Report of project 3

Fengx463@umn.edu

1. **The implementation of the decision tree**

   This code uses Hund's algorithm and build the decision tree recursively.
   After reading in the data, **a reduction process is applied. The columns with all zeros are abandoned**. Then I create a global table to assign an image id for each image and another global table to assign an attribute id for each unreduced attribute, as well as a global label table to look up the label.

   Next, a decision tree is built recursively. At each node, first there is an exhaustive search for the best split attribute and best split value of that attribute. Then the key of this node is set to be a best-split-attribute, best-split-value pair. Then split the data into left and right groups and eliminate the selected attribute from the attribute list. Then pass the data and unused attribute list to the left and right child. The termination condition includes three cases: (a) when it runs out of the attribute. (b) when all the passed in data are of the same label. (c) when the passed-in data size is below the user set minimum frequency. When the termination criterion is triggered, the node is denoted as leaf node and the key of this leaf node is the class, that is decided by the majority vote.

   Finally, the decision tree is written into file by serializing it in pre-order. The format I chose to store the serialized tree is the following. Each node value is a (attribute_id, split) pair, delimited by '\n'. The leaf node is a single value of the class label and is marked by length being one. When reading this tree from the file, when a length-one value is read in, the corresponding node will be denoted as leaf and the recursion will return. **Note that since there is a reduction process to the attribute, the attribute numbers don't correspond to original unreduced attributes. So in the written out tree file, you may see that the best-split attribute-id may not be the same as the original unreduced attribute ids.**

   The key step in the decision tree building above, is the search for the best split attribute and best split value. This is done by looping over all the unused attributes and loop over the candidate values. A brute force way is of O(n^2) time complexity. But I first sort the candidate values of an attribute, and when scanning over the values, the count only changes by one at each step. At each split value, the number of the left and right class is obtained, based on which the impurity of this split can be calculated. Then the one with the smallest impurity is selected as the best split attribute and split value on this node. The impurity metric used here is Ginni coefficient. $\mathrm{Gini} = 1 - \sum_i p(i|t)^2$ (which seems to the variance of Multinomial distribution).

2. **The result and the analysis.**

Decision Tree					
	minfreq =	1	5	10	20
Rep 1	training accuracy	1	0.983717	0.968117	0.94695
	testing accuracy	0.8753	0.8742	0.8771	0.8733
Rep 2	training accuracy	0.972567	0.950383	0.929583	0.903483
	testing accuracy	0.8229	0.824	0.8217	0.8186

Looking at the decision tree data, test accuracy, although below training accuracies, are not too different from them. So the overfitting problem is not significant. Rep2 accuracy is generally worse than rep1 accuracies, which makes sense because rep2 is the reduced data by PCA which loses some information.

As minfreq increases from 1 to 20, the test accuracy doesn't decrease very much for both rep1 and rep2. This result shows that near the leaf nodes, the remaining attributes already don't contain too much information that can help the classification of the data. **So whether or not to do further splitting after minfreq reach 20 doesn't make too much difference to the classification problem.**

3. **The implementation of the random forest.**
The random forest is built on top the decision tree induction algorithm above. First 40% original size of sample is selected randomly with replacement. Then build the tree based on this sample. This process is repeated 100 times and 100 trees is built, thus forming a forest. Then given a test data, it's classified by each tree and the final classification is decided by the majority vote all these trees. This process is written in rf.py whose output is prediction files named in a format like "pred%03d". The majority vote, i.e. the merging of these decision trees is done in rfmerge.py

Radom Forest					
	minfreq =	1	5	10	20
Rep 1	training accuracy	0.995167	0.989067	0.97875	0.971433
	testing accuracy	0.9458	0.9465	0.9347	0.937
Rep 2	training accuracy	0.978367	0.961033	0.948283	0.929633
	testing accuracy	0.927075	0.927675	0.923175	0.916475

4. **Analysis of Random Forest**
We can see from the table above that the performance of random forest is much better than a single decision tree. All the accuracies are above 90%. Also the features and the trend analyzed in section 2 still holds here.