ATTENTION!!

I've sent this as both an email and a canvas message but I want to be triply sure that you guys are aware of it so you don't think I'm trying to cheat. I created a dataset with the true labels of the training data so that I could continue to test against it while I was locked out from making more submissions on kaggle. I somehow ended up with a copy of it in with my other submissions and accidentally submitted it to the competition. Naturally, it scored a perfect 1.0. I have unchecked it to be used in my final submission but it is still showing on the leader board. I have not found a way to delete it but I want you guys to be aware that I should not be considered for the extra credit. I'm sure it is immediately obvious that something was fishy but I am just trying to cover my bases to be certain.

Thinking about it it probably got in the folder when I was testing it to make sure it was working.

Introduction

I chose to do the kaggle competition rather than the exploratory project. That being said, I'm assuming you don't need me to recap what the dataset is like I would normally do if writing a small research paper.

I used python's sklearn library to implement my machine learning since, as far as I know, nothing said this wasn't allowed. It definitely took a bit of learning to figure out how it worked and the format that they wanted things in. However, once I figured it out the process of trying out different models or tweaking the hyperparamaters was really easy and significantly better than if I was trying to make these adjustments to the library we're building via the homework.

I also want to say that I did not realize there was a restriction on the number of submissions to kaggle that we could do per day. If I had realized that I would not have waited until the final day to submit my attempts. I wasted 7 of my attempts on models that I knew would preform poorly before I realized it was limited to 10 per day.

I'm also not sure if we need to link it for this or if its just for the final project (or just for exploratory projects), but here is the github for my attempts on this project. `https://github.com/Tweal/CS6350FinalProject`

Data Cleaning and Preprocessing

I did initially try running without doing any data preprocessing other than one hot encoding the categorical variables. The results weren't great but they weren't nearly as bad as I expected, I saw about a 5% increase after cleaning the data. I'm assuming this is because only about 1% of the data has missing values.

As far as cleaning the data I took it in two steps, first was replacing the missing values and second was encoding the categorical variables. To replace the missing values I used sklearn's SimpleImputer and simply replaced missing values, denoted as '?' in this dataset, with the most frequent variable in the training dataset. I did experiment with instead replacing with the most common matching label for the training data but did not see a noticeable improvement in the results doing so.

I used two approaches to encode the categorical variables. I started out using panda's get_dummies to one hot encode them. This had the advantage of being extremely trivial to implement but I quickly discovered that it was less than ideal because I couldn't tell it what the values should be for each of the features so I encounted a problem where there was a value in the test dataset, specifically the *native.country* value of *Holand-Netherlands*, that was never seen in the training data and so sklearn gets a little unhappy when evaluating the test data. There was easy solution here to just hard code that column into the encoded data as 0 but I elected to go with the more robust solution of implementing it using sklearn's OHE module. This method has the advantage that you can pass it an array of categories and it will generate the dummy variables for values even if they are not found in the data. This meant that I could create a list of all of the values for each feature from both the training set and the test set and have them end up with the same columns after encoding. This did take quite a bit of finagling because I wanted the encoded data in a pandas dataframe for ease of viewing and manipulating. The problem with that is that sklearn returns a list of lists that I then had to manhandle back into a dataframe. However, once I figured it out I got it working and got basically the same results as when using pandas which is definitely good since it means that I did it correctly.

The other method of encoding that I tried was sklearn's OrdinalEncoder. This works as you'd expect and just assigns a distinct number to each option. The obvious advantages to this being that you don't end up with a hundred features and you can treat each feature as a numeric value. I implemented both of these methods because I wanted to see the difference they made in this dataset. It turns out it didn't make that big of a difference from OHE, the time it takes was lower but not enough for me to justify using one more than the other. It ended up being less than a 2% difference in each instance.

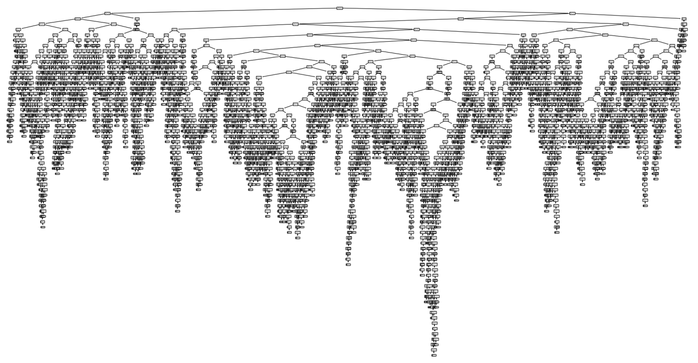
I also split the data 75:25 for training and test data once I realized that I couldn't use the kaggle submissions to test the provided test data. For the remainder of this paper I will refer to this split data as train and test with the dataset submitted to kaggle as final since that is how I referenced it in code.

Methods Implemented

For this first set of submissions I stuck with the things that have been covered in class, which might be why I wasn't able to break into the top ten. The top ten all being over 90% seems a bit crazy to me given that most of the blogs I've found talking about this dataset only get around 85%.

Despite knowing that it was going to be bad I started out with a regular decision tree. I tried it without depth limits, and as you'd expect it completely overfit the training data and got completely out of hand. That being said, it did make a cool picture when visualized, as you can see on the next page. The no max depth tree ended up with a test accuracy of 80.8% but for some reason this did not quite translate to the final dataset as it ended up of 74.5%. I further tried a regular decision tree with max depths of 10 and 5. Those ended up with test accuracies of 85.6% and 85.3% respectively, again these did not translate to the final dataset and ended up about 5% lower when submitted.

From there I moved on to the ensemble methods of random forest, adaboost, and bagged decision trees. For the random forest implementation I left it with the default of 100 trees and tried depths of 10, 5, and 1. The test accuracy were 86.1%, 85.3%, and 86.1% respectively. Depth 10 trees had a final accuracy of 75.0% and the depth 5 trees had 69.6%. The stumps preformed so poorly on the test data that i did not bother submitting to kaggle.



I ran AdaBoost with the default parameters of 10 learners with a learning rate of 1 as well as with 500 learners with a learning rate of 0.01. The default parameters resulted in a training accuracy 86.6% and a final accuracy of 77.9%, while the 500 learners and .01 learning rate had a test accuracy of 84.8% and final of 70.5%.

I ran the bagged decision trees with the default parameters, 10 trees with 100% sampling and all features, as well as with 500 trees and 50% feature sampling. The test results for those were 84.7% for default and 87.0% for the modified parameters. That ended up with final results of about 76% for both.

I'm not entirely sure why my final submissions are preforming so poorly vs my test submissions, I'm guessing it has to do with how I'm handling the missing data or unseen data. I really wish that I had started submitting them to kaggle earlier than the last day so that I would have known that the final data was performing so much worse than my training and test data and could have looked into it. Lesson learned for final submissions I suppose.

Going Forward

The first thing I plan on tackling going forward is finding out why my final results are so dissimilar to my training and test results. I'm guessing it has something to do with the data coming across things that it has never seen before in the final data so I might try running correlation matrices on the features to check about feature importance to see what I can drop as irrelevant.

Other than that I have three plans for the rest of the semester. The first is the easiest and that's to implement sklearn's KNNImputer method. As you can probably guess by the name it is a method of replacing misssing values by using the k nearest neighbors value. It requires that you have numeric data so it would require that I change up the order that I'm doing things a little bit but it shouldn't be hard.

The next thing that I want to play around with is tweaking the hyperparameters. Sklearn offers a method to iterate through options automatically but I'm not entirely sure how to use it so I will have to learn. I expect that I can probably squeeze another percent or so out of these methods with more optimal hyperparameters.

After that I want to try a few more advanced methods, namely I want to try doing it with an SVM and a neural network. I know a neural network is definitely overkill for this dataset but I still want to play around with it.