UNIVERSITY OF ILLINOIS AT URBANA CHAMPAIGN

CS 412 Homework 4

Nikolaos Kontakos
UID: kontako2

December 8, 2017

CLASSIFICATION METHODS USED

INTRODUCTION

For Assignment 4, there are two classification methods used to predict labels for test data sets: a basic classification method, decision trees, and an ensemble classification method, random forests. The decision tree utilizes a Gini Impurity measure in order to evaluate the best attribute to split a tree node on. The random forest utilizes bootstrap aggregation to sample the original testing data set and random attribute selection to determine best splits.

DECISION TREE

The basic classification method used is the decision tree. This classifier is built based on a testing data set; every node in the decision tree consists of one attribute that is fully split into its k children. In other words, every value of the attribute branches to a different child node. The attribute itself is selected with the Gini Impurity measure. Out of all possible attributes that remain, the measure selects the attribute that provides the largest reduction in impurity. The Gini formula is given below:

$$gini(D) = 1 - \sum_{i=1}^{n} (p_i^2)$$

$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

DECISION TREE IMPLEMENTATION]

The decision tree is completely implemented in DecisionTree.py. It consists of one main class, Node, which keeps track of each node in the decision tree, its attached attribute, all its kids, whether it is a leaf node, its class label (if a leaf node), and the value of its parent attribute.
The implementation:

1. Extract the data from both train and test files to separate lists.
2. Create an attribute list that holds all attributes present in the data; this is utilized in generating the decision tree and keeping track of which attributes are used for full splits.
3. Generate a decision tree using the training data list and the attribute list; it closely follows the recursive implementation in Chapter 8 of the textbook. It recursively constructs tree nodes using partitioned data from the original training data set; it selects the optimal attribute for full splits on the node with Gini measure.
4. Once the decision tree is created, the test data is ran against the tree in order to predict class labels for each data tuple. This is done by recursively going through the data tree

with the tuple; it checks the attribute on the current node against the attribute in the test tuple in order to go down the correct branches. Once it hits a leaf node, it will obtain the tuple's predicted class label. Repeat for each tuple.

5. Calculate the confusion matrix based on the actual and predicted class labels.

RANDOM FOREST

The random forest is completely implemented in RandomForest.py. It consists of the same class utilized for DecisionTree.py, the node class. It is implemented utilizing bootstrap data aggregation to generate sample training data sets, and random attribute selection from all attributes to optimize gini splits for nodes.

RANDOM FOREST IMPLEMENTATION

The implementation:

1. Extract the data from both train and test files to separate lists.

2. Generate a random forest tree using the training data set and a hard-coded number of trees to construct; it closely follows the bagging algorithm implementation in Chapter 8 of the textbook. For each tree the algorithm must construct, it randomly samples the training data set to construct a new data set that is 63.2 percent of the original size. Then, it creates an attribute list based on attributes present in this bootstrapped data set, and generates a decision tree with the data set and attribute list. It closely follows the decision tree implementation in DecisionTree.py.

3. When determining the optimal attribute to fully split a node, the algorithm randomly picks a number of attributes to evaluate from all given attributes. If number of attributes is D, then algorithm randomly selects square root of D attributes to evaluate on. This introduces randomness and also speeds up calculation of gini splits.

4. Once the random forest is created, the test data is ran against each individual tree in the random forest, similar to the decision tree implementation. Once all trees have given their predictions for a data tuple, majority voting is utilized to choose the final predicted class label. Repeat for each tuple.

5. Calculate the confusion matrix based on the actual and predicted class labels.

MODEL EVALUATIONS

	Test Set			Training set		
Class Label	1	2	3	1	2	3
Overall Accuracy	0.6444			1.0000		
Precision	0.0000	0.7248	0.7021	1.0000	1.0000	1.0000
Recall	0.0000	0.7745	0.6535	1.0000	1.0000	1.0000
Specificity	0.8916	0.7561	0.7742	1.0000	1.0000	1.0000
Accuracy	0.8044	0.7644	0.7200	1.0000	1.0000	1.0000
F-1	0.0	0.7488	0.6769	1.0000	1.0000	1.0000
F-0.5	0.0	0.7342	0.6918	1.0000	1.0000	1.0000
F-2	0.0	0.7640	0.6627 6	1.0000	1.0000	1.0000

Table 1: Balance Scale Decision Tree

	Test Set			Training set		
Class Label	1	2	3	1	2	3
Overall Accuracy	0.7733			1.0000		
Precision	0.0	0.7748	0.7719	1.0000	1.0000	1.0000
Recall	0.0000	0.8431	0.8713	1.0000	1.0000	1.0000
Specificity	1.0000	0.7967	0.7903	1.0000	1.0000	1.0000
Accuracy	0.9022	0.8178	0.8267	1.0000	1.0000	1.0000
F-1	0.0	0.8075	0.8186	1.0000	1.0000	1.0000
F-0.5	0.0	0.7875	0.7899	1.0000	1.0000	1.0000
F-2	0.0	0.8285	0.8494	1.0000	1.0000	1.0000

Table 2: Balance Scale Random Forest

	Test Set					Training set				
Class Label	1	2	3	4	5	1	2	3	4	5
Overall Accuracy	0.9745					1.0000				
Precision	0.9647	0.6966	0.9862	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Recall	0.9593	0.7769	0.9805	1.0000	0.0	1.0000	1.0000	1.0000	1.0000	1.0000
Specificity	0.9829	0.9907	0.9937	1.0000	0.9994	1.0000	1.0000	1.0000	1.0000	1.0000
Accuracy	0.9751	0.9850	0.9895	1.0000	0.9994	1.0000	1.0000	1.0000	1.0000	1.0000
F-1	0.9620	0.7345	0.9833	1.0000	0.0	1.0000	1.0000	1.0000	1.0000	1.0000
F-0.5	0.9636	0.7113	0.9851	1.0000	0.0	1.0000	1.0000	1.0000	1.0000	1.0000
F-2	0.9604	0.7594	0.9816	1.0000	0.0	1.0000	1.0000	1.0000	1.0000	1.0000

Table 3: Nursery Decision Tree

	Test Set					Training set				
Class Label	1	2	3	4	5	1	2	3	4	5
Overall Accuracy	0.9688					0.9999				
Precision	0.9381	0.9130	0.9715	1.0000	0.0	1.0000	1.0000	0.9996	1.0000	1.0000
Recall	0.9687	0.4846	0.9772	1.0000	0.0	0.9996	1.0000	1.0000	1.0000	1.0000
Specificity	0.9688	0.9987	0.9868	1.0000	1.0000	1.0000	1.0000	0.9998	1.0000	1.0000
Accuracy	0.9688	0.9850	0.9838	1.0000	1.0000	0.9999	1.0000	0.9999	1.0000	1.0000
F-1	0.9531	0.6332	0.9744	1.0000	0.0	0.9998	1.0000	0.9998	1.0000	1.0000
F-0.5	0.9441	0.7759	0.9727	1.0000	0.0	0.9999	1.0000	0.9997	1.0000	1.0000
F-2	0.8827	0.0	0.6571	0.9347	0.0	0.9997	1.0000	0.9999	1.0000	1.0000

Table 4: Nursery Random Forest

	Test Set		Training set	
Class Label	1	2	1	2
Overall Accuracy	0.8589		0.8596	
Precision	0.7705	0.8988	0.7708	0.8986
Recall	0.7749	0.8966	0.7696	0.8992
Specificity	0.8966	0.7749	0.8992	0.7696
Accuracy	0.8589	0.8589	0.8596	0.8596
F-1	0.7727	0.8977	0.7702	0.8989
F-0.5	0.7714	0.8984	0.7706	0.8987
F-2	0.7740	0.8970	0.7698	0.8991

Table 5: LED Decision Tree

	Test Set		Training set	
Class Label	1	2	1	2
Overall Accuracy	0.8633		0.8596	
Precision	0.7865	0.8965	0.7708	0.8986
Recall	0.7664	0.9068	0.7696	0.8992
Specificity	0.9068	0.7664	0.8992	0.7696
Accuracy	0.8633	0.8633	0.8596	0.8596
F-1	0.7763	0.9016	0.7702	0.8989
F-0.5	0.7824	0.8985	0.7706	0.8987
F-2	0.7703	0.9047	0.7698	0.8991

Table 6: LED Random Forest

	Test Set				Training set			
Class Label	1	2	3	4	1	2	3	4
Overall Accuracy	0.4780				0.9943			
Precision	0.4885	0.4521	0.4802	0.4869	0.9945	0.9947	0.9961	0.9920
Recall	0.4776	0.4041	0.5216	0.5098	0.9918	0.9974	0.9948	0.9933
Specificity	0.8169	0.8411	0.8294	0.8161	0.9982	0.9982	0.9987	0.9973
Accuracy	0.7260	0.7340	0.7580	0.7380	0.9967	0.9980	0.9977	0.9963
F-1	0.4830	0.4267	0.5000	0.4981	0.9932	0.9960	0.9954	0.9926
F-0.5	0.4863	0.4416	0.4879	0.4913	0.9940	0.9952	0.9958	0.9922
F-2	0.4798	0.4128	0.5127	0.5051	0.9923	0.9968	0.9951	0.9930

Table 7: Synthetic Social Decision Tree

	Test Set				Training set			
Class Label	1	2	3	4	1	2	3	4
Overall Accuracy	0.7420				1.0000			
Precision	0.4101	0.4184	0.4190	0.3449	1.0000	1.0000	1.0000	1.0000
Recall	0.3321	0.2408	0.3793	0.5843	1.0000	1.0000	1.0000	1.0000
Specificity	0.8251	0.8914	0.8411	0.6201	1.0000	1.0000	1.0000	1.0000
Accuracy	0.8580	0.8780	0.8860	0.8620	1.0000	1.0000	1.0000	1.0000
F-1	0.3670	0.3057	0.3982	0.4338	1.0000	1.0000	1.0000	1.0000
F-0.5	0.3917	0.3646	0.4104	0.3757	1.0000	1.0000	1.0000	1.0000
F-2	0.3452	0.2632	0.3866	0.5131	1.0000	1.0000	1.0000	1.0000

Table 8: Synthetic Social Random Forest

CHOSEN PARAMETERS

DECISION TREE

DEPTH STOPPING MEASURE = 10 LEVELS

My decision tree implementation only utilizes one hard-coded parameter: a depth limit. This is a stopping measure that is set to 10 levels; it is primarily for the synthetic social dataset for performance and to prevent over-fitting. At 10 levels, the algorithm is much faster and sacrifices very little accuracy for the synthetic data set; only 2 percent difference.

RANDOM FOREST

RANDOM ATTRIBUTE SELECTION SIZE

In the random forest implementation, my algorithm utilizes random attribution selection to optimize full split (using Gini split) on decision tree nodes. This implementation utilizes a size of $\sqrt{d}$, where d is the number of all attributes present in the data set. This allows further randomization amongst the decision tree and their nodes.

DEPTH STOPPING MEASURE = 10 LEVELS

Similar to the decision tree implementation, my algorithm also implements 10 levels as a hard-coded stopping measure for building the decision tree. I utilize 10 levels for reasons similar to my decision tree algorithm.

SAMPLE SIZE = 0.632

When utilizing bootstrap aggregation to sample a new, randomized training set from the original data set, the random forest algorithm utilizes a sample size of 63.2 percent. The implementation follows the .632 bootstrap method mentioned in the book, in which 63.2 percent of the original data tuples "will end up in the bootstrap sample" and the remaining 36.8 percent will not be selected to train on. This also allows the model to shrink the size of the training sets it utilizes, while not being small enough to cause too much variation amongst individual decision trees. In addition, the sample is not large enough to cause the random forest to be similar to the decision tree.

NUMBER OF TREES

For the random forest implementation, the program hard codes the number of trees to utilize for each specific data set. Balance.scale uses 500 trees, nursery uses 50 trees, led uses 100 trees, and synthetic.social uses 100 trees. The main reason I utilize a few trees for datasets with many attributes and tuples, such as synthetic.social, is to improve the speed of the program and make it run within 3 minutes. In addition, beyond 100 trees, the performance of synthetic social does not necessarily increase; only the variance in metric evaluation decreases slightly. In this implementation, it is not worth building more than a 100-tree random

forest for nursery and synthetic social; the increase in accuracy and other metrics does not justify the subsequent increase in runtime.

CONCLUSION

My random forest ensemble method improves the performance of the decision tree basic classification method (for all metrics), on all sets but nursery. This is most likely due how accurate the original decision tree was at predictions for the nursery tuples; the random forest did slightly worse due to its use of random attribute selection and bootstrap aggregation.

BALANCE SCALE

Random forests greatly improves the performance of this data set. The overall accuracy of the testing set was improved from 64.44 percent to 77.33 percent, a 12.89 percent improvement. The accuracy especially improved for class labels 2 and 3; it shows the random forest is better at recognizing tuples of different classes. The precision and recall both increased drastically for class labels 2 and 3; as a result, the random forest is better at predicting class labels for imbalanced data sets (since there are very few test tuples with class label 1). This is further shown by the improvement in F-1 and F-Beta scores. The specificity increases greatly, showing that the random forest improves at correctly labeling negative tuples.

NURSERY

Random forests slightly reduces the performance of models for this data set. The overall accuracy of the testing sets went down from 97.45 percent to 96.88 percent. Accuracy, specificity, recall, and the F measures all decrease; this is most likely due to the random forests having randomization in its algorithm (attribute selection and bagging). Since the original decision tree already performs well on the nursery data set, the random forest is not needed.

LED

Random forests slightly improves the performance for this data set. The overall accuracy of the testing set was slightly improved in random forests, from 85.89 to 86.33 percent. Random forest is slightly better at classifying tuples. Since there are only two labels, there are a lot of tie votes; my algorithm breaks the tie by choosing the label that shows up first in the list. This makes the performance of random tree vary on the LED test, based on attribute selection. Consequently, either decision tree or random forest can be implemented for significant results.

SYNTHETIC SOCIAL

Random forest does drastically improves the performance for this data set. The overall accuracy improves from 47.80 to 74.20 percent with random forest implementation, a 26.4 percent increase. Specificity and accuracy greatly; in other words, the random forest is much better at making overall predictions and at identifying true negative test tuples than the decision tree. However, the random forest does worse in precision, recall, and the F measures; as a result, the random forest did not identify positive tuples well. This is most likely due to the stopping

criteria being 10 levels; there were not enough branches in the decision trees for the random forest to make strong decisions in its predictions.