Project Extension 2: Random Forest

Alison Reed
Tony Maricic

December 2023

1 High Level Overview of Our Extension

For the second part of our project, We chose to build a decision tree and extend it with a random forest. A decision tree finds the most significant features in the model, stores them, and divides the data by them. In order to build the decision tree, we sorted the data set by each feature, tested the entropy of dividing the data set at every point in each feature where the label changed, and then actually divided the data set by the feature and threshold that resulted in the greatest information gain ($max(H(S) - (H(S^\ell) + H(S^r)))$). If the split resulted in a set with entropy of 0 or extended the tree to the maximum level, we created a leaf. Otherwise, we repeated the process above. Once our decision tree was complete, we classified test examples by traversing the tree and outputting the label of the resulting leaf.
To build a random forest, we created 100 decision trees using a random subset of training examples and features. Using random features decreases the correlation between the decision trees. This makes the model rely on a wider, more diverse array of features so if we have an anomalous input, it is less likely to sway the model. Thus the model is more resilient to outliers. Random Forests also mitigate over-fitting by taking the majority vote of the results of individual trees, which by themselves may over-fit the data.

2 Breaking Down The Code

Decision trees can handle unscaled data so we were able to skip straight to the decision tree implementation! We coded an entropy function -$\Sigma_C P^C log_2 P^C$. We created two tree node classes: one for internal nodes which stored a feature, threshold, and left and right child and one for leaves which stored a label and the set of training examples which had been filtered to that leaf. We then recursively created the decision tree:

1. if the data set has entropy of 0 or if we have reached the max depth of the tree, create a leaf
2. otherwise, search for the best split (i.e. the data split that maximizes information gain) by sorting the data by each feature and testing the entropy of all potential splits
3. create an internal node which stores the best split's feature and threshold
4. recursively create the subtrees rooted at the left and right child of the internal node.

Then predict by iterating through the test examples, starting at the root of the tree for each example and traversing the tree until coming across a leaf whose label is the prediction!

For the random forest, create m decision trees using N training examples randomly chosen with replacement and $\sqrt{d}$ random features as this is a classification model. We stored the features we used when building each tree because we pass in the whole original data set for prediction but need to limit the prediction features of each tree to the features it was trained on. Then we return the majority label predicted by the trees.

3 Results

Implementation	Data Set	Decision Tree / Random Forest	Maximum Depth	# of Trees	Accuracy
Scikit-learn	MNIST digits	Decision Tree	None	1	82.89290681502086%
Scikit-learn	MNIST digits	Random Forest	None	100	96.105702364395%
From Scratch	MNIST digits	Decision Tree	7	1	81.22392211404729%
From Scratch	MNIST digits	Random Forest	7	100	92.76773296244785%
From Scratch	KMNIST Japanese Characters	Decision Tree	7	1	40%
From Scratch	KMNIST Japanese Characters	Random Forest	7	100	68.92857142857143%

Figure 1: In all implementations, the accuracy of the random forest outdid that of the singular decision tree by over 10 percent. The accuracy of the predictions on the Japanese character data-set is much lower than the accuracy of the predictions on the digits data-set because the Japanese character images had 784 features (compared to the digits' 64 features) so, in this case, giving the trees a maximum depth of 7 probably led to under-fitting.

Hyper-parameters:
Limiting the maximum depth of the tree reduces the likelihood of over-fitting, improves run time, and reduces error due to noise. We were hoping to match the results of the decision tree shown in the lecture slides so we set max depth to 7. A max depth of 7 worked well on the digits data-set but probably led to under-fitting in the Japanese character data-set. Increasing the number of decision trees in the random forest will increase accuracy but with diminishing returns. Again, as we aimed to match the lecture slides, we set the number of decision trees equal to 100.