CS6220 HW3 Option 1.2: The Power of Random Forest

Introduction

The code, screenshots, and artifacts are all contained in this Github repo.

Dataset

I decided to use the Bank Marketing Dataset from the UCI Machine Learning Repository. It can be found here. The goal is: given the dataset, predict whether or not a client will subscribe to a term deposit.

The dataset has 11162 total instances. It has 17 attributes (including the target variable):

Categorical

1. Job : What the person does, i.e., admin,technician, services, management, etc.
2. Marital : marital Status, i.e., married, single, divorced
3. Education: Highest education secondary, tertiary, primary, unknown
4. Default : Whether or not the person has housing
5. Housing : Whether or not the person has housing
6. Loan : Whether or not the applicant has taken out a loan from the bank.
7. Deposit : yes, no (Dependent Variable)
8. Contact : Type of contact provided (unknown, cellular, telephone)
9. Month
10. poutcome: unknown, other, failure, success

Numerical

1. Age: age of the person
2. balance: current bank balance
3. day:
4. duration
5. campaign
6. pdays
7. Previous

I decided to use an 80-20 split for the training and test set. Therefore, the training set has 8929 examples and the test set has 2233 examples.

Here are 5 concrete training/test examples from the dataset:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	deposit
0	59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	1042	1	-1	0	unknown	yes
1	56	admin.	married	secondary	no	45	no	no	unknown	5	may	1467	1	-1	0	unknown	yes
2	41	technician	married	secondary	no	1270	yes	no	unknown	5	may	1389	1	-1	0	unknown	yes
3	55	services	married	secondary	no	2476	yes	no	unknown	5	may	579	1	-1	0	unknown	yes
4	54	admin.	married	tertiary	no	184	no	no	unknown	5	may	673	2	-1	0	unknown	yes

Decision Tree and Random Forest Implementations

For both the decision tree classifier and random forest classifier, I decided to use sklearn's implementation. They can be found here:

1. **Decision Tree:** https://github.com/scikit-learn/scikit-learn/blob/844b4be24/sklearn/tree/_classes.py#L639
2. **Random Forest:** https://github.com/scikit-learn/scikit-learn/blob/844b4be24/sklearn/ensemble/_forest.py#L1064

The decision tree implementation uses an optimized version of the CART (Classification And Regression Tree) algorithm. It is very similar to the C4.5 algorithm, but improves upon by supporting numerical target variables (which we use in our case) and does not compute rule sets. It constructs the tree using the feature-threshold combination that gives us the largest information gain at each node.

The Single Decision Tree has the following hyperparameters:

Hyperparameter	**Value**
Split Criterion	Gini Impurity
Spit Selection	Best (choose the best split, i.e., the split with the highest information gain according to the criterion.)
Max Tree Depth	6
Min. Samples to Split an internal node	2
Min. Samples required to be a leaf node	1

We trained the single decision tree on the Bank Marketing dataset and obtained the following results:

- **Training Time:** 0.054s
- **Train Accuracy:** 80.81%
- **Test Accuracy:** 78.01%

The Random Forest has the following hyperparameter configuration:

Hyperparameter	**Value**
Number of Estimators (Trees)	100
Split Criterion	Gini Impurity
Max Tree Depth	6
Min. Samples to Split an internal node	2
Min. Samples required to be a leaf node	1

We trained the random forest consisting of 100 decision trees on the Bank Marketing dataset and obtained the following results:

- **Training Time:** 0.366s
- **Train Accuracy:** 81.76%
- **Test Accuracy:** 80.03%

The top 10 best performing trees in the random forest have the following test accuracies:

```
1: 0.7908643081056874
2: 0.790416480071652 4
3: 0.7890729959695477
4: 0.7819077474249888
5: 0.7801164352888491
6: 0.7783251231527094
7: 0.7774294670846394
8: 0.7756381549484997
9: 0.77384684281236
10: 0.7707120465741155
```

The top 10 worst performing trees in the random forest have the following test accuracies:

```
1: 0.6318853560232871
2: 0.6318853560232871
3: 0.6336766681594268
4: 0.639050604567845 9
5: 0.6426332288401254
6: 0.643976712942230 1
7: 0.6444245409762651
8: 0.6502463054187192
9: 0.6538289296909987
10: 0.6569637259292431
```

Analysis

The table below compares the training time and the test accuracy for the single decision tree and the random forest models:

Model	**Training Time**	**Test Accuracy**
Single Decision Tree	0.054s	78.01%
Random Forest w/ 100 Decision Tree Estimators	0.366s	80.03%

It is helpful to understand the differences in the actual algorithm for both models: the single decision tree simply trains itself on the dataset and improves its prediction capability by choosing the best split (the one with the highest information gain) at each step. Naturally, this leads to better and better split thresholds as the decision tree is exposed to more examples. The random forest consists of a collection of decision trees (hence the name “forest”.) The intuition behind a random forest is as follows: a single decision tree, while an excellent algorithmic choice for pattern recognition, is seen as a “weak learner”. In other words, the classification of each example is based on a single tree’s ‘opinion’. The random forest eliminates the weakness of a single opinion by training multiple decision trees on different sub-samples of the dataset and averages the ‘opinion’ of each tree for a particular classification. Subsamples are used instead of the entire dataset to prevent any one tree from overfitting and making noisy classifications.

Notice above that the training time is nearly 6x higher for the random forest classifier. However, given how small the magnitude of the training time is for the single decision tree, we can ignore the scaling factor and not overthink the difference in training times; it does make sense that 100 decision trees trained on subsamples of the dataset would take significantly longer than a single decision tree trained on the whole dataset. Now, we address the most important factor: *the test accuracy*. **The random forest model outperforms the single decision tree by 3.02%**. While this may seem like an underwhelming improvement even after using 100 decision trees, there are a few reasons this may have occurred:

1. Firstly, the random forest was expected to outperform the decision tree. While there is no concrete explainability, the intuition is that the random forest's classification is a *weighted* 'opinion' of multiple decision tree classifiers. Naturally, a set of 'weak learners' come together to make a 'strong learner' that is able to represent more nuances in the dataset.
2. From a mathematical standpoint, the random forest is expected to reduce the variance of the classification error (not much difference in the bias). Therefore, when the models are exposed to an unexpected test dataset (examples that have never been seen before), random forests are expected to have a lower number of errors in classification.
3. Note that the complexity of the decision trees in the random forest are the same as the single decision tree (to ensure more interpretability for the results.) Given this, note that *some of the classifiers perform extremely well* (slightly better than the single decision tree) *and are thus weighted more than the others that performed poorly.* The final classification is a weighted 'opinion' of these strong and weak classifiers. Given that the strong ones are slightly better than the single decision tree, the accuracy is pulled up. Similarly, the accuracy is pulled down by some of the worse-performing classifiers providing incorrect classifications for certain examples. This is the reason why the random forest does not perform significantly better, but just marginally better than the single decision tree.

Possibility for Further Improvements

A natural question that arises with random forest is whether we can improve the performance by adding more trees. I ran an experiment by increasing the number of estimators and keeping all other parameters the same. The results are tabulated below:

#Estimators	**Training Time**	**Accuracy**
100	0.366s	80.03%
150	0.526s	79.76%
200	0.688s	79.85%

We see a surprising trend: *the accuracy of the models slightly decreases by increasing the number of estimators.* Note that the accuracy is averaged over 5 training runs to avoid any one-off events due to the randomness of the model algorithm. Some of the reasons this could've happened was:

1. Increasing the number of estimators means more estimators trained on a particular subsample of the dataset. It could just be that we introduced a greater number of underperforming estimators that pull down the test accuracy slightly by providing more weight to the incorrect label for example.
2. Adding more estimators could've skewed the weighting scheme towards the underperforming estimators if said estimators made a few correct predictions but got most of the following predictions incorrect.

We can infer the following: *adding more estimators to the random forest model does not guarantee an improvement in the model accuracy*. The estimators that comprise the model may be severely underperforming and end up pulling the accuracy down. While this may be surprising from a superficial look, the results are understandable when we look under the hood. It is analogous to general public opinion: if you end up adding too many people to a debate, the opinions become convoluted and skewed and we end up with a conclusion that is riddled with unnecessary noise.