C4.5 Implementation [100 points]#

In this assignment you will be implementing the C4.5 decision tree algorithm and running it on real emails to train a spam filter.

https://en.wikipedia.org/wiki/C4.5_algorithm

First get the assignment by running:

```
git clone https://github.com/pjreddie/C4.5-Homework
cd C4.5-Homework
```

You will be modifying the file `tree.py`. Tests for your program are in `test.py`. To test your implementation, run:

```
python test.py
```

You will see the number of tests your implementation passes and any problems that arise.

Note: Please use Python 3.6 for the assignment and only modify the `tree.py` file. This is the only file you will be submitting.

Splitting The Data [10 points]###

Decision trees work by partitioning or splitting the input space into smaller regions. Your program will have to split a dataset into two pieces by determining which points fall on which side of a threshold for some feature.

First let's look at the format of the data. Start a `python` shell and try these commands:

```
import data
d = data.get_college_data()
print d
print d[0]
print d[0].label
print d[0].values
```

So `data` is just a list of `Point`s. Each `Point` has a label, in this case "College" or "No College", and each `Point` has a set of values.

Fill in the `split_data` function so that `left` contains all points whose value for the feature is less than the threshold and `right` contains all the other points.

Calculating Entropy [10 points]###

The C4.5 algorithm finds partitions for the data that minimize entropy so we need to be able to calculate entropy. Entropy is given as the negative sum across all events (in this case classes) of the probability of that event times the log probability of that event:

```
  - sum(prob(event) * log(prob(event)))
```

To calculate the probabilities for each class in a given dataset we first need to count the occurences of each class. Fill in `count_labels` to return a dictionary containing the number of times each label occurs in the data.

Next fill in `counts_to_entropy` to convert a dictionary of counts to the entropy of the data. Once these two methods are complete, run `python test.py` and make sure the entropy calculation test succeeds.

Finding The Right Threshold [30 points]###

Given a dataset and some feature to split on, we need to be able to find the best split that gets us the most information gain. One way to do this is to look at every data point and try splitting on that data point's value for the feature. Implement this method in `find_best_threshold`.

While this will give the correct answer, it involes re-splitting the data numerous times and recalculating entropy from scratch which can be slow on large datasets. We need a faster way to find the best threshold.

One way to do it is to sort the dataset by the feature we are splitting on. Then we can go through the sorted data in order, moving data points from the `right` split to the `left` and keeping a rolling count of the probabilities of each label. This saves a lot of work when calculating information gain for each split. Implement this method in `find_best_threshold_fast`.

Finding The Best Split [10 points]###

Now we can find the best split of the data over some feature but to run C4.5 we want to find the best split over all thresholds and all features. Fill in `find_best_split` to return the best feature and threshold that maximize information gain.

Finish C4.5 [20 points]###

Given some training data we don't want to just split it one time, we want to keep going until we can't get any more information gain (or until some maximum tree depth is reached).

Fill in the `c45` function to do the following:

- If all the points have the same label, if no split can provide any information gain, or if the maximum tree depth has been reached, make a leaf with the data, recording the expected value of each label (this is provided in `make_leaf`.
- Otherwise split the data at the best threshold for the best feature. Make an internal (non-leaf) node for that feature and threshold.
- Create the left and right subtrees of the node by recursing on the two partitions of the data.

Run Your Submission [10 points]###

Once all the basic tests pass it's time to run your algorithm on real data. Download and unzip the spam dataset:

http://pjreddie.com/media/files/spam.tar.gz

You can do this with the commands:

```
wget http://pjreddie.com/media/files/spam.tar.gz
tar xzf spam.tar.gz
```

Also open `test.py` and comment out or delete line 45 so that it runs the final test. This test will train your decision tree on about 20,000 emails labelled as spam and not spam ("ham"). We have set up the the data loader to load the emails and calculate character statistics as features (e.g. the frequency of the letter 'a' or of a '?').

Run `python test.py` again to train your decision tree and calculate your accuracy on the validation data. By default it trains a single tree with a maximum depth of 4. This should get you an accuracy of around 68% at classifying spam.

Tune your `submission` function to get your accuracy over 75% on the validation data. Describe what you did to increase your accuracy. How does changing the maximum depth of the tree affect your accuracy? Write a comment above your `submission` function.

Perfect Your Submission [10 points + EXTRA]###

Finalize your `submission` function to take a train and test data and return a list of predictions for the test data. You can stick with just training a single decision tree, you could try modifying the c45 algorithm, you could average predictions across multiple decision trees, or you could come up with something new!

After the homework deadline we will run all of your submissions on a test dataset to see who can train the best model. The test data will be different than either the training or validation data so make sure you don't overfit your model too much. Extra points on the homework will be awarded to people with the highest performing algorithms.

Submitting

Submit your modified `tree.py` file to the dropbox folder on Folio.