CS 334 — Homework 2 Solutions

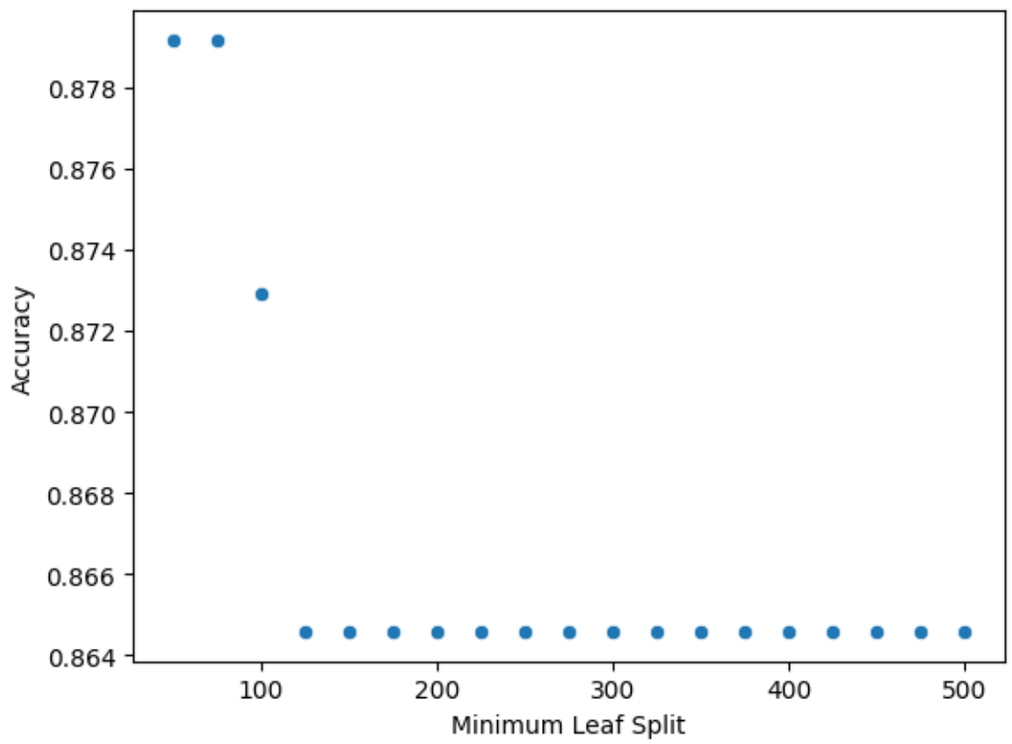
Jack Anstey

Problem 1: Decision Tree Implementation

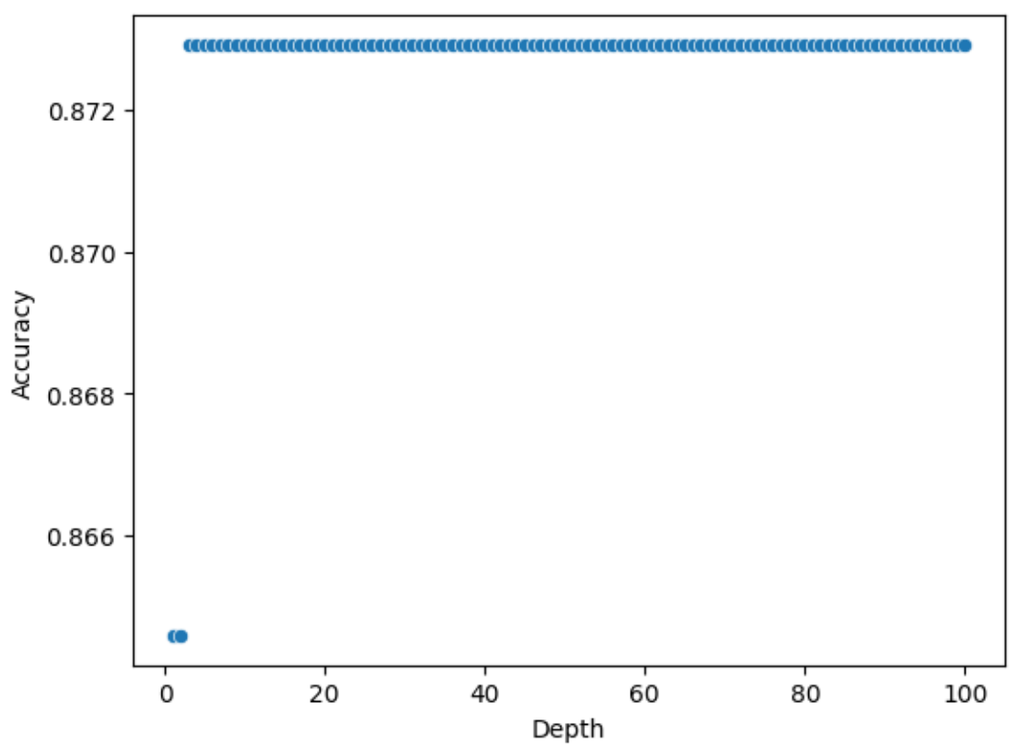
a) See dt.py

b) See dt.py

c) For this graph, I chose a constant maximum depth of 4 and a minimum leaf sample of 100 ranging from 50 to 500 with 25 step increments:



For this graph, I chose a constant minimum leaf sample of 100 and maximum depth ranging from 1 to 100:



d) The computational complexity of my decision tree training and prediction algorithms begin by the definitions of its inputs. First, the prediction algorithm is run using xFeat and y, two dataframes with sizes $n*d$ and $n*1$ respectively. The algorithm first checks the ending criterion, but since we are calculating complexity, we will assume the worst case which is that every single leaf goes to max depth, which I will refer to as m.

After some short steps where we edit xFeat to add y to it, we start by iterating through all of the features in xFeat (sans the label), so we use this for loop d times. Within this loop, we go from min leaf sample to the n-(min leaf sample), so in the worst case this is n. In this loop, we start by sorting xFeat by the feature, which for built in sorting algorithms will take $n * log(n)$ time.

We then do some linear comparisons, so the ordering dominates and we don't have to do too much digging here. Post for loops we sort one last time, set some variables, then return recursively.

This leads us to our complexity calculation with a overall time complexity of $O(m * ((d * (n * (n * log(n)))) + n * log(n)))$. Once we simplify, we get a complexity of $O((m * d * log(n)) * (n^2 + 1))$

Problem 2: Exploring Model Assessment Strategies

a) See q2.py

b) See q2.py

c) See q2.py

d) The results of the AUC and time each method took are included in this table:

	Strategy	TrainAUC	ValAUC	Time
0	Holdout	0.950084	0.636825	0.017015
1	2-fold	0.955713	0.774247	0.026023
2	5-fold	0.952963	0.794554	0.062057
3	10-fold	0.954355	0.792352	0.133121
4	MCCV w/ 5	0.954888	0.753512	0.026023
5	MCCV w/ 10	0.959132	0.911041	0.045041
6	True Test	0.952502	0.809935	0.000000

As we hoped, the Training AUC for every single strategy is very good and all hovering around 95%. ValAUC and Time, therefore, tell us a more complete story on the level of robustness of the estimates.

Holdout, as expected, is the worst, not only in comparison to the "True Test" values, but to literally every other strategy. Sometimes, simple does not mean better.

All of the folds immediately provide better ValAUC values, all above 77% which is pretty good! 5-fold provides the best result, while 10-fold is a close runnerup. While doing some number of folds above 10-fold may yeild a better ValAUC, the value of doing so is pretty low. The time it took 10-fold to complete is already more than double that of 5-fold, and would become even worse for a greater numebr of folds. Therefore, I would consider 5 folds the optimal number here for both ValAUC and for the time it takes, especially in comparison to the "True Test" ValAUC.

Monte Carlo Cross-Validation, while being more complicated to implement, provided great results. 5 samples was worse than 2-fold, but 10 samples was better than literally everything, including the "True Test" ValAUC! As we saw in the slides, this is a little suspect. My assumption is that overtraining could be occurring here (where we have lowered bias but increased variance), but we can only figure this out definitely through more test datasets. 10 sample MC-CV also provides the best time (not including True Test), so from this sample of choices I would choose MC-CV 10 sample to be to superior strategy.

Problem 3: K-NN Implementation

a) Using GridSearchCV, I tested a large variety of hyperparameters for both KNN and Decision Tree classifiers.
For KNN, I chose a variety of k, ranging from 1 to 9 skipping all even numbers (as even numbers only provide issues and little benefit). I also chose to try multiple different distance metrics, which included euclidean and manhattan distance. After computation, the optimal k was found to be 9 and the optimal metric was manhattan distance.
For the decision tree, I chose a wide breadth of hyperparameter values. I tried a maxdepth for the decision tree with a range of 1 to 9 and a min leaf samples range of 50 to 300. The optimal maxdepth is 3 and the optimal minimum leaf sample is 63.

b) See q3.py

c) See q3.py

d) The optimal hyperparameters and AUC/Acccuracy utilizing different sized training datasets is as follows:

```
KNN optimal k: 9
KNN optimal metric: manhattan
Decision Tree optimal max depth: 3
Decision Tree optimal minimum leaf sample: 63

  Training Set Portion   KNN AUC  KNN Accuracy    DT AUC  DT Accuracy
0        Full Training  0.819907      0.877083  0.835700     0.887500
1         95% Training  0.811437      0.875000  0.863207     0.887500
2         90% Training  0.785246      0.872917  0.843003     0.883333
3         80% Training  0.795959      0.866667  0.832660     0.870833
```

Neither KNN or Decision Tree were found to be particularly sensitive when reducing the size of the training dataset. The AUC for KNN fell slightly when using a training dataset that was reduced, but only yielded a difference of less than 4% and was just under 80% in the worse case scenario.
A similar story was found for the AUC for the decision tree. It a much more consistent AUC percentage, but some strange results were had when using 95% and 90% of the training dataset to train the model. The AUC actually increased, instead of falling. It was within 2% of the original AUC value, so I'm not too concerned. My assumpition is that it's due to random chance, rather than the model actually being better with less training data.

Somehow, the accuracy was even less sensitive. For both KNN and Decision Tree classifiers, the accuracy stayed the same for 2 significant figures when using 95% and 90% of the original training data(!). Even though only using 80% of the training data did deviate more than the other training sets, it was still all within a 1% margin of difference. It is clear from not only the lack of difference of the AUC values, but the resolute steadfastness of the accuracy for both classifiers that the optimal parameters chosen are actually optimal.