CS373 HW3

March 13, 2018

Due date: 11:59pm Monday March 26, 2018.

Instructions for submission:

You need to implement this assignment from scratch in Python. Do not use any already implemented version of these two algorithms like the scikit-learn library. Programs that print more than what is required will be penalized.

Your code needs to run in the server: `data.cs.purdue.edu`. Check that it works there by executing your script from the terminal.

For part 1, submit your code as a Python program. For part 2, type your answers in a PDF file. Your homework **must** contain your name and Purdue ID.

To submit your assignment, log into `data.cs.purdue.edu` (physically go to the lab or use ssh remotely) and follow these steps:

1. Make a directory named `yourusername-hw3` (all letters in lower case) and copy your PDF file and Python file inside it. To do it remotely, use:

   ```
   scp ./path/to/your-file.pdf your-id@data.cs.purdue.edu:./remote/path/from-home-dir/
   ```

2. Go to the directory containing `yourusername-hw3` (e.g., if the files are in /homes/dan/dan-hw3, go to /homes/dan), and execute the following command:

   ```
   turnin -c s373 -p hw3 yourusername-hw3
   ```

   (e.g. Dan would use: `turnin -c s373 -p hw3 dan-hw3` to submit his work)

   Note that `s373` is the course name for turnin. **It is not a typo.**

3. To overwrite an old submission, simply execute this command again.
4. To verify the contents of your submission, execute the following command:

   ```
   turnin -v -c s373 -p hw3
   ```

1 Specification

1.1 Dataset Details

Download the file yelp.csv from the course page. The document has 24,813 rows.

In order to read the CSV data and obtain a numpy matrix X with the necessary attributes you can use the following code:

```
import pandas as pd
data = pd.read_csv(filename, sep=',', quotechar='"', header=0)
features = ['stars', 'priceRange', 'goodForKids', 'caters',
            'outdoorSeating', 'waiterService', 'attire', 'noiseLevel',
            'alcohol', 'open', 'goodForGroups']
data = data[features]
X = data.as_matrix()
```

1.2 Code Details

Your python script should take at least three (3) arguments as input.

1. datasetPath: corresponds to path to the dataset file, in the same format as yelp.csv.
2. model: model that you want to use. In this case, we will use:
   - `vanilla` for the full decision tree
   - `depth` for the decision tree with static depth
   - `prune` for the decision tree with post-pruning
3. training set size as a percentage.

Each case may have some additional command-line arguments, which will be mentioned in its format section. Use `sys.argv` to get the list of arguments.

Your code should read the dataset from the datasetPath, extract the required features, train your decision tree on the training set, and test it on the test set. Name your file `decisiontree.py`.

For debugging purposes, you can use a small fraction of the dataset, for example, by using `X[:1000]` to work with the first 1000 data points.

2 Part 1: Decision Trees

Note: You need to submit only your code as a separate file (`decisiontree.py`) for this section.

Features: Consider the 11 attributes mentioned in the dataset section as the input features.

Class label: Consider the attribute `goodForGroups` as the class label. This will be the last element of each row in the dataset.

Note that `stars` and `priceRange` are numeric attributes.

1. Implement a binary decision tree with no pruning using the ID3 algorithm. **(20 points)**

   Format:

```
$ python decisiontree.py ./some/path/filename.csv vanilla 90
Training set accuracy: 0.9123
Test set accuracy: 0.8123
```

The third argument here is the training set percentage. So, for example, 90% would be used as the training set and 10% as the test set.

2. Implement a binary decision tree with a given maximum depth. **(25 points)**
   Format:

```
$ python decisiontree.py ./some/path/filename.csv depth 80 10 14
Training set accuracy: 0.9123
Validation set accuracy: 0.8523
Test set accuracy: 0.8123
```

   The third argument is the training set percentage. The fourth argument is the validation set percentage (as a percentage of the whole dataset). The fifth argument is the value of depth.

   So, for example, the above command would use a training set of size 80%, a validation set of size 10%, and a test set of size 10%, and with maximum depth as 14.

   Note that you have to print the validation set accuracy for this case.

3. Implement a binary decision tree with post-pruning using reduced error pruning. **(30 points)**

```
$ python decisiontree.py ./some/path/filename.csv prune 80 10
Training set accuracy: 0.9123
Test set accuracy: 0.8123
```

   The third argument is the training set percentage. The fourth argument is the validation set percentage (as a percentage of the whole dataset).

   So, for example, the above command would use a training set of size 80%, a validation set of size 10%, and a test set of size 10%.

3 Part 2: Analysis

Note: You need to submit only your answers in the PDF for this section.

1. For this part alone, shuffle the dataset *before* splitting it into training set, test set, etc. You can use `shuffle` from the `random` module. (**4 points**)

   Compute the accuracies of the three variations with and without shuffling. What differences do you observe and how do you explain them?

   (For the static-depth case, consider the following values of depth = {5, 10, 15, 20, 25} and pick the one that gives the best accuracy on the validation set.

   For all three cases, use training set size as 80% and validation set size as 10% when needed.)

2. In the static-depth case, calculate the optimal depth of your decision tree for each case when training percentage is 10%, 25%, 50%, 75%, and 90%. (**9 points**)

   (Again, consider values of depth from {5, 10, 15, 20, 25}.)

   Plot a curve of test set accuracy and training set accuracy against training set size on the same plot. Also plot the baseline default accuracy that would be achieved if you just predicted the most frequent class label in the overall data.

   Explain how and why the optimal choice of depth varies with the size of the training set.

3. When pruning, why is it better to use a separate validation set to decide when to prune instead of using the test set itself? (**4 points**)

   Explain with reference to your own observed accuracies.

4. How would you convert your decision tree (in the `depth` and `prune` cases) from a classification model to a ranking model? (**8 points**)

   That is, how would you output a ranking over the possible class labels instead of a single class label?