HMC CS 158, Spring 2018
Problem Set 2 Project: Titanic Survival

Goals:
- To investigate the performance of various Decision Tree classifiers.
- To practice classification and evaluation using `scikit-learn`.
- To try your hand at making better predictions!
- To open up the "black-box" of `scikit-learn` and implement a `DecisionTreeClassifier`.

For this assignment, you should work individually or with the same partner as last week.

Submission

You should submit any answers to the exercises in a single file `writeup.pdf`. This writeup should include your name and the assignment number at the top of the first page, and it should clearly label all problems. Additionally, cite any collaborators and sources of help you received (excluding course staff), and if you are using slip days, please also indicate this at the top of your document.

Your code should be commented appropriately. The most important things:
- Your name and the assignment number should be at the top of each file.
- Each class and method should have an appropriate doctsring.
- If anything is complicated, it should include some comments.

There are many possible ways to approach the programming portion of this assignment, which makes code style and comments very important so that staff can understand what you did. For this reason, you will lose points for poorly commented or poorly organized code.

When you are ready to submit, make sure that your code compiles and remove any debugging statements. Then rename the top-level directory as `<username1>_<username2>_ps2`, with the usernames in alphabetical order (e.g. `hadas_yjw_ps2`). This directory should include the electronic version of your writeup and main code, any files necessary to run your code (including any code and data provided by staff), and follow the same structure as the assignment directory provided by staff. So, for this assignment, your directory should have the following structure:
- `<username1>_<username2>_ps2/`
  - `data/`
    - `titanic_train.csv`
    - `titanic_test.csv`
    - `<username1>_<username2>_titanic.csv` (extra credit)
  - `source/`
    - `dtree.py` (with your modifications)
    - `titanic.py` (with your modifications)
    - `util.py`
  - `writeup.pdf`
  - `dtree.pdf` (pdf printout of `dtree.py`)
  - `titanic.pdf` (pdf printout of `titanic.py`)

Parts of this assignment are adapted from course material by David Kauchak (Middlebury).

Package this directory as a single `<username1>_<username2>_ps2.zip` file, and submit the archive. Additionally, to aid the staff in grading, submit your `pdf`'s as separate files, and submit your predictions as a separate file as well.

Introduction[1]

Last week, we used some baseline classifiers to predict passenger survivability on the Titanic. This week, we will use `scikit-learn` to train a `DecisionTreeClassifier` on the data.

Starter Files

code and data

- code : `titanic.py` (Parts 1 and 2), `dtree.py` (Part 3)
- data : `titanic_train.csv`, `titanic_test.csv`

documentation

- Decision Tree Classifier:
  http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- Cross-Validation:
  http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
- Metrics:
  http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

Download the code and data sets from the course website. (Be sure to use the starter code for this week's assignment! You will have to insert your `RandomClassifier` code from last week.)

Note that any portions of the code that you must modify have been indicated with `TODO`. Do not change any code outside of these blocks.

1 Evaluation [7 pts]

Using the predictive capabilities of the `scikit-learn` package is very simple. In fact, it can be carried out in three simple steps: initializing the model, fitting it to the training data, and predicting new values. For this section, you will modify `titanic.py`.

(a) **(1 pts)** Train and evaluate a `DecisionTreeClassifier` (using the class from `scikit-learn` and referring to the documentation as needed). Make sure you initialize your classifier with the appropriate parameters; in particular, use the 'entropy' criterion discussed in class. What is the training error of this classifier?

[1] This assignment is adapted from the Kaggle Titanic competition, available at https://www.kaggle.com/c/titanic. Some parts of the problem are copied verbatim from Kaggle.

Optional: If you want to look at the trained decision tree, you can use GraphViz[2], an open source graph visualization software, and the pydot library[3], a Python interface to Graphviz's dot language. Since this requires installing other packages, we have provided you with a decision tree (trained on the entire training data set) in dtree.pdf.

(b) **(2 pts)** So far, we have looked only at training error, but as we learned in class, training error is a poor metric for evaluating classifiers. Let us use cross-validation instead.

Implement the missing portions of error(...) according to the provided specifications. You may find it helpful to use train_test_split(...) from scikit-learn. To ensure that we always get the same splits across different runs (and thus can compare the classifier results), set the random_state parameter to be the trial number.

Next, use your error(...) function to evaluate the training error and (cross-validation) test error of each of your three models. To do this, generate a random 80/20 split of the training data, train each model on the 80% fraction, evaluate the error on either the 80% or the 20% fraction, and repeat this 100 times to get an average result. What are the average training and test error of each of your classifiers on the Titanic data set?

(c) **(2 pts)** One problem with decision trees is that they can *overfit* to training data, yielding complex classifiers that do not generalize well to new data. Let us see whether this is the case for the Titanic data.

One way to prevent decision trees from overfitting is to limit their depth. Repeat your cross-validation experiments but for increasing depth limits, specifically, $1, 2, \ldots, 20$. Then plot the average training error and test error against the depth limit. (Also plot the average test error for your baseline classifiers. As the baseline classifiers are independent of the depth limit, their plots should be flat lines.) Include this plot in your writeup, making sure to label all axes and include a legend for your classifiers. What is the best depth limit to use for this data? Do you see overfitting? Justify your answers using the plot.

(d) **(2 pts)** Another useful tool for evaluating classifiers is *learning curves*, which show how classifier performance (e.g. error) relates to experience (e.g. amount of training data).

Run another experiment using a decision tree with the best depth limit you found above. This time, vary the amount of training data by starting with splits of 0.05 (5% of the data used for training) and working up to splits of size 0.95 (95% of the data used for training) in increments of 0.05. Then plot the decision tree training and test error against the amount of training data. (Also plot the average test error for your baseline classifiers.) Include this plot in your writeup, and provide a 1-2 sentence description of your observations.

2 Extra Credit: Contest [+3 pts]

Now let us experiment (and push yourself) a bit more by holding a Titanic competition. For this section, you will modify titanic.py.

For this contest, we have provided a held-out test data set, titanic_test.csv, that is missing all information about passenger survival. Staff has the true survival information for these passengers, and your goal is to minimize the prediction error on this data set.

[2] http://www.graphviz.org/

[3] https://pypi.python.org/pypi/pydot

You should submit your predictions in a file called `<username1>_<username2>_titanic.csv` in your submission. This file should include your predictions on the test data set, which can be written using `write_predictions(...)`. Additionally, **include a short report in your writeup that describes what you did, including any rationale**.

Staff will compare your predictions to the ground truth and award extra credit points based your description of your approach and on the performance of your classifier.

Ground rules:

- *Your predictions must be auto-generated from your code and use only the provided data sets.* In particular, no looking up answers on the internet (which defeats the entire purpose of this exercise).
- *You must use a Decision Tree classifier.* Now that you are familiar with the basics of decision trees and evaluation techniques, try your hand at improving your model. Feel free to experiment with feature combinations or decision tree parameters. You must try at least one variation on the exercises in this assignment.

Note: It is okay to try something simple. It is also okay if it turns out that your new method performs worse than the original one – just explain and document your results. (You probably will not win the contest however...)

3 Decision Tree Implementation [9 pts]

Now that you know how to use `scikit-learn`'s `DecisionTreeClassifier`, let us try implementing the classifier! For this section, you will modify `dtree.py`.

We have based our `DecisionTreeClassifier` on that of `scikit-learn` so that the attributes and methods look familiar. If you take a look at the class `DecisionTreeClassifier`, you should see that it contains an attribute `tree_` of class `Tree` that does the heavy lifting. `Tree` is itself based on `sklearn.tree._tree.Tree`, which is a Python wrapper around a C class.

Look at the `Tree` documentation (adapted from `scikit-learn`) to understand how it represents a tree data structure. Basically, a tree is stored using parallel arrays, and the index into that array indicates the node. So, for example, the `tree`

```
    |
/---0---\
|       |
1    /--2--\
     |     |
     3     4
```

would have `node_count = 5` and the following arrays to indicate its structure:

```
            node   0  1  2  3  4
children_left  = [ 1 -1  3 -1 -1]
children_right = [ 2 -1  4 -1 -1]
```

where a leaf node is indicated by both left and right children being `TREE_LEAF = -1`.[4]

Your goal is to complete the implementation of `Tree`. We have provided some test code in `main(...)`. The movie data set is the same as the one used in-class. However, `Tree` has several differences from the decision trees presented in-class:

- `Tree` is a binary tree and therefore splits data into two subsets by comparing a feature value to a threshold. For node `node` with feature f = `feature[node]` and threshold t = `threshold[node]`, samples with $f \leq t$ are sorted to the left subtree, and samples with $f > t$ are sorted to the right subtree. We choose this threshold by iterating over all possible thresholds and finding one with the highest information gain. This means that the decision tree you implement will work for both discrete and continuous features.

- As `Tree` sorts samples to node `node`, it encodes the class counts in `value[node]`. This means that each decision tree node implicitly represents a probability distribution over the classes.

`Tree` has two main methods and several helper methods. You may find it useful to look through the documentation of each method before writing any code.

(a) **(1 pts)** Complete `Tree._entropy(...)`. Note that you will receive full credit only if your implementation works for an arbitrary set of classes; in particular, it should work for more than two classes.

(b) **(1 pts)** Complete `Tree._split_data(...)`.

(c) **(1 pts)** Complete `Tree._information_gain(...)`. Again, you will receive full credit only if your implementation works for an arbitrary set of classes.

(d) **(4 pts)** Complete `Tree._build_helper(...)`.

(e) **(2 pts)** Complete `Tree._predict(...)`.

If you have implemented everything correctly, the `assert` statements in `main(...)` should pass.

[4] We understand that many of you would probably use `TreeNode` classes to implement `Tree` – if we were implementing our own machine learning library from scratch, we would probably do the same! However, for this assignment, using the same implementation as `scikit-learn` allows us to leverage the same `print_tree` function, and you can directly compare your inferred tree with that of `scikit-learn`.