Salary Prediction Analysis Report

Yaxin Huang huang.yax@husky.neu.edu

Jalpan Randeri randeri.j@husky.neu.edu

Rohit Vallu vallu.r@husky.neu.edu

Nadim Mahesaniya mahesaniya.n@husky.neu.edu

1. Introduction

This report is for the data mining project of labeling the census income dataset. The project aims at identifying whether an incoming tuple should be classified as "<=50K" income range or ">50K". The project follows the below procedure to explore on this topic:

- Data Inspection, described in section 2.
- Data Preprocessing, described in section 3.
- Single Classifier Cross Validation Test, described in section 4.
- Ensemble Learning, described in section 5.
- Evaluation on the Testing Dataset, described in section 6.
- Conclusion, which is the section 7.

At last our project achieves the testing accuracy of 0.83196, by using C4.5 with AdaBoost, and the C4.5 parameters are as the following:

- Pruning confidence = 0.35
- Turn off subtree-raising
- Apply Laplace smoothing

2. Data Overview

The census income training data contains 14 features and the last column is the label, as described in the description file. By running the Rscript("AnalyzeData.R") we can find out some characteristics of the training data as listed below:

- There are 32563 training instances.
- There are 2399 row contains the missing values.
- The features containing missing values are the 2, 7, 14 columns, and those features

all contain nominal values.

- Another issue we analysed was the given dataset contains nearly 3 : 1 class imbalance.

Therefore, during the data preprocessing, we need to apply missing value imputation and balancing the data before we build any classifier from the training set.

3. Data Preprocessing

3.1 Missing Value Imputation

Since some of the classification algorithm will discard the features containing missing values, such as C4.5 Decision Tree, we definitely need to replace the missing values. What we did is to replace them with the mode of the corresponding feature, because the features with missing values are all nominal. Here we utilized the `ReplaceMissingValues` provided by Weka[3] to fill in the mode.

3.2 Oversampling the Imbalanced Training Dataset

There are far more instances of lower income label than the ones with higher income label. If we directly use the imbalanced training data to train a classifier, they result may not be satisfying when we test the classifier, because the standard learning algorithms tend to bias towards the majority class. As the classifier attempts to achieve higher overall accuracy, what it will do might be classifying all incoming testing instance as having lower salary, which is an extremely bad result because actually the classifier is learning nothing.

Our goal in the next step is to pick several classifiers who perform better, in cross validation, than other classifiers or same classifiers with different parameters, and a balanced training dataset is required for the next step.

The oversampling technique is used here. The typical procedure for oversampling, which is called random oversampling is to select the minority instances multiple times to create a balanced dataset. But what we did here is different from the random oversampling. We took advantage of the `Resample` provided by Weka, and the following table shows the

options of interest:

Table 2-1 Relevant Options in Resample by Weka

Option in Resample	Description
-Z	The size of the output dataset, as a percentage of the input dataset (default 100)
-B	Bias factor towards uniform class distribution. 0 = distribution in input data -- 1 = uniform distribution. (default 0)

By setting these options correctly we can get a balanced dataset. We set the -B option as 1 since the uniform distribution is desired. And the percentage -Z is calculated in this way:

```
size_data = size of the imbalanced training dataset.
num_lower = number of instances with lower income (the majority class)
num_higher = number of instances with higher income (the minority class)
-Z = [(size_data + num_higher - num_lower) / size_data] * 100
```

By using the above option in `Resample` we achieve the effect of oversampling.

4. Testing on Different Classifiers with Cross Validation

This section describes the next step in our experiment after the data preprocessing. In order to found out the classifiers and the corresponding parameters which may have a good performance when Ensemble Learning is applied, we firstly ran through the cross validation for C4.5 Decision Tree, KNN Classification, Logistic Regression, Naive Bayes Classification and Naive Bayes Tree, and meanwhile adjusted the parameters of each classifier and compared the error rates. The following subsections contains the descriptions of the algorithms we used, the parameters of interest and the results got from cross validation.

4.1 C4.5 Decision Tree

Algorithm Description:

C4.5 Decision Tree is a classification algorithm very similar to ID3(the version that splits a node based on information gain) we learned in class. The major difference of C4.5 and ID3 are listed as below[4]:

- The information gain measure is biased towards the attributes having the largest number of values, because such attributes will produce "purest" subtree and thus the information gain will be highest. C4.5 uses an extension of information gain, gain ratio[2], to split the nodes. It considers the total number of instances split to a node with respect to the total number of instances in the whole dataset.
- C4.5 can handle continuous-valued features by setting a threshold.
- C4.5 will simply drop the attributes with missing values. Thus, missing value imputation before apply C4.5 is necessary.
- Post-pruning will be done after the decision tree is built.
- C4.5 allows different class labels with different costs.

Parameters Used:

In Weka, a set of parameters can be passed to the algorithm and we picked a subset of them to test:

Table 4-1 Parameters Used in C4.5 Decision Tree Testing

Options in C4.5	Description
-C	This is the pruning confidence. Set confidence threshold for pruning. (default 0.25)
-R	Use reduced error pruning. If this option is set, then the pruning confidence should not be set.
-S	Don't perform subtree raising.
-A	Laplace smoothing for predicted probabilities.

- The pruning confidence affects the size of the tree. The smaller pruning confidence leads to more pruning and a smaller tree. Thus, this is a parameter to control overfitting.
- The reduced error pruning is what we learned in class. It first split the original training data into training set and validation set. The tree is built on the training set and the validation set is used to post-prune the tree. If a node needs to be pruned, it will be replaced by a leaf node.
- In C4.5, if the default pruning method is used, instead of subtree replaced by a leaf node, it does the subtree raising. In this case, a node may be moved upwards towards the root of the tree, replacing other nodes along the way. Subtree raising often has a negligible effect on decision tree models. There is often no clear way to predict the utility of the option, though it may be advisable to try turning it off if the induction process is taking a long time. This is due to the fact that subtree raising can be somewhat computationally complex. Therefore, in our experiment, we both tested the case using subtree raising and without subtree raising.

Results:

Figure 4-1 displays the error rate of different option settings. The error rate when using reduced error pruning instead of the default pessimistic pruning is much higher (more than 0.11) than the other settings. And when the pruning confidence increases, the tree size also increases and therefore the error rate decrease. However this might lead to overfitting. In addition, when we turn off the subtree raising in pessimistic pruning (-S), generally speaking, the error rates are smaller than using the subtree raising.

Therefore, in our next step, we tested the performance of using C4.5 with the following settings in Boosting and Bagging:

- [-C 0.25 -S -A], [-C 0.35 -S -A] and [-C 0.45 -S -A] in Boosting
- [-C 0.35 -S -A] in Bagging

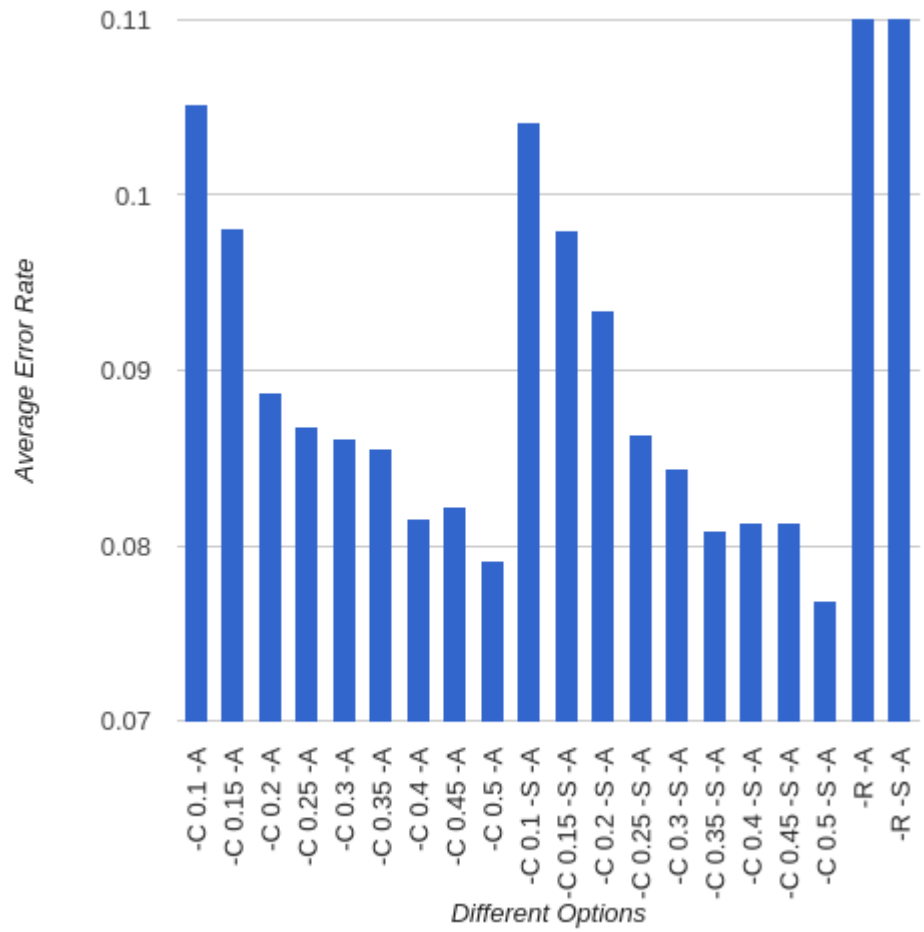

Figure 4-1 C4.5 Using Different Parameters

Although the cross validation average error rate goes down as the pruning confidence increase, in order to avoid overfitting we picked the lower C to run our next experiments.

4.2 KNN Classification

Algorithm Description:

k-Nearest Neighbors[5][6] is a algorithm used for classification. The input consists of the k closest training examples in the feature space. The output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors. KNN classification is Instance based learning in which no model is built and it is lazy learning algorithm in which any decision is delayed until new query is encountered.

Parameters Used: value of K

We need to pick the value of K before running the algorithm to label the test set. This is one of the shortcoming of KNN algorithm. To pick a better K, we evaluated the performance by computing the error rate of KNN classifiers with K starting from 1 to 20.

Results:

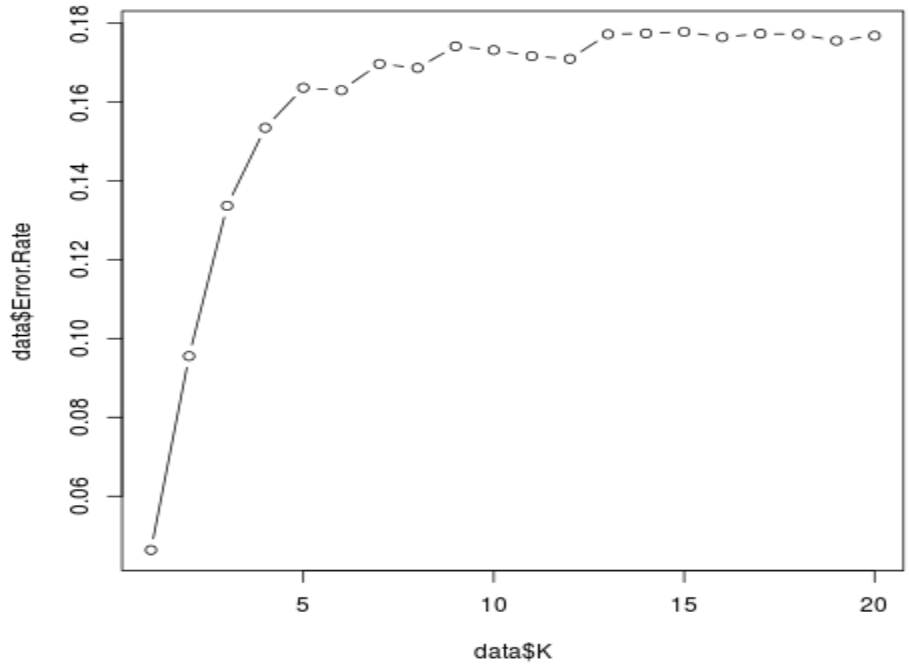

Figure 4-2 Error Rate as K from 1 to 20

Figure 4-2 shows that as the K increases the error rate increases. And in our test runs, the higher K consumes more time to output the results. Thus, in our ensemble learning, we used K = 1 to train the KNN classifier.

4.3 Logistic Regression

Description:

Logistic Regression[7][8] is a is a direct probability model. Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables, which are usually (but not necessarily) continuous, by estimating probabilities. Logistic regression is a discriminative method that learns

P(y | x) directly from the training data without assuming any particular form of distribution for P(x | y) with assumption that the training instances are mutually independent.

Parameters Used:

The SimpleLogistic implemented by Weka provides a series of options, but we did not find the options of interest. Thus, in our experiment we used the default options.

Results:

We ran a single 10-fold cross validation for Logistic Regression, and it took a long time to run, about 5 times as long as the C4.5 took. The average error rate is 0.177184466. The performance was the worst among all other classifiers. Therefore we only used Logistic Regression in Bagging.

4.4 Naive Bayes Classification

Algorithm Description:

Naive Bayes Classification[9][10] is a probabilistic classification based on applying Bayes' theorem which assume that each attribute is independent of any other attributes given the class label. In simple terms, a naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature.

Parameters Used:

The only option we found interesting in the NaiveBayes provided by Weka is the "-D" option. If this option is set, the continuous features will be supervised discretized into groups. Otherwise the algorithm will assume the continuous features have Guassian Distribution by default.

Results:

Table 4-2 Naive Bayes Average Error Rate with Different Options

Option	Average Error Rate
Using -D	0.1671723301
Without -D	0.2431027508

As shown in table 4-2, by using the supervised discretization, the error rate dropped approximately 0.076. This might probably because the numeric features in our dataset might not be normally distributed. And for the numeric features we have in the dataset, such as age, hours-per-week, capital-gain etc., probably discretization is a sensible choice since our intuition is that this values can be split into groups. Therefore, in our next step, we used the supervised discretization option for Naive Bayes Classification.

4.5 Naive Bayes Decision Tree

Description:

Naive Bayes Decision Tree[1] is a hybrid approach that attempts to utilize the advantages of both decision trees (i.e., segmentation) and Naive-Bayes (evidence accumulation from multiple attributes). The algorithm is similar to the classical recursive partitioning schemes, except that the leaf nodes created are Naive-Bayes categorizers instead of nodes predicting a single class. A threshold for continuous attributes is chosen using the standard entropy minimization technique, as is done for decision-trees. The utility of a node is computed by discretizing the data and computing the 5- fold cross-validation accuracy estimate of using NaiveBayes at the node.

Results:

There is no options provided by the NBTree implemented by Weka. We ran the algorithm using 10-fold cross validation, and the average error rate was 0.1664037217. We incorporated the NBTree in our next step, Ensemble Learning, to see if the performance can be improved.

4.6 Summary of Classifier Cross Validation Test

Table 4-3 Summary of Performance of Classifiers Will be Used in Ensemble Learning

Classifier	Options	Average Error Rate	Speed
C4.5	-C 0.25 -S -A	0.0863470874	Fast
	-C 0.35 -S -A	0.0808050162	
	-C 0.45 -S -A	0.0813511327	
KNN	K=1	0.0463592233	Medium
Logistic	N/A	0.177184466	Slow
Naive Bayes	-D	0.1671723301	Fast
NBTree	N/A	0.1664037217	Medium

Table 4-3 displays the classifiers and corresponding options we will use in the next step. The best performance is achieved by KNN when K=1. However we need to question if the C4.5 and KNN is overfitting the traning data. In the next step we will use all classifiers for Bagging, and C4.5, KNN, Naive Bayes and NBTree for Boosting.

5. Ensemble Learning

In the last section, we discussed the performance of different classifiers using different options. This section describes the Ensemble Learning we experimented and the analysis we made. The positive class is "<=50K" and the negative one is ">50K".

5.1 Boosting:

Boosting is a meta-learning ensemble algorithm which converts weak learners into stronger ones. In this approach instead of sampling we weigh the examples every iteration for the given number of learners. The samples used for each step are not draw the same way for the same population rather incorrectly predicted cases are given incremented weights during next step. The final classifier is the weighted vote of all the learners.

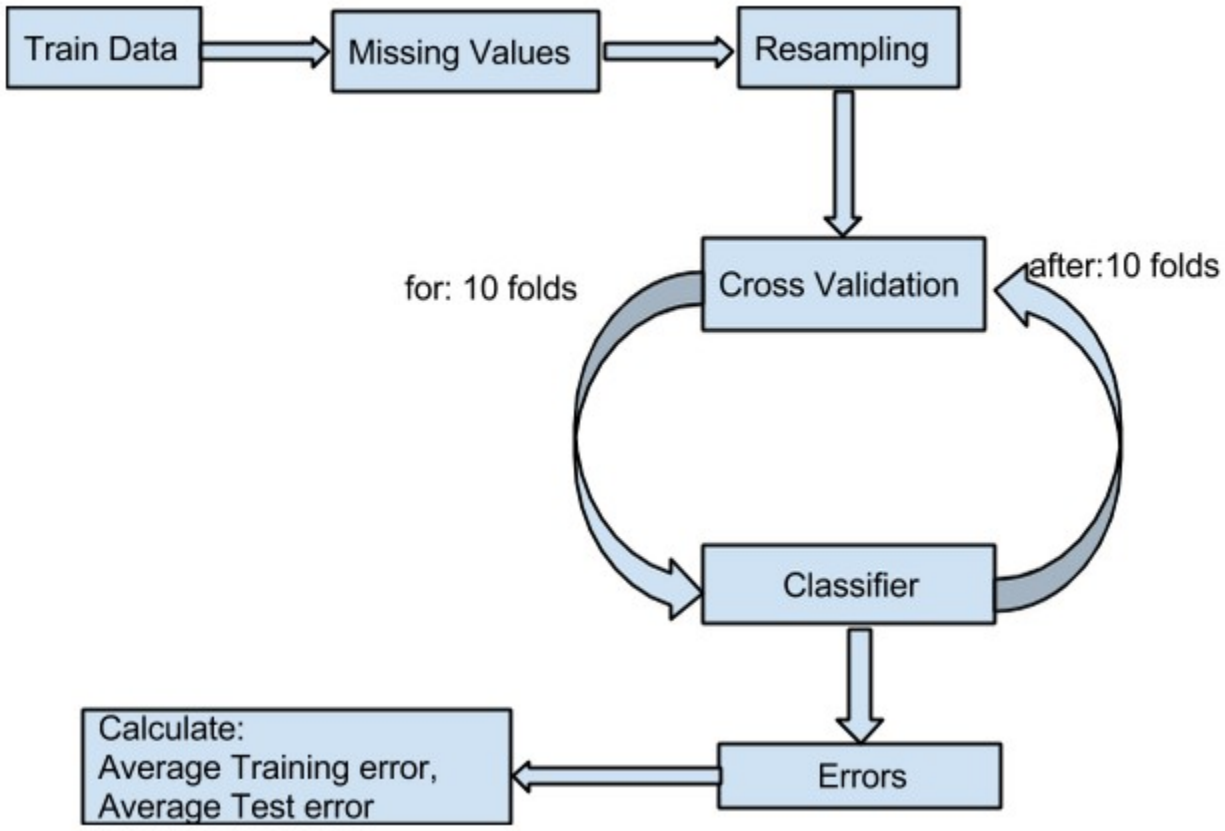

Figure 5-1 The Boosting Procedure of Our Program

Figure 5-1 illustrates how we did boosting in our program. The program accepts the training data in the ARFF form, and does the missing value imputation and resampling mentioned in section 3. After it is processed, the training data is divided into training set and testing set in each fold of the 10-fold cross validation. Within each fold, the AdaBoostM1 model provided by Weka is built upon a base learner, which is one of C4.5, KNN, Naive Bayes and NBTree. At last the program outputs the average training error, TPR, TNR, and average testing error, TPR, TNR.

Table 5-1 displays the AdaBoost results for different base learners and different iterations. The larger the iteration number, the more learner will be trained in the boosting process, and the results show that the larger the iteration number, the lower the training error rate. However, too many iterations might lead to overfitting. Although it is not very obvious, but as an example, the error rate of C4.5 has a slight increasing trend

as the iteration number increases.

Table 5-1 AdaBoost Performance upon Different Base Learners and Different Iterations

Model	Options	Boosting Iteration	Train ER	Train TPR	Train TNR	Test ER	Test TPR	Test TNR
C4.5	-C 0.25	10	0.0	1.0	1.0	0.03428	0.94759	0.98421
	-C 0.35	10	2.022E-5	0.99995	1.0	0.03480	0.94727	0.98325
	-C 0.45	10	2.022E-5	0.99995	1.0	0.03503	0.94565	0.98401
	-C 0.25	15	0.0	1.0	1.0	0.03284	0.95083	0.98345
	-C 0.35	15	0.0	1.0	1.0	0.03430	0.94991	0.98154
	-C 0.45	15	2.022E-5	0.99996	0.99999	0.03517	0.94658	0.98319
	-C 0.25	20	4.045E-5	0.99993	0.99998	0.03580	0.94423	0.98388
	-C 0.35	20	1.797E-5	1.0	0.99996	0.03491	0.94549	0.98455
	-C 0.45	20	0.0	1.0	1.0	0.03547	0.94691	0.98215
KNN	-K 1	10	0.0	1.0	1.0	0.04389	0.93291	0.97971
NB	-D	10	0.15831	0.80588	0.87712	0.15993	0.80364	0.87613
		15	0.16010	0.81393	0.86617	0.16140	0.81264	0.86485
		20	0.15737	0.80930	0.87573	0.15999	0.80623	0.87355
NBTree	N/A	10	5.169E-5	0.99995	0.99994	0.03869	0.93925	0.98299
		15	2.022E-5	0.99995	1.0	0.03727	0.94227	0.98314
		20	2.022E-5	0.99995	1.0	0.03695	0.94452	0.98144

Because the KNN ran too slow as the number of K or the number of iterations increased, we only tested the K=1 and iteration number = 10 KNN classifier with boosting, taking the running time into consideration. Its training results looks perfect but this is a sign of overfitting.

Naive Bayes with AdaBoost drops the error rate approximately 0.007, but compared with other classifiers it is not the best choice. The NBTree's performance was dramatically increased by approximately 0.13. The error rate of C4.5 with AdaBoost, in comparison, was about 0.05 lower than the plain C4.5 on average.

Now we compare the C4.5 with NBTree, with the iteration number equal to 10 because we don't want to overfit the data. With boosting they both had an increase in performance

and the NBTree increased more, but as table 5-2 shows, the C4.5 did slightly better in testing performance. Therefore, if at last the Boosting outperforms Bagging, the C4.5 with AdaBoost will be chosen and the options are the one shown in table 5-2.

Table 5-2 Comparison between C4.5 and NBTree when Iteration Number = 10

Model	Options	Boosting Iteration	Train ER	Train TPR	Train TNR	Test ER	Test TPR	Test TNR
C4.5	-C 0.25	10	0.0	1.0	1.0	0.03428	0.94759	0.98421
NBTree	N/A	10	5.169E-5	0.99995	0.99994	0.03869	0.93925	0.98299

5.2 Bagging :

Bagging is also an ensemble meta-learning algorithm which is used for improving accuracy and stability of algorithms used in regression. In the bagging or bootstrap aggregation, re-sampling of the training data is done repeatedly and in each repetition, random subsets are chosen. In bagging the final classifier is the weighted vote of all the classifiers. Using the each bootstrap samples the classifier is trained and for the classification the majority vote of the classification result was taken into consideration.

Our bagging procedure, as figure 5-2 shows, did the same data preprocessing as what described in section 3. After the training data is cleaned, it will be put into the 10-fold cross validation. Each fold a part of the data is set aside to be the testing set. And the remaining data will be used to generate several bags of data. Each bag contains almost the same number of tuples and the class label within the bag is balanced. The number of bags will be generated is decided by how many classifiers we are using.

The program decides an incoming tuple's label by majority vote of the 3 or 5 classifiers we used. We expect that because different classifiers are used, they might focus on different aspect of the data and by doing majority vote the result can be improved. Finally after 10 folds, the training and testing performance will be output, and the results can be found in table 5-3.

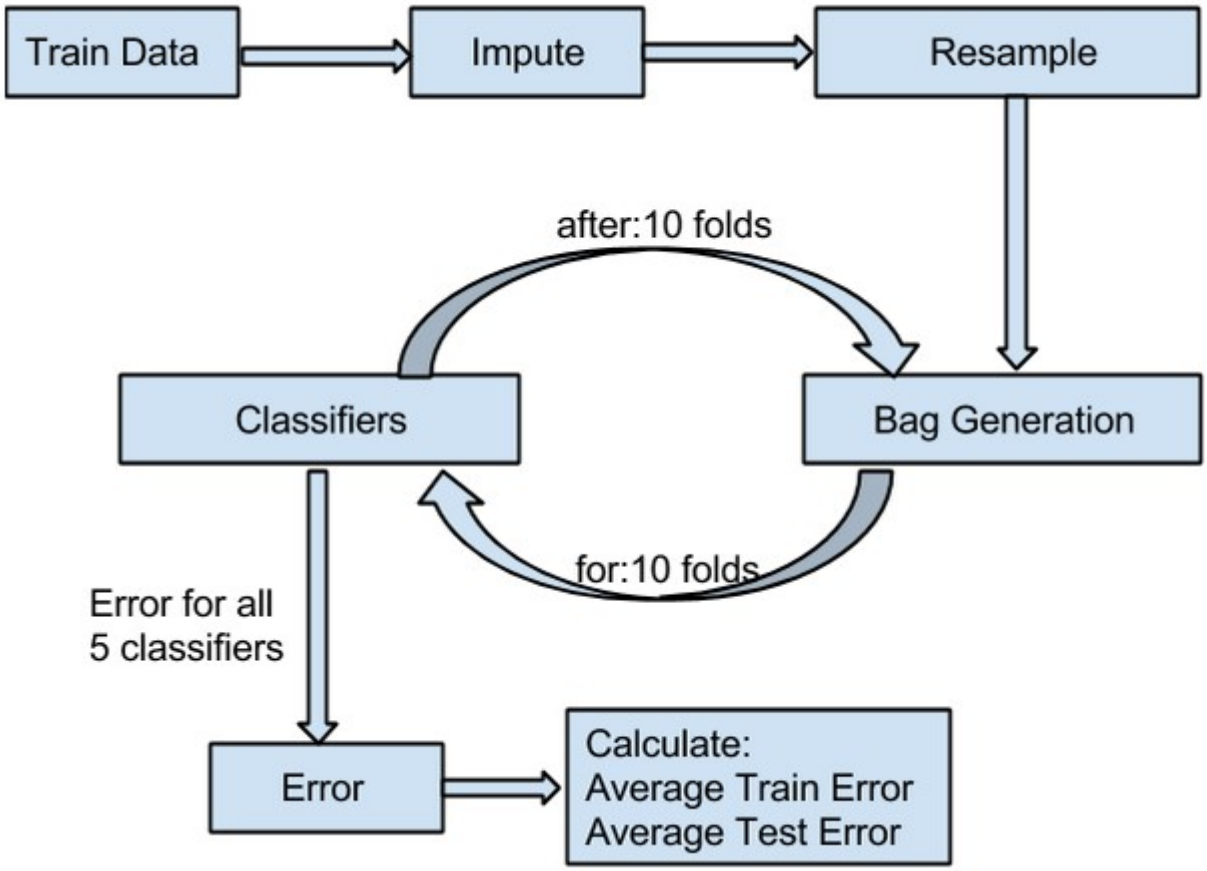

Figure 5-2 The Bagging Procedure of Our Program

Table 5-3 Bagging Results

Number of Bags	Base Learner	Test ER	Test TPR	Test TNR
3	C4.5 -C 0.35 -S -A KNN -K 1 Naive Bayes -D	0.16282	0.78	0.89
5	C4.5 -C 0.35 -S -A KNN -K 1 Naive Bayes -D Logistic Regression NBTree	0.15675	0.80	0.89

The result is not as good as the ones produced by Boosting. Through our debugging process, we found that KNN did better than other classifier when the true label of a testing tuple is negative (>50K). For other classifiers they often made mistakes in this case. However the problem lies in the TPR. It shows that the bagging classifier is not

doing a very good job in identifying the positive instances. This is weird at first glance, and our analysis is that maybe for an incoming positive tuple, some of the classifiers can label it as positive but unfortunately the majority thinks it is negative, and this case happens from time to time, leading to the low testing TPR in bagging.

6. Evaluation of the Testing Dataset

In comparison, the Boosting method increased the performance more and the Bagging decreased the performance. Thus, at last we picked the C4.5 with AdaBoost to apply on the testing data "census-income.test" and here is the output we got:

Table 5-4 C4.5 with AdaBoost on "census-income.test" dataset

Train ER	Train TPR	Train TNR	Test ER	Test TPR	Test TNR
2.0226E-5	0.99995	1.0	0.16804	0.87382	0.69656

From the table above we can calcuate:
Training Accuracy = 0.99997
Testing Accuracy = 0.83196

We also did an experiment on the testing set using default C4.5 setting in Weka and without any missing data imputation, imbalanced data treatment or ensemble learning, the testing accuracy is about 85.8485. Thus, the result we got above from Boosting is worse than the case when pure C4.5 is applied. We analyzed the reason and it might be the overfitting problem of the AdaBoost algorithm.

7. Conclusion

As the section 6 states, the testing accuracy got from all the data preprocessing and ensemble learning is not as good as using the pure classifier. The reason might be the overfitting problem, since all of them have a good training performance (error rate, TPR and TNR). Although the result is not ideal, we did learn a lot through the whole project exploration.

8. References

[1] Kohavi, Ron. 1996. Scaling up the Accuracy of Naive-Bayes Classifiers: Decision-Tree Hybrid.

[2] Han, Jiawei. Kamber, M. Pei, J. Data Mining Concepts and Techniques, 3rd Edition: 340-341.

[3] Weka 3: The Data Mining Tool in Java. http://www.cs.waikato.ac.nz/ml/weka/

[4] Wikipedia.org: C4.5 Algorithm. http://en.wikipedia.org/wiki/C4.5_algorithm

[5] Wikipedia.org: KNN algorithm. http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[6] Yijun Zhao. CS6220 NEU Teaching Material: KNN algorithm. http://www.ccs.neu.edu/course/cs6220sp15/KNN_nopause.pdf

[7] Yijun Zhao. CS6220 NEU Teaching Material: Logsitic Regression. http://www.ccs.neu.edu/course/cs6220sp15/Logistic_no_pause.pdf

[8] Wikipedia.org: Logistic Regression. http://en.wikipedia.org/wiki/Logistic_regression

[9] Wikipedia.org: Naive Bayes. http://en.wikipedia.org/wiki/Naive_Bayes_classifier

[10] Yijun Zhao. CS6220 NEU Teaching Material: Naive Bayes. http://www.ccs.neu.edu/course/cs6220sp15/nb_no_pause.pdf