Assignment 4 – Fall 2018

Created by Shi, Yu, last modified on Nov 26, 2018

Assignment 4 (Distributed Thu. Nov. 8, 2018, Due 11:59 PM Tue. Dec. 11, 2018)

Before Dive in

- Programming assignments take more time, and it counts more in your final grade, so please start early.
- This is an individual assignment. You can discuss this assignment in Piazza, including the **performance** your implementation can achieve on the provided dataset, but please do not work together or share **codes**/**parameters**.
- Related libraries or programs can be found online, but you are prohibited to use these resources directly. The purpose of this programming assignment is to learn how to build a classification framework step by step.
- You will need to make decisions about parameter values when implementing these algorithms (e.g., depth of decision tree, number of trees in random forest, etc.). Please do not ask or share parameter settings on Piazza. You need to tune the parameters and find the best setting for each dataset yourself. You need to briefly describe why and how you find these parameter settings in your final report.
- You can use C/C++ or Java or Python2/3 as your programming language (No, you cannot use Matlab or C#. And no, Scala is not Java). Your programs should be able to compile correctly in EWS Linux environment. This assignment will be graded automatically using an auto-grader so please make sure you follow all the input and output definitions strictly. The auto-grader will try to compile your source code and check the output of your programs. The auto-grader also has plagiarism detection feature, so please do not copy other people's programs or use a modified version of existing resources. A more detailed project organization guidance can be found at the end of the assignment.
- Late policy:
  10% off for one day (Dec. 12th, 11:59 PM),
  20% off for two days (Dec. 13th, 11:59 PM),
  40% off for three days (Dec. 14th, 11:59 PM).
- A section titled Frequently Asked Questions can be found at the end of the assignment, and we will keep updating it when more questions emerge on Piazza.

The Big Picture

In this assignment, we will build a general purpose classification framework from scratch. This framework contains two classification methods: a basic method and an ensemble method. More specifically, decision tree and random forest. Given certain training dataset following a specific data format, your classification framework should be able to generate a classifier and use this classifier to assign labels to unseen test data instances.

We will then test our classification framework with several datasets using both the basic method and the ensemble method, and evaluate the quality of the classifiers with different metrics.

In order to build this classification framework, we need to finish four steps as follows.

- Step 1: Read in training dataset and test dataset, and store them in memory.
- Step 2: Implement a basic classification method, which includes both training process and test process. Given a training dataset, the classification method should be able to construct a classifier correctly and efficiently. Given a test dataset, the classification method should be able to predict the label of the unseen test instances with the aforementioned classifier.
- Step 3: Implement an ensemble method using the basic classification method in Step 2. The ensemble classification method should also be able to train a classifier and predict labels for unseen data instances.
- Step 4: Test both the basic classification method and the ensemble method with different datasets. Evaluate their performances and write a report.

We introduced many classification methods in the lecture. However, in this assignment, we pick the following basic classifier and ensemble classifier.

- Decision Tree (gini-index) as the basic method, Random Forest as the ensemble version of the classification method.

Note 1: Other classifiers can also fit into this basic-ensemble framework. For example, Naive Bayes as the basic method, and AdaBoost as ensemble method. We do not cover them in this assignment.

Note 2: Different Random Forest methods can be found online (we introduced two in the lecture notes). You can choose either one but Forest-RI might be easier to implement.

Note 3: Many off-the-shelf ML tools can be found online for our classification tasks, e.g., sklearn. Since the purpose of this assignment is to go through the basic classifier and ensemble classifier step by step, it is not allowed to use any third-party library in your code.

Note 4: The classification tasks we are handling here can be either **binary classification** or **multi-class classification.** Please make your code robust enough to handle both cases.

Step 1: Data I/O and Data Format (0 pts)

The classification framework should be able to parse training and test files, load these datasets into memory. We use the popular LIBSVM format in this assignment.

The format of

<label> <index1>:<value1> <index2>:<value2> ...
.....

.....

Each line contains an instance and is ended by a '\n' character. <label> is an integer indicating the class label. The pair <index>:<value> gives a feature (attribute) value: <index> is a non-negative integer and <value> is a number (we only consider categorical attributes in this assignment). Note that one attribute may have more than 2 possible values, meaning it is a multi-value categorical attribute.

You need to be careful about not using label information as an attribute when you build the classifier, in which case you can get 100% precision easily, but you will get 0 pts for Step 2 and Step 3.

Step 2: Implement the Basic Classification Method (40 pts)

In this step, you will implement the basic classification method, i.e., decision tree with **gini-index** as the attribute selection measure (note: the gini-index score is computed by considering every possible categorical value of the feature as a branch). Your classification method should contain two steps, training (build a classifier) and test (assign labels to unseen data instances). Many existing classification or prediction packages have the feature to save the learned classifier as a file so that users can load the classifier into the framework later and apply the same classifier on different test datasets asynchronously. However, in this assignment, we are going to skip this save-classifiers-as-files feature, and assume that the training and test processes happen in the same run.
Before you begin with your implementation, please first read the Project Organization and Submission section at the end of the assignment to get a general idea on how to organize your project and how to name your programs so the auto-grading system can compile and execute your programs correctly.
The classification method you implemented should be able to take both training file and test file as input parameters from the command line, use training dataset to build a classifier and test dataset with labeling information to evaluate the quality of the classifier. For example, if you are implementing Decision Tree using C/C++, your program should be able to finish the entire classification process when you type this command:

./DecisionTree train_file test_file

If you use Java:

java classification.DecisionTree train_file test_file

If you use Python:

python2 DecisionTree.py train_file test_file (or python3 DecisionTree.py train_file test_file)

Your program should output (to stdout) k*k numbers (k number per line, separated by space, k lines total) to represent the confusion matrix of the classifier on testing data, where k is the count of class labels. These lines are sorted by the label id. In i-th line, the j-th numbers are the number of data points in test where the actual label is i and predicted label is j. And you will be needing these values in Step 4.
An example output can be found as follows (when k=4):

50 30 16 23

12 43 21 9

0 21 71 12

1 23 11 67

Note that if k=2, the output is a 2x2 matrix, which is similar to the table where the numbers represent true/false positive/negative. For more information about the confusion matrix and how to evaluate performance based on it, please read https://en.wikipedia.org/wiki/Confusion_matrix or watch the video.

To sanity check your output format, we provide a script, accuracy.py (Edit on Nov 26: this is clickable), for computing overall accuracy. You can pipe your output to this script using command line and check if it can generates the correct overall accuracy. For example, if C/C++ is used as the coding language, you may append " | python2 accuracy.py" at the end and execute:

./DecisionTree train_file test_file | python2 accuracy.py

The same also applies for java and python.

Step 3: Implement Ensemble Classification Method (40 pts)

In this step, you will implement an ensemble classification method using the basic classification method in Step 2, i.e., random forest. Very similarly, your ensemble classification method should take train_file and test_file as input, and output k*k values in k lines as the confusion matrix so that you can use these numbers to evaluate classifier quality in Step 4. For example, you are implementing random forest using C/C++, and the command line of running your program should look like this:

./RandomForest train_file test_file

If you use Java:

java classification.RandomForest train_file test_file

If you use Python:

python RandomForest.py train_file test_file

Step 4: Model Evaluation and Report (15 pts)

In this step, you will apply both methods you implemented in Step 2 and Step 3 on three real-world datasets and a synthetic dataset. Note that we preprocessed these datasets and partitioned them into training and test for you (click on the numbers in the table to download), so please do not use the original datasets directly.

Name	Num of Attributes	Training Set Size	Test Set Size	Source Link	Labels
balance.scale	4	400	225	http://archive.ics.uci.edu/ml/datasets/Balance+Scale	1 is Balanced, 2 is Right, 3 is Left
nursery	8	8093	4867	http://archive.ics.uci.edu/ml/datasets/Nursery	1 is priority, 2 is very_recom, 3 is spec_prior, 4 is not recom, 5 is recommend.
led	7	2087	1134	http://archive.ics.uci.edu/ml/datasets/LED+Display+Domain	1 is three, 2 is other numbers
synthetic.social	128	3000	1000	N/A	1, 2, 3, and 4 represent four social groups respectively

If you are curious about the semantic meaning of each attribute or labels, please read the source links associated with the first three datasets. The fourth dataset is generated via distributed representation learning on a synthetic social network (a.k.a, network embedding), where binning is further performed to generate categorical attributes. You do not need to know these details in order to finish this assignment.

The main difference between the fourth dataset and the other three is that the former has a significantly larger number of attributes.

You need to apply both your basic classification method and your ensemble classification method on each dataset, and calculate the following model evaluation metrics using the output of your classification methods for both training and test datasets. Please do this manually or write a separate program (you do not need to turn this in), by taking the k*k numbers as input. Do not output these metrics from your classifiers directly since the auto-grader won't be able to recognize them.

For overall performance, output:

Accuracy (overall accuracy = sum of the diagonal of your output matrix / sum of all entries in your output matrix)

For each class, output:

F-1 Score.

Note 1: To evaluate the performance on each class, we regard the target class as positive and all others as negative.

Note 2: The detailed equations of the above measures based on Confusion Matrix can be found in https://en.wikipedia.org/wiki/Confusion_matrix.

Parameter settings for each dataset in your submission

Please hardcode your choice of parameters (depth of decision tree, number of trees in random forest, etc.) **for each dataset** into your program. In other words, your program should choose the corresponding parameters according to the input datasets. Specifically, if **train_file** (the path to the training set) **contains** the string "balance.scale" the parameters tuned for the provided balance.scale dataset are used in this round of execution. The same goes for nursery, led, and synthetic.social.

Besides the four provided pairs of training-test files, we also hold some hidden datasets for the auto-grader to evaluate your implementation. The hidden datasets differ from the provided datasets in that the training-test files are resampled, while the raw data used to generate the hidden datasets are the same as those used to generate the provided datasets. The filename of any hidden datasets would also contain balance.scale, nursery, led, or synthetic.social. As a result, the parameters you selected for the provided datasets will also work fine with the hidden datasets.

Report

Now you are ready to write your report. You should include the following sections in your report:

- At the very beginning of the report, summary of the overall accuracy on the test dataset for each dataset (2 methods * 4 datasets = 8 numbers). These metrics are used to score the performance of the model, which is to be detailed in the next two sections;
- Brief introduction of the classification methods in your classification framework;
- All model evaluation measures you calculated above ((1 overall metric + # class * 1 per-class metric) * (1 on training set + 1 on test set) * 2 methods * 4 datasets);
- Parameters you chose during implementation and why you chose these parameters;
- Your conclusion on whether the ensemble method improves the performance of the basic classification method you chose, why or why not.

Competition (5 pts)

To add just a little competition, we test your random forest implementation on a hidden dataset very similar to the provided synthetic.social. Ranking by the overall accuracy the implementation achieves on the test data, the person ranks first receives 5 points, the person ranks last receives 0 points, and everyone else in between receives a score linearly interpolated according to their rank. In the unfortunate case, an implementation does not finish within three minutes or does not generate results, the person would receive 0 points.

Homework Scoring Criterion

The total score for this homework is 100 points, including 80 points for implementation, 15 points for the report, and 5 points for the competition.

For implementation, one receives 10 points for each model that outperforms the (very lenient) cut-off score on each hidden dataset. The performance is solely determined by overall accuracy on the test dataset. We provide the cut-off scores as follows. Since the hidden datasets have distributions very similar to the provided ones, one can assume they would pass on the hidden datasets as long as they can pass on the provided datasets.

	DecisionTree	RandomForest
balance.scale	0.50	0.60
nursery	0.80	0.80
led	0.80	0.80
synthetic.social	0.40	0.50

Note that plagiarism would result in 0 points. In the past, we have also identified codes that generate fake confusion matrix as output, instead of implementing the algorithms. In this case, 0 points would also be given, since the models were not actually implemented.

We grade the report based on its completeness and grade the competition as described in the previous section.

Project Organization and Submission

The auto-grading process will be performed on a Linux server, so make sure your source code can be compiled in the EWS Linux environment (or in a major distro like Ubuntu, Mint, Fedora or Arch with a relatively new version of gcc/g++ or JDK).

If you use C/C++
You must attach a Makefile so that the grader can compile your programs correctly. If you are a Windows user and use Visual Studio as your IDE, please add a Makefile after you finish implementation and make sure your code can be compiled properly using gcc/g++. Your project structure should look like this:

```
yournetid_assign4/
        |----------report.pdf
        |----------c_code/
                    |----------Makefile
                    |----------*.c/c++
```

If C++ is used instead of C, you may replace the above folder name c_code/ with cpp_code/ although using c_code/ should still work thanks to the Makefile.

After compilation, your Makefile should be able to generate binary target files. Your binary files should be DecisionTree and RandomForest. Also, as we discussed in Step 2 and 3, the input arguments for your binary should look like this:

./DecisionTree train_file test_file

If you use Java

Very similarly, please make sure your code can be compiled in Linux environment. Your entrance class should be placed under the classification package, and your project structure should look like this:

```
yournetid_assign4/
        |----------report.pdf
        |----------java_code/
                    |----------other_packages/
                                        |----------*.java
                    |----------classification/
                                        |----------*.java
```

The grader will simply execute javac classification/*.java to compile your programs, and your program should be able to generate binary files with corresponding classification method names. The input arguments are defined as follows.

java classification.DecisionTree train_file test_file

If you use Python

Very similarly, please make sure your code can be executed in Linux environment. Also, please make explicit whether you are coding in python2 or python3. Your project structure should look like this:

For python2,

```
yournetid_assign4/
        |----------report.pdf
        |----------python2_code/
                    |----------*.py
```

For python3,

```
yournetid_assign4/
        |----------report.pdf
        |----------python3_code/
                    |----------*.py
```

The grader will run the following command to test your programs:

python2 DecisionTree.py train_file test_file (or python3 DecisionTree.py train_file test_file)

After your finish your implementation and report, please compress your project into zip format and rename your zip file to yournetid_assign4.zip. Please make sure after unzip the submission, everything will be in a folder with "yournetid_assign4" as the name

rather than layers of nested folders, and you can ensure this using the format_check.py (Edit on Nov 26: this is clickable) script we provide. (Run the command: python2 format_check.py yournetid_assign4.zip).

Double check before you submit

Your programs should be able to handle both absolute paths and relative paths to different training and test files. Do not assume that the files are located in the same folder of your programs.

The hidden datasets are roughly the same scale as the 4 datasets that we provided. Make sure that your programs can finish within 3 mins for each dataset. The auto-grader will shut down your program after 3 mins if no results are detected (and you will receive 0 points for the corresponding step).

Remember to summarize overall accuracy at the beginning of your report.

In the past, when the same auto-grader was used for programming assignment, around 20% of the submissions would not pass the auto-grader, mostly because these submissions did not strictly follow the project organization rules we stated above. In this case, TAs will manually grade these assignments but 6 to 14 points will be taken off from your assignment. Some common mistakes include using Java without package structure, naming the program by decisiontree instead of DecisionTree, generating incorrect zip file name or folder name after unzip, etc. Also for this reason, we provide the two scripts accuracy.py and format_check.py for self sanity check this year (Edit on Nov 26: if they elude you, search for these two filenames on this page to find the clickable links). Consider using them before submission.

Now you can submit your assignment through Compass 2g!!

Congratulations, you just finished the 2nd (and last) programming assignment of CS412!!!

Frequently Asked Questions

1. How to deal with unseen values or unseen features in test dataset?

In Decision Tree, this is hard to deal with during testing. You can ignore a feature if you never saw it in training however you need to bypass an unseen value in test data for a seen feature. In this homework, you can assume no feature/index unseen in the training set will appear in the test set; when we say ignoring a feature here, we intend to provide a solution to the more general application scenario.

You can do 1) randomly choose a label, 2) randomly choose a path to finish traversing the tree, or 3) choose a path which most data flow if you kept this information in your decision tree.

There are more elegant ways to handle such cases but since we did not cover them in the lecture notes, you can go with one of the naive ways we mentioned above.

2. Should random forest use overlapped attributes or overlapped subsets?

Yes.

3. How to handle duplicate tuples in training?

You should count each tuple as one when you are building classifiers no matter whether they are duplicate tuples or not.

4. How many branches should a feature have during tree construction?

Since assignment 4 only deals with **categorical** features, we don't do value grouping nor select splitting point for the feature. We simply regard every possible value as one branch. Thus the # of branches should be the same as # of possible values of the feature.

5. Should parameters be tuned on a validation set?

In principle, model parameters are supposed to be tuned according to the model performance on the validation set, which is distinct from the training test and the test set. However, in this homework, we do not hold a validation set for simplicity, and you would tune the parameters on the test set.

6. Why not host this programming homework on hackerrank?

HackerRank enforces a time limit, which can be set to one minute at most. While decently optimized implementations of decision tree and random forest should complete each test case in one minute, we would allow three minutes as aforementioned.