Course Project Report

1. Introduction

The real-world motivation that led me to choose this task was from herbalists who classify plants based on their attributes. I wanted to explore how machine learning algorithms could be used to automate this process.

I set out to investigate whether regression, perceptron, and decision tree models could be used to effectively classify plants based on attributes. I then wanted to compare the models and see which ones performed the best on the data.

My personal motivation was that I wanted to gain a better understanding of machine learning classification models and that I wanted to do a project related to mushrooms. My goals were to implement several machine learning models that we learned in class successfully in order to gain a better understanding and learn how to implement them.

One of the biggest challenges was having a weak understanding of machine learning models. Thus, in order to achieve this task, I focused on implementing one model at a time and then experimented with ways to improve the model by using documentation and asking TAs. Then I would compare the model to my other results and draw conclusions

Early on I realized that a regression model is not applicable to a classification problem, so I replaced it with a k-nearest neighbor model. Perceptron performed the worst even after feature selection. The next most accurate model was a basic decision tree classifier. The best performing models were the random forest, K-nearest-neighbors, and a voting classifier of 10 bootstrapped decision trees. These models all performed almost identically with the voting classifier technically performing the best. However, the difference between these models was under .2 %.

2. Data Mining Task

The input data was a mushroom classification dataset from kraggle.com. This dataset was in the form of a csv file and consisted of 22 different categorical variables across 8000 different mushrooms. The dataset also classified mushrooms as either poisonous or edible. The output of the data mining approach was a perceptron classifier, decision tree classifier, k-nearest-neighbors classifier, a decision tree based voting classifier, and a random forest classifier. All these models were applied on the raw dataset and the dataset after feature selection.

The questions that I set out to investigate in this project were: what is the highest accuracy a classifier can achieve, which classifier will perform the best, and will feature selection improve the accuracy of models?

The key challenges to solving this task were creating decision tree classifiers that do not overfit, correctly encoding categorical data, deciding what form of feature selection to use, becoming familiar with the psychic learn API, and trying to represent results visually.

3. Technical Approach

The first step taken was to explore and clean the data. To do this both the labels associated with each feature and the number of unique labels were printed out. During this process, the label '?' was discovered and was replaced with numpy.nan values. Next, one-hot encoding was performed to prepare the data for use in machine learning models. Additionally, duplicate values were removed.

After performing data cleaning, feature selection was performed on the mushroom data. To prune features a perceptron-based feature selection algorithm was used (per the recommendation of TA Emma Mickas). To do feature selection, the perceptron algorithm was run on the full data set to identify which features had the lowest weights. The four features with the lowest weights were removed.

Next, both the unpruned and pruned data was split into training and testing sections and classifiers were applied. The general approach was to train and test models on the full data set, train and test models on the feature pruned data set, and then compare results. The models applied were perceptron, a decision tree, k-nearest neighbors, a voting classifier of 10 bootstrapped decision trees, and a random forest classifier.

Perceptron was simply the standard model from scikit-learn. The decision tree classifier had a specified max depth of 5 and criterion set to entropy. The next classifier was k-nearest neighbors, the algorithm limited to identifying the five nearest neighbors. The next classifier used was a voting classifier consisting of 10 bootstrapped decision trees. Each of the individual decision trees had a max depth of 5, a criterion of entropy, and the splitting method was set to random. The last model was a Random Forest classifier. The base classifier from scikit-learn was used with a max depth of five specified.

To address the challenges listed above I did the following:

To create decision tree classifiers that do not overfit I specified max depth and used a voting classifier. The purpose of specifying max depth is to prevent excessive overfitting to the data. Additionally, I used a voting classifier with 10 bootstrap decision trees. This creates a classifier that is arriving at its prediction based on 10 different decision trees on different sections of the data. This makes the voting classifier more robust than one decision tree and because the decision trees are based on different sections of the data it reduces overfitting. Thus, the voting classifier will be more generalizable.

To address the issue of categorical variables I researched several methods of converting categorical variables into numerical variables such as pandas .getDummies(), scikit-learn's LabelEncoder and scikit-learn's OneHotEncoder. I ended up using one hot encoding.

Another issue I faced was deciding which form of feature selection to use. To gain guidance, I went to Emma Mickas' office hours. They suggested that I should use a perceptron-based feature selection method. Additionally, office hours were especially helpful in helping me become familiar with the scikit-learn API. Once I was able to learn the general structure of machine learning models in scikit-learn I was able to implement the specific models I wanted. However, one challenge that I was unable to fully resolve was representing my data visually. While I was able to succeed in visualizing decision trees, I struggled to use t-SNE plots as I could not understand the API.

```
# psudo code

# read in csv
data = getDataFromCSV()

# print out features and their labels
for feature in data:
    print feature, "number of labels: ", len(feature)

# replace the '?' label
data.replace('?', numpy.nan)

# train a perceptron for feature pruning
model = perceptron(data)

# remove four lowest features
pruned_data = data.dropFeatures(model.weights(lowest=4))

# try each model on data and pruned data
p = perceptron(data).accuracy
p2 = perceptron(pruned_data).accuracy
print(p, p2)

dt = decisionTreeClassifier(data).accuracy
dt2 = decisionTreeClassifier(pruned_data).accuracy
print(d, dt2)
print(dt.graph, dt2.graph)

knn = knnClassifier(data).accuracy
knn2 = knnClassifier(pruned_data).accuracy
print(knn, knn2)

vdt = votingDecisionTreeClassifier(data)
vdt2 = votingDecisionTreeClassifier(pruned_data).accuracy
for 10:
    m = decisionTreeClassifier(data, bootstrap=True)
    vdt.add(m)

    m2 = decisionTreeClassifier(pruned_data, bootstrap=True)
    vdt2.add(m2)
print(vdt, vdt2)

rf = randomForestClassifier(data).accuracy
rf2 = randomForestClassifier(pruned_data).accuracy
print(rf, rf2)
```

4. Evaluation Methodology

The data set I used has a collection of hypothetical samples of 23 species of grilled mushrooms drawn from the Auburn Society Field Guide to North American Mushrooms. The source of this data set was kraggle.com. One of the specific struggles with this data set was that it is relatively small. Only containing about 8000 mushrooms.

This means that decision trees were likely to overfit to my data and still execute relatively fast. Overfitted decision tree classifiers can be extremely fast on small data but when applied to larger data sets can become extremely slow. This is because a tree of extremely large depth does not significantly impact performance with small data sets, but a large tree will be very slow for a larger data set.

I realized that the decision trees were overfitting when the classifiers used had an accuracy of 100% on both the training and testing data. To address this problem a max depth of five was specified for each decision tree. After selecting this max depth, the decision tree classifier averaged an accuracy of 95% on testing data. Additionally, to address this problem, I used a voted classifier of bootstrap decision trees with a fixed depth of 5 to try and make a model that would be more generalizable to larger data.

Another issue I encountered was struggling with encoding categorical variables to numerical variables so that all the models could be applied. Additionally, the format of the data was a CSV file and I struggled to format this into an input that scikit-learn could read. However, after going to office hours, I successfully encoded the categorical variables and transferred the CSV file into a format that scikit-learn could read.

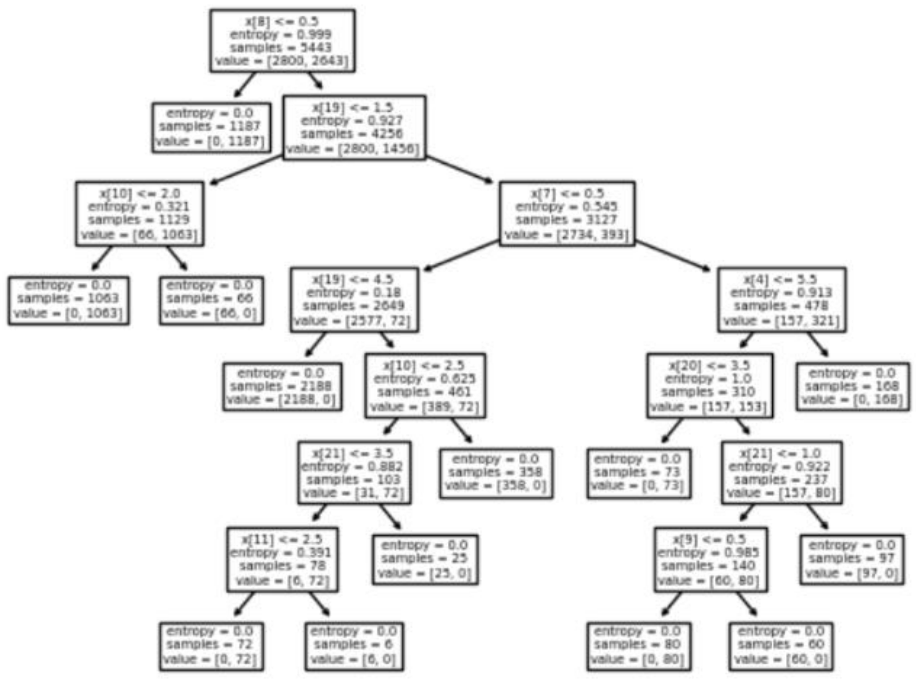
To evaluate the models, I used scikit-learn's metrics accuracy score. This allows me to test the accuracy of a model using testing data. I used this metric as it is commonly used in machine learning, and it was recommended to me in office hours. Also, another way I measured the success of my models specifically for decision trees was looking at their depth. When I initially ran a decision tree classifier, I did not specify Max depth. To evaluate this, I would print out a graphical representation of the decision tree using scikit-learn's plot_tree() functionality.

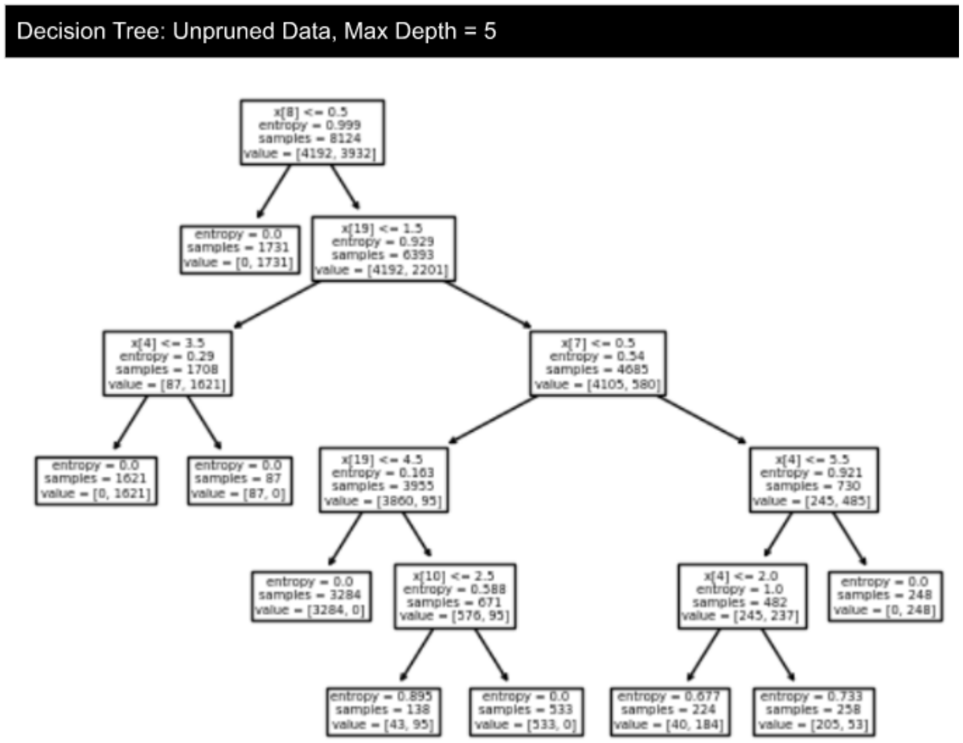
5. Results and Discussion

Initially I started with only one decision tree model. This decision tree achieved an accuracy of 100% on both the training and the testing data. This was due to me forgetting to specify a max depth and thus the decision tree would fit my data perfectly. To address this, I specified a max height of five based and I implemented the voting classifier on the recommendation of TA Emma Mickas. When comparing decision tree models, I used a visualization of the tree to see their depth. When I did not specify the max depth the decision tree had a length of seven.

Additionally, I realized that because the amount of data I had was small even a decision tree with a specified max depth could generalize poorly because it was working off such a small amount of data. To improve the robustness of the model I decided to use a voting classifier of 10 different decision trees using bootstrapping. Each of these individual 10 decision trees were given a random selection of the data to be their train data and their test data. This was so I could evaluate each individual decision tree as well as evaluate how accurate the voting class fire was overall. Next, I decided to use a random Forest classifier. This was because I wanted to compare the accuracy of my voting classifier with a random forest classifier which is one of the most effective classifiers for tabular data. After creating these models, I also wanted to try other non-tree-based models on the data to see how they would perform.

Decision Tree: Unpruned Data




Decision Tree: Unpruned Data, Max Depth = 5
x[8] <= 0.5 entropy = 0.999 samples = 8124 value = [4192, 3932]
entropy = 0.0 samples = 1731 value = [0, 1731]
x[19] <= 1.5 entropy = 0.929 samples = 6393 value = [4192, 2201]
x[4] <= 3.5 entropy = 0.29 samples = 1708 value = [87, 1621]
x[7] <= 0.5 entropy = 0.54 samples = 4685 value = [4105, 580]
entropy = 0.0 samples = 1621 value = [0, 1621]
entropy = 0.0 samples = 87 value = [87, 0]
x[19] <= 4.5 entropy = 0.163 samples = 3955 value = [3860, 95]
x[4] <= 5.5 entropy = 0.921 samples = 730 value = [245, 485]
entropy = 0.0 samples = 3284 value = [3284, 0]
x[10] <= 2.5 entropy = 0.588 samples = 671 value = [576, 95]
x[4] <= 2.0 entropy = 1.0 samples = 482 value = [245, 237]
entropy = 0.0 samples = 248 value = [0, 248]
entropy = 0.895 samples = 138 value = [43, 95]
entropy = 0.0 samples = 533 value = [533, 0]
entropy = 0.677 samples = 224 value = [40, 184]
entropy = 0.733 samples = 258 value = [205, 53]

I decided to use a perceptron and a k-nearest neighbors model. The reason I use the k-nearest neighbor's model and not the regression model was because regression models are not suited for classifier problems, which I did not know at the time of writing my proposal. K-nearest neighbors was a desirable choice because k-nearest neighbors can work with higher dimensional data and is a good model for classification. After seeing that the perceptron was not performing as well as the other models, I wanted to attempt data pruning. At the recommendation of TA Emma Mickas, I decided to use a perceptron-based pruning method. I ran a perceptron algorithm on the full data set, identified which weights had the least impact on the data, and then the features that those weights correspond to I removed. I removed the four least impactful features from the data. The reason I used four features is because through experimentation I found that removing the lowest four features was the most effective. Interestingly, the only model that improved on the prune data set was the perceptron; it had no effect on the accuracy of the other models.

The accuracies of the models are as follows:

Accuracy of Model on Data	Perceptron	Decision Tree	KNN	Voting (varies within .01 due to random splitting)	Random Forest

Original Data	0.7967176426706453	0.9757553151809026	0.9985080193957478	1.0	0.9926144756277696
Pruned Data	0.9324878776575904	0.9757553151809026	0.9992540096978739	0.9992614475627769	0.9937223042836041

All of the models performed reasonably well. This was likely because my data was hypothetically created to work for data mining projects as a learning exercise. Additionally, the fact that there was not a lot of data in the data set made it so models were able to perform quickly with a high degree of accuracy. The lowest performing model was perceptron on the unpruned data as discussed above. The reason that perceptron had a higher accuracy with the pruned data is that unimportant features will no longer be affecting the updating of the weights. Thus, leading to more accurate predictions.

However, the reason that this pruning did not work for the decision trees is because feature selection generally does not have a significant impact on decision trees. This is because decision trees select the optimal feature for the split, thus unimportant features will not be used in deciding of nodes. Additionally, when k-nearest neighbors is evaluating which points are a point's closest neighbors it will rely on the features that are more impactful. Therefore, it will ignore less important features and this is likely why there was no significant impact when using the pruned data set.

6. Lessons Learned

I learned a lot about scikit-learn and about classifier machine learning models. I learned how to use scikit-learn and my confidence with the API has increased significantly. Also, I have learned a lot more about how decision trees work as most of my models are decision tree based. Additionally, I learned how feature pruning using perceptron is not that effective when you are using other models that are not perceptron. If I were to do something differently, I would use a different feature selection method. In addition, one aspect of the project that I really struggled with was displaying my data and results visually. I attempted several times to use t-SNE plots however the API was extremely difficult for me especially when I was also trying to learn the scikit-learn API. Now that I am more familiar with scikit-learn, if I were to do a similar project again, I would spend more time learning how to do t-SNE plots.

7. Acknowledgements

Special thanks to Emma Mickas for their help with understanding the scikit-learn API and helping me understand decision trees.

Special thanks to Professor Doppa for helping with feature selection and helping me understand all the models used.

Special thanks to the scikit-learn team for their API and documentation.

The following AUR maintainers for their python libraries:

https://aur.archlinux.org/packages/python-mlxtend

https://aur.archlinux.org/packages/anaconda

The following websites:

https://www.kaggle.com/datasets/uciml/mushroom-classification

https://realpython.com/knn-python/

https://towardsdatascience.com/an-exhaustive-guide-to-classification-using-decision-trees-8d472e77223f

https://realpython.com/logistic-regression-python/

https://www.linkedin.com/pulse/how-use-virtual-environment-inside-jupyter-lab-sina-khoshgoftar