Assignment 4 - Fall 2016

Created by Tao, Fangbo, last modified by Gui, Huan on Nov 28, 2016

Assignment 4 (Distributed Thu. Nov. 3, 2016, Due Tue. Dec. 6, 2016)

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 but please do not work together or share code or results.
- Related libraries or programs can be found on-line, 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.
- There are four steps in this assignment, which covers most sections in Chapter 8, you should be able to work on Step 1 and Step 2 after the lecture on Oct 27, and you should be able to work on Step 3 and Step 4 after the lecture on Nov 3.
- You will need to make decisions about parameter values when implementing these algorithms (e.g., depth of the decision tree, number of samples in ensemble methods, 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 with many datasets (we only provide 4 datasets here so you have something to work with). The auto-grader also has plagiarism detection feature, so please do not copy other people's programs or use modified version of existing resources. A more detailed project organization guidance can be found at the end of the assignment.
- Late policy:
  10% for one day (Dec. 7th, 11:59PM),
  20% for two days (Dec. 8th, 11:59PM),
  40% for three days (Dec. 9th, 11:59PM).
- A section titled Frequently Asked Questions on Piazza is added at the end of the assignment.

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 real-world 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 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 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., sk-learn. 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 an integer starting from 1 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 Basic Classification Method (20 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 training_file test_file
```

If you use Java:

java classification.DecisionTree training_file test_file

If you use Python:

python DecisionTree.py training_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 confusion matrix and how to evaluate performance based on it, please read https://en.wikipedia.org/wiki/Confusion_matrix or watch the video.

Step 3: Implement Ensemble Classification Method (20 pts)

In this step, you will implement an ensemble classification method using the basic classification method in Step 2, i.e., random forest. For more details of Random Forest, please refer to slide 52 in http://hanj.cs.illinois.edu/cs412/bk3_slides/08ClassBasic.pdf or https://en.wikipedia.org/wiki/Random_forest#Algorithm. Very similarly, your ensemble classification method should take training_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 training_file test_file

Step 4: Model Evaluation and Report (10 pts)

In this step, you will apply both methods you implemented in Step 2 and Step 3 on 4 real-world datasets. Notice 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	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
poker	10	1042	677	http://archive.ics.uci.edu/ml/datasets/Poker+Hand	1 one pair. 2 three of a kind

If you are curious about the related domain application, and the semantic meaning of each attribute or labels, please read the source links associated with each dataset, but you do not need to know these details in terms of finishing this assignment.
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

For each class, output:

Sensitivity, Specificity, Precision, Recall, F-1 Score, F \beta score (\beta = 0.5 and 2)

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.

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

- brief introduction of the classification methods in your classification framework;
- all model evaluation measures you calculated above (7 metrics * 2 methods * 4 datasets);
- parameters you chose during implementation and why you chose these parameters;
- and also your conclusion on whether the ensemble method improves the performance of the basic classification method you chose, why or why not;

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 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++.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
```

```
        |----------------code/
                    |----------------Makefile
                    |----------------*.c/c++
```

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 classification package, and your project structure should look like this:

```
yournetid_assign4/
        |----------------report.pdf
        |----------------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 compiled in Linux environment. Your project structure should look like this:

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

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

python DecisionTree.py train_file test_file

After your finish your implementation and report, please compress your project into zip format (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 rename your zip file to 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 unseen datasets are roughly the same scale of the 4 datasets that we provided here. 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).

Whenever we use auto-grader for programming assignment, around 20% of the submissions won't pass the auto-grader, mostly because these submissions do not follow the project organization rules we stated above strictly. In this case, TAs will manually grade these assignments but we will take 3 to 7 points off your assignment if this happens. Some common mistakes include when using Java, didn't use package structure, named your program decisiontree instead of DecisionTree, incorrect zip file name or folder name after unzip, etc.

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

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

Frequently Asked Questions on Piazza

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 a unseen value in test data for a seen feature.

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 didn't 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. For more details, refer to https://piazza.com/class/is2n71wakk671d?cid=389.

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.

No labels