CS446: Machine Learning **Fall 2014**

Problem Set 2

Nikhil Unni *Handed In: September 25, 2014*

1. Learning Decision Trees

   a. I wrote a python script to do both of these parts. The code is in script1.py. My ID3 tree is:

```
if Color = Purple:
   if Act = Stretch:
      if Age = Adult:
         class = T
      else:
         class = F
   else
      class = F
else:
   if Size = Small:
      class = T
   else:
      if Act = Stretch:
         if Age = Adult:
            class = T
         else:
            class = F
      else:
         class = F
```

   b. The code for this heuristic is in the same aforementioend script.py script. I actually ended up with the same tree as part [a.]. I designed it so that the MajorityError of an attribute is the minimum of the MajorityError of each class split. This resulted in the exact same tree as above. (See part [a.] if you want to see the tree.)

   c. Again, for both heuristics I got the same tree when training on the first 12 examples with maximum depth of 2. My trees for both were:

```
if Color = Purple:
   class = F //This was actually (if Act=Dip) => False, (else) => False,
             //when generated by my code, so I just simplified to this.
else:
   if Size = Small:
      class = T
```

```
else:
   class = F
```

Testing on the remaining 4 instances, the two algorithms get 1/4, or 25% incorrect on the data. It makes sense that our accuracy on the training data went down by limiting the size of the tree. By doing this, our hypothesis becomes less expressive and begins to underfit the data. So when we test on the remaining fourth of our original dataset it seems only natural that it'd get at least 1 wrong.

2. Decision Trees as Features

   a. I chose not to change any features from the original features given. Also, for all of the problems below, I stuck with Weka to run the algorithms.

   b. My accuracies from the 5-fold cross-validation were : 340/400, 333/400, 339/400, and 328/400, giving an overall accuracy, p_a, of 83.34%. This yields a 99% confidence interval of [0.816,0.851].

   c. My accuracies from the 5-fold cross-validation were: 355/400, 351/400, 343/400, 354/400, and 360/400, with an average of $p_a = 88.15\%$. This gives a 99% confidence interval of [0.863,0.900].

   d. My accurances for the tree of maximum depth 4 from the 5-fold cross-validation were: 310/400, 310/400, 304/400, 316/400, and 311/400, with an average of $p_a = 77.55\%$. This gives a 99% confidence interval of [0.763,0.788].
   My accuracies for the tree of maximum depth 8 from the 5-fold cross-validation were: 333/400, 335/400, 324/400, 324/400, and 348/400, with an average of $p_a = 84.20\%$. This gives a 99% confidence interval of [0.801,0.883]

   e. My accuracies for the SGD trained on ID3 stumps from the 5-fold cross-validation were: 346/400, 346/400, 348/400, 340/400, 340/400, with an average of $p_a = 86.00\%$. This gives a 99% confidence interval of [0.849,0.871]

3. Evaluation

Overall, I was pretty underwhelmed by my results. The relative frequency of negative examples in the sample data was about 71%, so even if I had some function, $f(x) = False$, it'd perform almost as well as my "trained" algorithms. However, to be fair, when I tested this "zero function" on the 5-fold cross-validation, it scored around 66% with a pretty tight confidence interval. So the trained algorithms *do* perform better (to a certain degree of confidence), it's just that they're not remarkably better.

The order of accuracies, from least p_a to greatest is:

1. ID3 with maximum depth of 4 : $p_a = 77.55\%$, CI : [0.763,0.788]
2. Stochastic Gradient Descent : $p_a = 83.35\%$, CI : [0.816,0.851]
3. ID3 with maximum depth of 8 : $p_a = 84.20\%$, CI : [0.801,0.883]

4. SGD trained on ID3 features : $p_a = 86.00\%$, CI : [0.849,0.871]
5. ID3 with unbounded depth : $p_a = 88.15\%$, CI : [0.863,0.900]

However, SGD, and ID3 depth 8 have overlapping confidence intervals, as do ID3 depth 8/SGD trained on ID3 and SGD trained on ID3/ID3 with unbounded depth, so we can't *really* say that one is better than the other with confidence; it could've just been an "unlucky" sample from the population. However, we can say with confidence that ID3 with unbounded depth is better than the base SGD, which is better than ID3 depth 4, which is better than $f(x) = False$.

4. Implementation

1. I wrote all of #1 in Python (and then additionally wrote a testbench, BalloonTester.java that reads in balloons.train.arff, just to double check), which is in script1.py. I wrote a small Node class to be used for a binary tree, just so that the algorithm could be nice and elegant (recursive). I calculate the entropy as expected, and find the attribute (the right index of my list of instances), and then "split."

   I do this by stripping my data into two, one with only elements of one class of my split attribute, and the other with only elements of the other class of the split attribute. Thankfully, there were only 2 classes per feature, so I could just model it as a binary tree. Then I set the left Node of my current root Node to the algorithm passed in with the first list, and the right Node to the algorithm passed with my second list. My base cases are when I've stripped all the attributes, or when the calculated entropy is 0, so I can just set the outgoing Nodes to be True or False.

   I used a nearly identical process with the second heuristic, only I changed the information gained logic inside the function.

2b. I implemented SGD in the given shell SGD.java, and evaluated it with SGDTester.java. This is the algorithm that took me the longest – I found myself tweaking the parameters for a long time without any results. Most noticibly, the α value is incredibly sensitive. Anything too big (yet still less than 1), the algorithm doesn't converge, and I get -Infinities populated throughout my weight vector. Anything too small, and I consistently misclassify when it comes to evaluating the algorithm. But I made a few changes to the baseline algorithm that greatly improved my accuracy with SGD. First off, I ran it on the same dataset 100 times. This had a marginal improvement. But then, with each iteration, I shuffled the data. This seemed to have a really good effect, and I went from the 70% range, where the function $f(x) = False$ was, and I got into the range that I have now. For the final evaluation, I just picked an arbitrary seed value for my random number generator (used in the shuffle) so that I don't constantly get indeterministic results.

3e. This program also took a lot of time, but not because I was tweaking it, but mostly dealing with Weka. But aside from the given parameters of the assignment, I experimented a bit with this algorithm, mostly because I was disappointed by the

lack of marked improvement. I tried SGD on linear combinations of unbounded ID3 trees, since they were my most successful algorithm. The results were so incredible that I'm not sure if I messed up somewhere in the code. It gave me on the 5-fold cross-validation : 395/400, 397/400, 397/400, 394/400, 392/400, giving an overall $p_a = 98.75\%$. This has a confidence interval of [0.981,0.994]. I've included the outputted data on badges.train in 2.e.experiment.pred, if you'd like to evaluate it for legitimacy.