for attribute selection: CfsSubsetEval, ClassifierSubsetEval and WrapperSubsetEval. With the help of meta-classifier ‘AttributeSelectedClassifier’, we can directly measure the impact of different feature sets on the prediction accuracy of a certain classifier. Thus, we convert the feature selection problem to another parameter optimization problem for the above three filters. This approach needs more discussion and reasoning to prove its effectiveness before it can be implemented since it could be a misinterpretation of attribute selection.

**Appendix I**

**Java code to test accuracy of selected classifiers**

**using 10-fold cross-validation**

**Appendix II**

**Sample prediction on given testing data set**

**using the Random Forest classifier with default parameters**

(Note: First 4 rows are padded with labeled data which allows Weka to recognize this column as nominal class. Actual predictions start from the 5th row.)

**Appendix III**

**Testing all desired parameter combinations for the Logistic classifier using MultiSearch meta-classifier**