EECS 738 Final Project Report

Su Yan
Wei Fei
Jiahui Wang

Table of Content

I. 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 was poor. In phase two of project, we select three models with good performance from the candidate models in phase one. Then we optimize the parameters roughly based on all features of provided training data and testing data, the accuracy and AUC are not optimal for the effect of meaningless features. Hence, we do the feature selection to reduce the meaningless features and then do the parameter optimization.

II. Problem Statement

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 find three optimal classifiers to use. Second, we do the feature selections to reduce the meaningless features, and then optimize parameters based on the performance of models.

III. Exploratory Analysis of the Dataset

- Training data:
    - Features: 50
    - Samples: 72326
    - Type of features: integer number, floating point number
    - Label: Yes or No (corresponding to 1 or 0)

- Testing data:
    - Features: 50
    - Samples: 48220
    - Type of features: integer number, floating point number
    - Label: Yes or No

In terms of preprocessing, we transformed the class attributes from 0's and 1's to 'No' and 'Yes' correspondingly to let weka recognized as nominal class.

Observe some feature values can be as small as 1.0E-4. Some are as large as 1.0E6. Thus, we perform normalization to remap all features to [0, 100]. This can avoid extreme large/small feature weights for the purpose of numerical stability of the model and ensure quick convergence of optimization algorithms. This normalization is implement with weka.filters.unsupervised.attribute.Normalize

IV. Experiment Design: How do you build your models? How to optimize your models? How do you perform feature selection? And how do you evaluate your model?

1. Model selection

We select our models based on two principles. In the previous stage, we obtained the list of prediction accuracies of most weka's built-in classifiers using their default parameters and 10-fold cross validation (Indicated in following table). We select models with top accuracy and built-in regularizations from the list. They are Logistic, RandomForest and RBFNetwork.

Table 1: Performance of weka classifiers with default parameter

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.)

2. Model parameter optimization (before feature selection)

From preliminary results, we could observe that models with only default parameters are not reliable and good enough. We mainly use the *MultiSearch* (https://github.com/fracpete/multisearch-weka-package) library in this stage. The library allows the optimization of any arbitrary number of parameters and supports both ranged parameter values and array of values. *Multisearch* provides exhaustive search over all given parameter combinations and pick one with best performance.

Figure 1: Example log of optimization by Multisearch

```
=== Initial space - Start ===
Determining best values with 10-fold CV in space:
4-dimensional space:
 - 1. dimension: classifier.classifier.maxIts, min: 0.0, max: 5.0, list: -1,1,5,10,20,50
 - 2. dimension: classifier.classifier.ridge, min: -4.0, max: 4.0, step: 2.0
 - 3. dimension: classifier.classifier.minStdDev, min: 0.1, max: 0.9, step: 0.1
 - 4. dimension: classifier.classifier.numClusters, min: 0.0, max: 5.0, list: 5,10,20,50,100,200

Performance (1, -2, 0.1, 5): 85.5238779968476 (ACC): cached=false
Performance (1, -4, 0.1, 5): 85.5238779968476 (ACC): cached=false
Performance (5, -4, 0.1, 5): 85.85017835909632 (ACC): cached=false
Performance (10, -4, 0.1, 5): 85.85294361640351 (ACC): cached=false
Performance (-1, -4, 0.1, 5): 85.89857036197218 (ACC): cached=false
Performance (-1, -2, 0.1, 5): 85.89857036197218 (ACC): cached=false
Performance (20, -4, 0.1, 5): 85.89718773331859 (ACC): cached=false
Performance (50, -4, 0.1, 5): 85.89857036197218 (ACC): cached=false
Performance (5, -2, 0.1, 5): 85.85017835909632 (ACC): cached=false
Performance (10, -2, 0.1, 5): 85.85294361640351 (ACC): cached=false
Performance (1, 0, 0.1, 5): 85.53079114011558 (ACC): cached=false
Performance (20, -2, 0.1, 5): 85.89718773331859 (ACC): cached=false
Performance (5, 0, 0.1, 5): 85.84603047313553 (ACC): cached=false
Performance (50, -2, 0.1, 5): 85.89857036197218 (ACC): cached=false
Performance (10, 0, 0.1, 5): 85.85017835909632 (ACC): cached=false
```

In general, we use the prediction accuracy as the standard to determine the performance of a model and 10-fold. The result of parameter optimization will be shown in part **V. Experimental Study Result**.

3. Feature selection with another round of parameter optimization

We perform feature selection by wrapping the target classifier (e.g. Logistic) by two meta-classifiers *MultiSearch and AttributeSelectedClassifier*. The *MultiSearch* at the outside layer is still in charge of parameter optimization through 10-fold cross validation. *AttributeSelectedClassifier* at the middle layer perform feature selection with given algorithm and passes the selected feature to target classifier (e.g. Logistic) for performance evaluation.

This layered structure has great performance overhead since we need to perform feature when evaluating each parameter combination. But this sturcture can prevent possible information leaking to classifiers since we always perform feature selection only on the training part of the data after the split in cross validation. The validation part of the data is not touched during feature selection.

4. Iterative feature selection with wrapper method

In previous step, we select important features by evaluating the relations between features and relations between features and class attribute. In other words, we select features using the property of the training dataset. In wrapper method, we have different philosophy. Optimal features should also be different for each specific learning algorithm and training set and should not only corresponds to their correlations. "The optimal features depend on the specific biases and heuristics of the learning algorithm, and hence the wrapper approach naturally fits with this definition." (Ron Kohavi, George H. John (1997). Wrappers for feature subset selection. Artificial Intelligence. 97(1-2):273-324)

With the implementation of package *weka.attributeSelection.WrapperSubsetEval* we come up with the following iterative method:

- Give WrapperSubsetEval a classifier and its current optimal parameters
- Get a set of attributes that maximizes the prediction

performance of give classifer
- Perform parameter optimization on resulting attribute set
- Give WrapperSubsetEval again the same classifier and newly optimized parameters
- Do this until convergence…

This method gives us an opportunity to get model-specific optimal feature set. However, the difficulty is the long computation time. Using the three-layer structure described in Part 3, one iteration of feature selection and parameter optimization takes about 7 days and 16 hours to finish. This exceeds the time limit of a single job on KU ITTC Clusters, which is 168 hours. Unitl now we haven't get any result from the wrapper method.

5. Model Evaluation

From the results generated by program, we mainly use AUC (ROC Area), MCC and Accuracy to determine which model has better performance after cross validation. The result will be shown in next section in table forms. The evaluation and analysis on the results we obtained will be presented in **Section VI**.

V. Experimental Study Results

1. Results of model parameter optimization (without feature selection)

After we selected three models and performed parameter optimizations without feature selection. We obtained the following results:

Table 2: Performance of Models Constructed by Three Classifiers

Models	Accuracy(%)	ROC Area	MCC	Parameters
Logistic	87.6296	0.881	0.592	ridge = 1.0E-4 maxIts = -1
RandomForest	87.9241	0.883	0.604	numTrees = 200 maxdepth = 0 numFeatures = 14
RBFNetwork	86.3382	0.858	0.545	maxIts = -1 ridge = 1.0E2 minStdDev = 0.7 numClusters = 200

2. Results of feature selection with another round of parameter optimization

In this part we performed feature selection for The three classifier we selected. Following eight attribute evaluation algorithms are used in this step.

- Correlation-based Subset Selection (weka.attributeSelection.CfsSubsetEval)
- Pearson's Correlation Coefficient (weka.attributeSelection.CorrelationAttributeEval)
- Gain Ratio (weka.attributeSelection.GainRatioAttributeEval)
- Information Gain (weka.attributeSelection.InfoGainAttributeEval)
- OneR Classifier (weka.attributeSelection.OneRAttributeEval)
- Principal Components Analysis (weka.attributeSelection.PrincipalComponents)
- RELIEF (weka.attributeSelection.ReliefFAttributeEval)
- Symmetrical Uncertainty Analysis (weka.attributeSelection.SymmetricalUncertAttributeEval)

The results in this step are listed in the following table:

(Note: ----- means job canceled by admin of ITTC clusters because it exceeds the time limit)

Table 3: Accuracy Performance of Logistic Classifier after Eight Feature Selection Methods

Attribute Evaluator + Search Method	Accuracy(%)	Attribute Selected	Optimized Parameter
CfsSubsetEval+BestFirst	87.3617	24 31 33 36 37 38 39 40 41 42 45 46 50	maxIts = 6, ridge = 1.0E-10
CfsSubsetEval+GreedyStepwise	87.2065	24 31 33 36 37 38 39 40 41 42 45 46 50	maxIts = 10, ridge = 1.0E1
CorrelationAttributeEval	87.6338	33,31,37,38,32,24,21,39,48,50,45,29,19,26	maxIts = -1, ridge = 1.0E-3

GainRatioAttributeEval	87.6337	45,33,38,31,37,43,42,46,32,24,26,36,34,35	maxIts = -1, ridge = 1.0E-3
InfoGainAttributeEval	87.4791	33,32,31,38,37,27,34,50,36,29,48,35,24,26	maxIts = -1, ridge = 1.0E-3
OneRAttributeEval	87.6338	38,33,32,37,31,34,27,36,35,39,45,50,24,41	maxIts = -1, ridge = 1.0E-3
PrincipalComponents	-------	-------	-------
ReliefFAttributeEval	87.5572	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18	maxIts = -1, ridge = 1.0E-3
SymmetricalUncertAttributeEval	87.6338	33,38,45,31,37,32,43,42,46,24,26,34,36,35,50	maxIts = -1, ridge = 1.0E-3

Table 4: Accuracy Performance of RandomForest Classifier after Eight Feature Selection Methods

Attribute Evaluator + Search Method	Accuracy(%)	Attribute Selected	Optimized Parameter
CfsSubsetEval+BestFirst	92.9347	24 31 33 36 37 38 39 40 41 42 45 46 50	numTrees = 200 maxdepth = 0 numFeatures = 20
CfsSubsetEval+GreedyStepwise	91.6580	24 31 33 36 37 38 39 40 41 42 45 46 50	numTrees = 10 maxdepth = 0 numFeatures = 4
CorrelationAttributeEval	87.9531	33,31,37,38,32,24,21,39,48,50,45,29,19,26	numTrees = 200 maxdepth = 0 numFeatures = 15
GainRatioAttributeEval	87.9144	45,33,38,31,37,43,42,46,32,24,26,36,34,35	numTrees = 200 maxdepth = 0 numFeatures = 14
InfoGainAttributeEval	87.9282	33,32,31,38,37,27,34,50,36,29,48,35,24,26	numTrees = 200 maxdepth = 0 numFeatures = 14
OneRAttributeEval	87.7914	38,33,32,37,31,34,27,36,35,39,45,50,24,41	numTrees = 200 maxdepth = 0 numFeatures = 15
PrincipalComponents	-------	-------	-------
ReliefFAttributeEval	-------	-------	-------
SymmetricalUncertAttributeEval	-------	-------	-------

Table 5: Accuracy Performance of RBFNetwork Classifier after Eight Feature Selection Methods

Attribute Evaluator + Search Method	Accuracy(%)	Attribute Selected	Optimized Parameter
CfsSubsetEval+BestFirst	-------	-------	-------
CfsSubsetEval+GreedyStepwise	-------	-------	-------
CorrelationAttributeEval	86.7544	33,31,37,38,32,24, 21,39,48,50,45,29, 19,26	maxIts = -1, ridge = 1.0E2, minStdDev = 0.5, numClusters = 200
GainRatioAttributeEval	86.7703	45,33,38,31,37,43, 42,46,32,24,26,36, 34,35	maxIts = -1, ridge = 1.0E2, minStdDev = 0.5, numClusters = 200
InfoGainAttributeEval	86.7580	33,32,31,38,37,27, 34,50,36,29,48,35, 24,26	maxIts = -1, ridge = 1.0E2, minStdDev = 0.5, numClusters = 200
OneRAttributeEval	-------	-------	-------
PrincipalComponents	-------	-------	-------
ReliefFAttributeEval	-------	-------	-------
SymmetricalUncertAttributeEval	86.6311	33,38,45,31,37,32, 43,42,46,24,26,34, 36,35,50	maxIts = -1, ridge = 1.0E2, minStdDev = 0.5, numClusters = 200

3. Results of iterative feature selection with wrapper method

We didn’t gain much information in this part. As we mentioned in Section IV, Part 4, the major difficulty is the long computation time. To cope with this, we tried to reduce the parameter combinations and only use 50% of the training data. However, after a week of computation, we only get the results of Logistic. The results are showing below.

(Note: ----- means job canceled by admin of ITTC clusters because it exceeds the time limit)

Table 6: Accuracy Performance of Three Classifiers with Wrapper Feature Selection Method (Incomplete)

Classifier + Attribute Evaluator + Search Method	Input Parameters	Output Accuracy (%)	Output Parameters
Logistic + WrapperSubsetEval + BestFirst	maxIts = -1, ridge = 1.0E-3	87.1803	maxIts = -1, ridge = 1.0
Logistic + WrapperSubsetEval + BestFirst	maxIts = -1, ridge = 1.0E-3	87.1360	maxIts = -1, ridge = 1.0
RandomForest + WrapperSubsetEval + BestFirst	numTrees = 200 maxdepth = 0 numFeatures = 14	-------	-------
RandomForest + WrapperSubsetEval + BestFirst	numTrees = 200 maxdepth = 0 numFeatures = 14	-------	-------
RBFNetwork + WrapperSubsetEval + BestFirst	maxIts = -1, ridge = 1.0E2, minStdDev = 0.5, numClusters = 200	-------	-------
RBFNetwork + WrapperSubsetEval + BestFirst	maxIts = -1, ridge = 1.0E2, minStdDev = 0.5, numClusters = 200	-------	-------

VI. Results Analysis and Discussion

After parameter optimization, the result become more reliable. For example, the accuracy of Logistic with default ridge parameters (1.0E-8) is 87.65%. The accuracy after parameter optimization is 87.63% with a ridge of 1.0E-4. Even though the accuracy after parameter optimization is a little lower than the classifier used default parameter, the model becomes more reliable since there's less chance of over-fitting. For the RandomForest classifier, the accuracy improves a little (Before: 87.91%, After: 87.9% ~ 92.9%).

After feature selection, we observe the performance increase of the RandomForest, especially with CfsSubsetEval feature selection method. Logistic and RBFNetwork do not has significant performance increase. Same as the process of parameter optimization, the models become more reliable since many irrelevant features have been filtered out.

By examine the Table 2, 3, 4 and 5 from the previous section, we observe that RandomForest has a better performance over the other two classifiers. So we decided to use RandomForest classifer and CfsSubsetEval feature selection method to construct our final model and make predictions on the given test data.

VII. Final Model Construction

To construct the final model, we further measure the performance of different parameter combinations on the attribute-selected training data using CfsSubsetEval feature selection method. To obtain more precise results, we perform 10-fold cross validation for twenty times for each parameter combination. Each time we randomly select the training and testing partition for the cross validation by using different random seeds. Then we calculate the mean and the variance of MCC and AUC of each parameter combination and compare which parameter set performs the best.

RandomForest has three parameters that we can optimize: numTrees (-I), maxDepth (-depth) and numFeatures (-K). From our previous experience, we observe that the *maxDepth* parameter need to be set to 0 (unlimited depth) to obtain the optimal performance. So here we do not optimize this parameter again.

Table 7: Mean and Variance of MCC and AUC for each parameter combination of RandomForest

Parameter Combination	mean (MCC)	variance (MCC)	mean (AUC)	variance (AUC)
I = 50, K = 10	0.6008	0.0015	0.8786	0.0005
I = 50, K = 12	0.6013	0.0011	0.8779	0.0007
I = 50, K = 14	0.6012	0.0014	0.8780	0.0007
I = 50, K = 16	0.6009	0.0013	0.8776	0.0009
I = 50, K = 18	0.6006	0.0017	0.8782	0.0007
I = 50, K = 20	0.6008	0.0019	0.8776	0.0007
I = 100, K = 10	0.6033	0.0017	0.8815	0.0005
I = 100, K = 12	0.6035	0.0013	0.8812	0.0005
I = 100, K = 14	0.6027	0.0011	0.8812	0.0006
I = 100, K = 16	0.6036	0.0014	0.8806	0.0006
I = 100, K = 18	0.6028	0.0017	0.8806	0.0005
I = 100, K = 20	0.6030	0.0011	0.8808	0.0007
I = 150, K = 10	0.6040	0.0010	0.8826	0.0005
I = 150, K = 12	0.6042	0.0012	0.8822	0.0008
I = 150, K = 14	0.6039	0.0009	0.8824	0.0005
I = 150, K = 16	0.6039	0.0012	0.8818	0.0006
I = 150, K = 18	0.6033	0.0011	0.8817	0.0006
I = 150, K = 20	0.6039	0.0012	0.8818	0.0004
I = 200, K = 10	0.6044	0.0010	0.8830	0.0004
I = 200, K = 12	0.6043	0.0011	0.8829	0.0005
I = 200, K = 14	0.6042	0.0012	0.8828	0.0005
I = 200, K = 16	0.6044	0.0012	0.8825	0.0005
I = 200, K = 18	0.6039	0.0012	0.8826	0.0005
I = 200, K = 20	0.6042	0.0013	0.8822	0.0006

From the above table, we observe that the variance of MCC and AUC are both very small. Thus we can determine the performance of parameter combinations by only look at the means of MCC and AUC.

Figure 2: Mean of MCC for each parameter combination of RandomForest

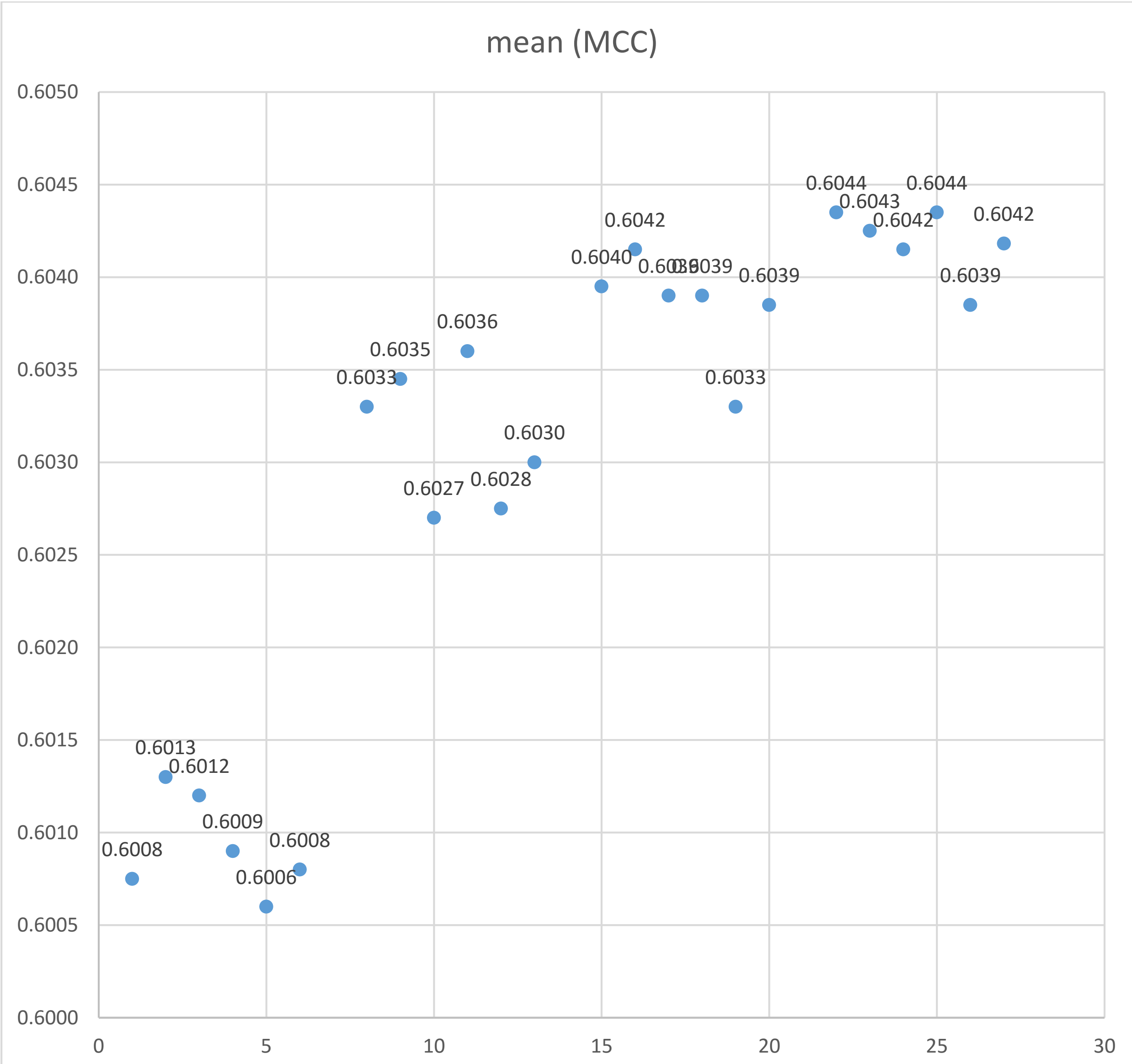
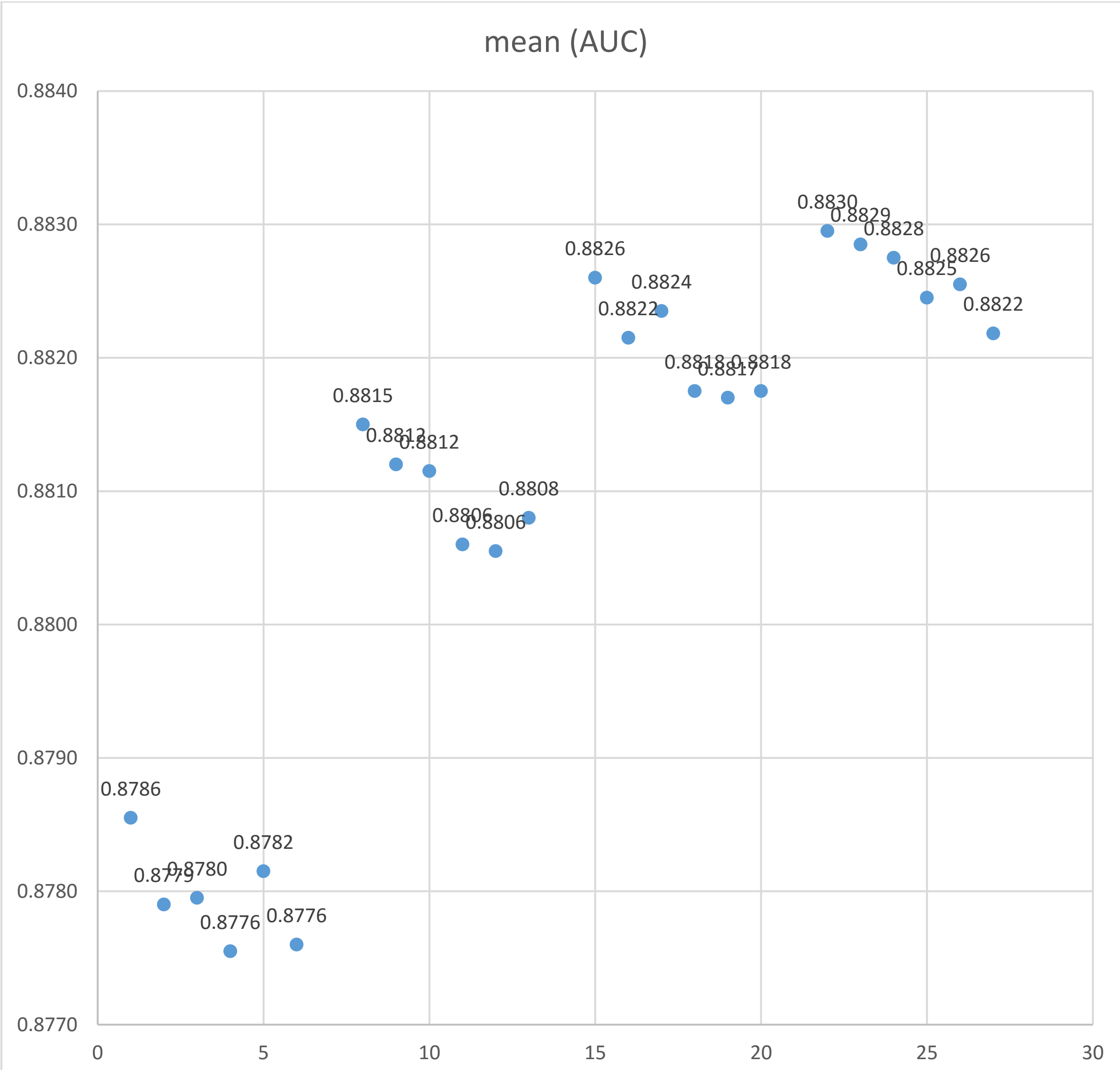


Figure 3: Mean of AUC for each parameter combination of RandomForest



From the above two graphs, we notice that when numTrees = 200, numFeatures = 10, the MCC and AUC value both reaches the peak of each graph.

Thus, we will build our final model with following specifications

- Feature selection method: CfsSubsetEval + BestFirst
- Classifier: weka.classifiers.trees.RandomForest
- Parameters: numTrees = 200, numFeatures = 10, maxDepth = 0

VII. Predicted Result on Test Data

Please see the file "EECS738_2736271_test.txt".

VIII. Code Descriptions

We submitted five Java files in total. Each of them has different purpose. To compile and run these files, three external jar libraries need to be in the Java Classpath:

- Jar library of multisearch package (https://github.com/fracpete/multisearch-weka-package/releases)
- Weka.jar (from weka version 3.7.11)
- Jar library of RBFNetwork classifier (Download through weka package manager)

Here we have a short description for each of the five files.

- PreliminaryTest.java

    This file checks the performance of most weka's built-in classifiers using their default parameters. The results of this file are recorded in Table 1.

- ParaOptimize.java

    This file is aimed to perform parameter optimization without feature selection. The file is designed to be run on ITTC clusters in parallel. To optimize parameters for a specific classifier, change the index on line 176 to the corresponding index in the parameters array. The results of this file are recorded in Table 2.

- FeatureSelection.java

    This file is similar to ParaOptimize.java with an extra layer to perform feature selection. The results of this file are recorded in Table 3, 4 and 5.

- WrapperSubset.java

This file implements the wrapper method feature selection described in Section IV, Part 4. The results of this file are recorded in Table 6.

- MCC_AUC.java

  This file performs another round of parameter optimization for RandomForest over attribute-selected training data (described in Section VII). The results are recorded in table 7.