Seth Beckett & Jasper Swenson
CS 6830
Project 8
Link: Github and Slides

Introduction

In this project we will be using two predictive model types, decision tree classifier and neural networks, to classify the types of glass given in our selected dataset. Our objective is to provide an overview of the accuracy of the models, and an analysis of how they change given different parameters, for example the maximum depth of a decision tree.

Dataset

For this project we chose to use this glass type dataset from Kaggle. Each entry in the dataset gives information about a piece of glass that falls into one of the 7 glass types, those types being:

1. Building_windows_float_processed
2. Building_windows_non_float_processed
3. Vehicle_windows_float_processed
4. Vehicle_windows_non_float_processed (none in this dataset)
5. Containers
6. Tableware
7. Headlamps

For each entry in the dataset the following characteristics are available:

- RI - refractive index
- Na: Sodium - unit measurement: weight percent in corresponding oxide
- Mg: Magnesium - unit measurement: weight percent in corresponding oxide
- Al: Aluminum - unit measurement: weight percent in corresponding oxide
- Si: Silicon - unit measurement: weight percent in corresponding oxide
- K: Potassium - unit measurement: weight percent in corresponding oxide
- Ca: Calcium - unit measurement: weight percent in corresponding oxide
- Ba: Barium - unit measurement: weight percent in corresponding oxide

- Fe: Iron - unit measurement: weight percent in corresponding oxide

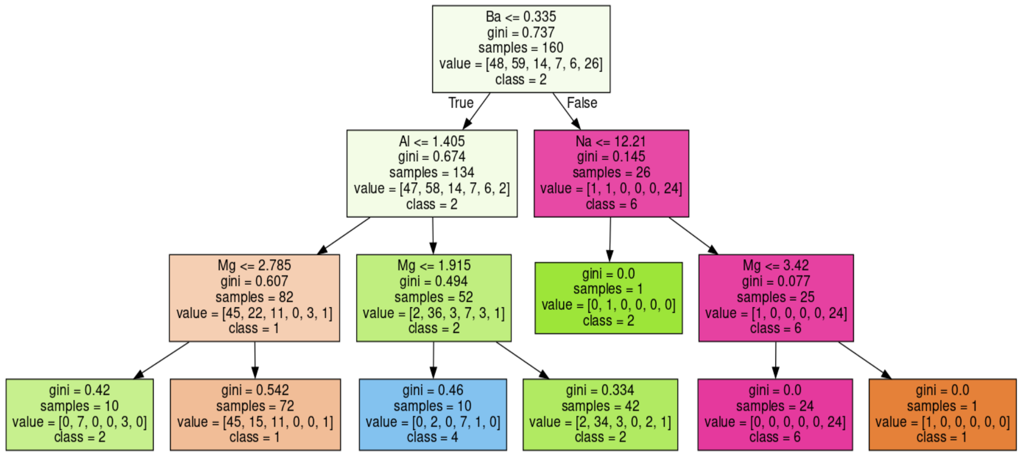
We will use the characteristics given for each entry in the dataset as our feature set for the decision tree and neural networks, luckily for us this dataset is given in a clean state so we don't need to do any formatting etc.

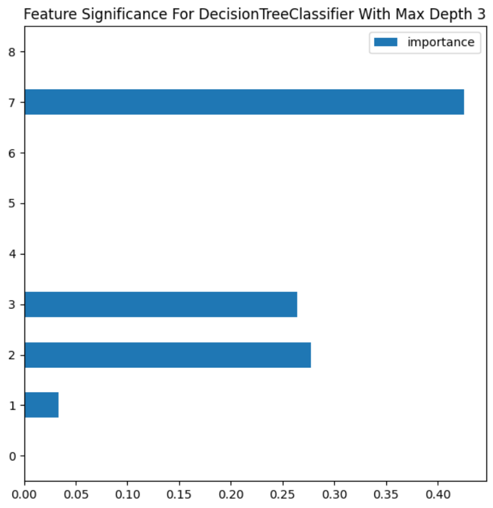
Analysis techniques - Decision Tree Classifiers

The first technique we applied in analyzing this dataset was the decision tree classifier, but before we could train the model we first had to set up our training and testing sets for both the feature set and glass types respectively. With that done we were able to create two decision tree classifier models using the sklearn.tree package. We varied the maximum depth for the two trees, giving the first one a maximum depth of 3 and the second one a maximum depth of 5. As with other projects we evaluated the accuracy of these models through the precision, recall, and f-score metrics provided by sklearn.metrics. We also visualized each tree using techniques we learned in class, and determined the most impactful features from our feature set.

Results - Decision Tree Classifiers

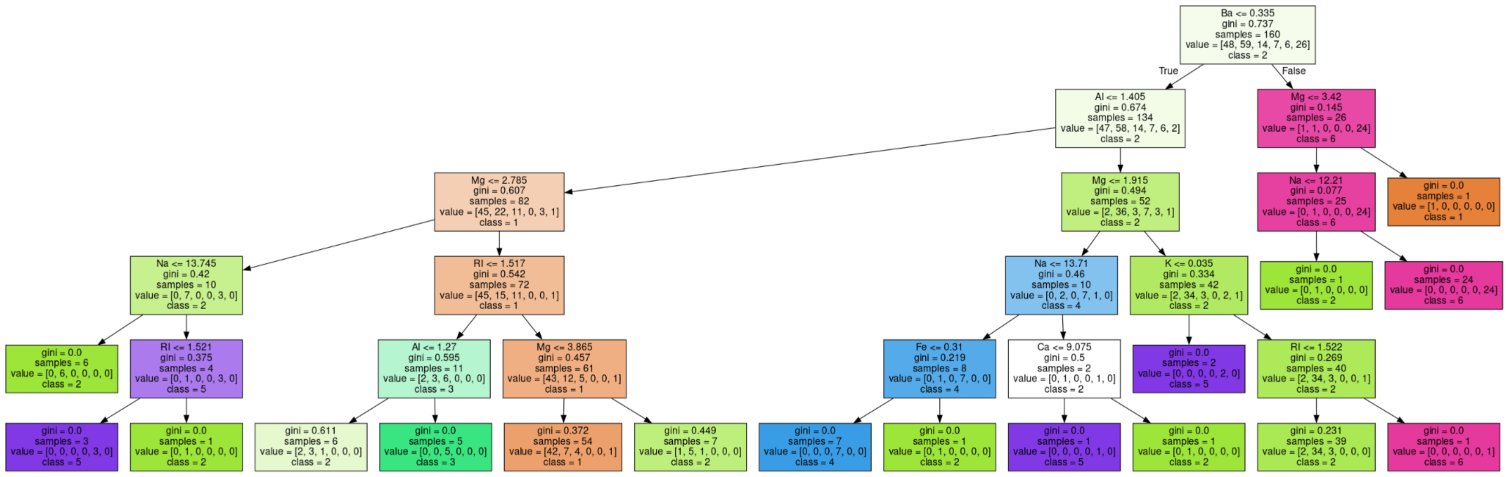
Using maximum depth 3:


Ba <= 0.335 gini = 0.737 samples = 160 value = [48, 59, 14, 7, 6, 26] class = 2
True
False
Al <= 1.405 gini = 0.674 samples = 134 value = [47, 58, 14, 7, 6, 2] class = 2
Na <= 12.21 gini = 0.145 samples = 26 value = [1, 1, 0, 0, 0, 24] class = 6
Mg <= 2.785 gini = 0.607 samples = 82 value = [45, 22, 11, 0, 3, 1] class = 1
Mg <= 1.915 gini = 0.494 samples = 52 value = [2, 36, 3, 7, 3, 1] class = 2
gini = 0.0 samples = 1 value = [0, 1, 0, 0, 0, 0] class = 2
Mg <= 3.42 gini = 0.077 samples = 25 value = [1, 0, 0, 0, 0, 24] class = 6
gini = 0.42 samples = 10 value = [0, 7, 0, 0, 3, 0] class = 2
gini = 0.542 samples = 72 value = [45, 15, 11, 0, 0, 1] class = 1
gini = 0.46 samples = 10 value = [0, 2, 0, 7, 1, 0] class = 4
gini = 0.334 samples = 42 value = [2, 34, 3, 0, 2, 1] class = 2
gini = 0.0 samples = 24 value = [0, 0, 0, 0, 0, 24] class = 6
gini = 0.0 samples = 1 value = [1, 0, 0, 0, 0, 0] class = 1

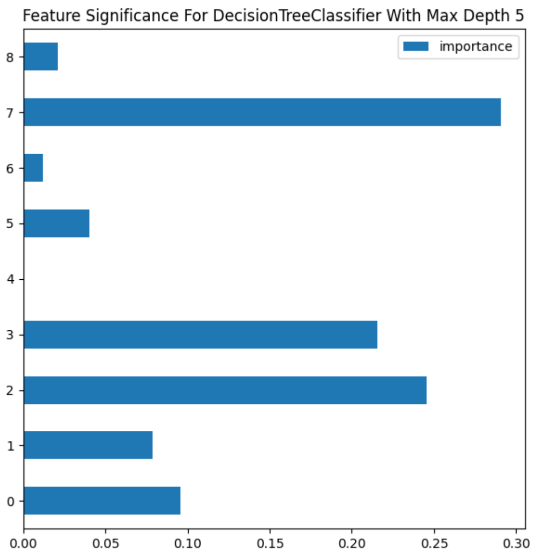


We found that the most informative feature in our feature set was Ba (Barium) with Mg (Magnesium) and Al (Aluminium) following in significance. This model had a precision score of 0.93, an f-score of 0.98, and a recall of 1.05. While it performed very well, we think performance may have been affected by the small size of our dataset.

Using maximum depth 5:



Ba <= 0.335 gini = 0.737 samples = 160 value = [48, 59, 14, 7, 6, 26] class = 2
True
False
Al <= 1.405 gini = 0.674 samples = 134 value = [47, 58, 14, 7, 6, 2] class = 2
Mg <= 3.42 gini = 0.145 samples = 26 value = [1, 1, 0, 0, 0, 24] class = 6
Mg <= 2.785 gini = 0.607 samples = 82 value = [45, 22, 11, 0, 3, 1] class = 1
Mg <= 1.915 gini = 0.494 samples = 52 value = [2, 36, 3, 7, 3, 1] class = 2
Na <= 12.21 gini = 0.077 samples = 25 value = [0, 1, 0, 0, 0, 24] class = 6
gini = 0.0 samples = 1 value = [1, 0, 0, 0, 0, 0] class = 1
Na <= 13.745 gini = 0.42 samples = 10 value = [0, 7, 0, 0, 3, 0] class = 2
RI <= 1.517 gini = 0.542 samples = 72 value = [45, 15, 11, 0, 0, 1] class = 1
Na <= 13.71 gini = 0.46 samples = 10 value = [0, 2, 0, 7, 1, 0] class = 4
K <= 0.035 gini = 0.334 samples = 42 value = [2, 34, 3, 0, 2, 1] class = 2
gini = 0.0 samples = 1 value = [0, 1, 0, 0, 0, 0] class = 2
gini = 0.0 samples = 24 value = [0, 0, 0, 0, 0, 24] class = 6
gini = 0.0 samples = 6 value = [0, 6, 0, 0, 0, 0] class = 2
RI <= 1.521 gini = 0.375 samples = 4 value = [0, 1, 0, 0, 3, 0] class = 5
Al <= 1.27 gini = 0.595 samples = 11 value = [2, 3, 6, 0, 0, 0] class = 3
Mg <= 3.865 gini = 0.457 samples = 61 value = [43, 12, 5, 0, 0, 1] class = 1
Fe <= 0.31 gini = 0.219 samples = 8 value = [0, 1, 0, 7, 0, 0] class = 4
Ca <= 9.075 gini = 0.5 samples = 2 value = [0, 1, 0, 0, 1, 0] class = 2
gini = 0.0 samples = 2 value = [0, 0, 0, 0, 2, 0] class = 5
RI <= 1.522 gini = 0.269 samples = 40 value = [2, 34, 3, 0, 0, 1] class = 2
gini = 0.0 samples = 3 value = [0, 0, 0, 0, 3, 0] class = 5
gini = 0.0 samples = 1 value = [0, 1, 0, 0, 0, 0] class = 2
gini = 0.611 samples = 6 value = [2, 3, 1, 0, 0, 0] class = 2
gini = 0.0 samples = 5 value = [0, 0, 5, 0, 0, 0] class = 3
gini = 0.372 samples = 54 value = [42, 7, 4, 0, 0, 1] class = 1
gini = 0.449 samples = 7 value = [1, 5, 1, 0, 0, 0] class = 2
gini = 0.0 samples = 7 value = [0, 0, 0, 7, 0, 0] class = 4
gini = 0.0 samples = 1 value = [0, 1, 0, 0, 0, 0] class = 2
gini = 0.0 samples = 1 value = [0, 0, 0, 0, 1, 0] class = 5
gini = 0.0 samples = 1 value = [0, 1, 0, 0, 0, 0] class = 2
gini = 0.231 samples = 39 value = [2, 34, 3, 0, 0, 0] class = 2
gini = 0.0 samples = 1 value = [0, 0, 0, 0, 0, 1] class = 6

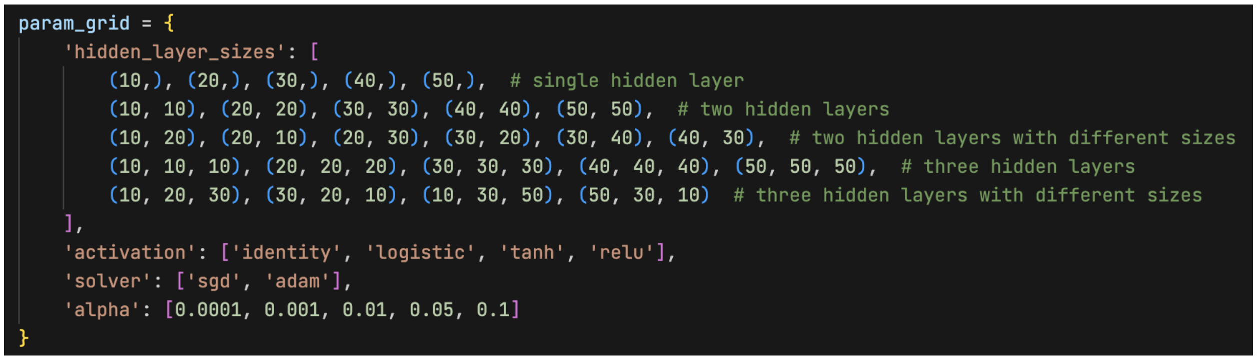


Interestingly, when using a maximum depth of 5 for this model we got slightly different results for the significance of our features. Ba (Barium) still came in first as the most significant features with Mg (Magnesium) and Al (Aluminium) coming in second and third, but the rest of our features now seem to have far more significance than they did with a depth of 3. This model had a precision score of 0.95, an f-score of 0.98, and a recall of 1.03, while high we believe these accuracies may again be a result of our relatively small dataset. In future projects we may decide to go with something larger.

When we compare the visualizations of each of these models it becomes clear that the maximum depth has a clear effect on the choices the algorithm makes at each level of the tree. For example, at level 2 the model with a maximum depth of 3 chooses to split on Al and Na, while the model with a maximum depth of 5 chooses to split on Al and Mg. These differences continue as the depth increases, which also has some effect on the gini score.

Analysis techniques - Neural Networks

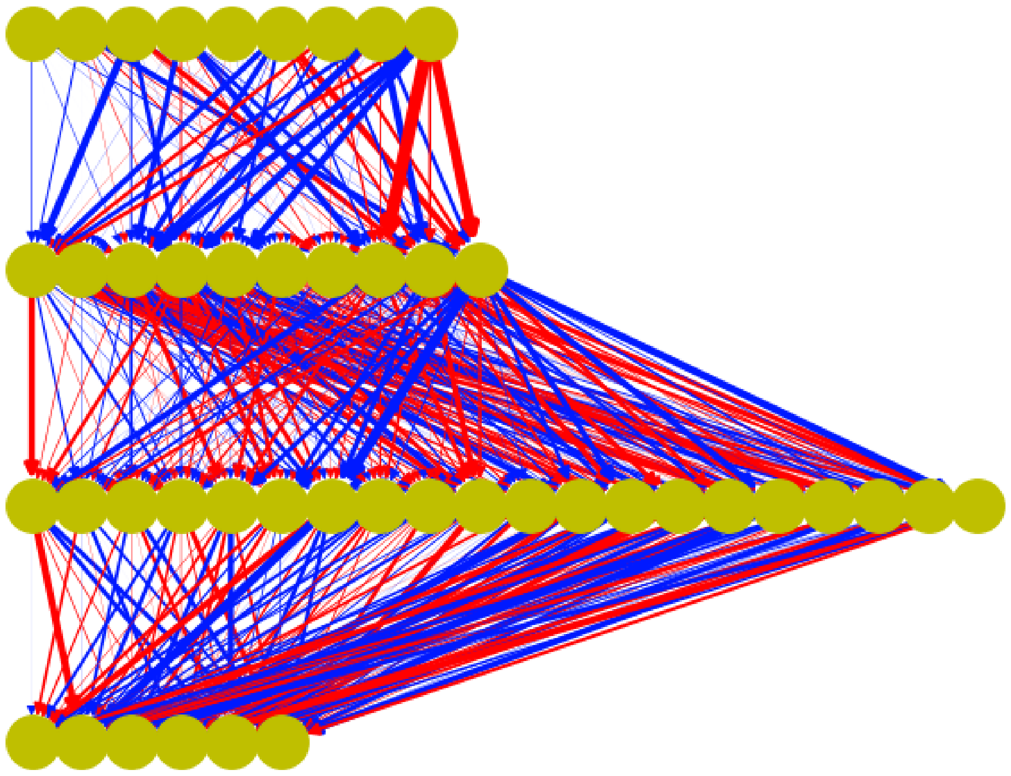
For the neural network to be fitted on the glass dataset, we used a multilayer perceptron classifier. There are a lot of different parameters and hyperparameters that can be tested, so to find the optimal parameters we used a cross validated gridsearch, which fit many models with each possible combination of specified parameters. The model with the best results ended up being the model fit using the tanh activation function, an alpha of 0.05, hidden layers of size 10 and 20, and used the adam solver function. Similar to the decision tree classifier, we then evaluated its performance using precision, recall, f1, and accuracy as metrics. The following picture shows the parameters we used to find the best neural network.



```
param_grid = {
    'hidden_layer_sizes': [
        (10,), (20,), (30,), (40,), (50,),  # single hidden layer
        (10, 10), (20, 20), (30, 30), (40, 40), (50, 50),  # two hidden layers
        (10, 20), (20, 10), (20, 30), (30, 20), (30, 40), (40, 30),  # two hidden layers with different sizes
        (10, 10, 10), (20, 20, 20), (30, 30, 30), (40, 40, 40), (50, 50, 50),  # three hidden layers
        (10, 20, 30), (30, 20, 10), (10, 30, 50), (50, 30, 10)  # three hidden layers with different sizes
    ],
    'activation': ['identity', 'logistic', 'tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1]
}
```

Results - Neural Networks

Not surprisingly, searching over our wide variety of parameters took a long time. When it finally did converge on the best model, we then created a new MLP classifier with those parameters, and fit it on our predefined testing data. Visualized, our neural network with the parameters of 10 and 20 hidden layer sizes, alpha = 0.05, tanh activation, and adam solver function looks pretty confusing. This makes sense, as neural networks (especially with lots of large layers) look very confusing due to the insane amounts of edge weights interconnecting.



We originally used a train-test split to test our neural network performance, but found that our testing set contained very few or none of some classes, so it made it difficult to accurately assess the metrics. To adjust for this, we used cross-validated metrics, and found that our model had a macro average precision of 0.64, recall of 0.67, f1 of 0.65, and an accuracy of 0.65. Our edge weights also were fairly uninterpretable and unrelatable to our decision tree, as the number of neurons and layers were varying. Given that there are 6 different target classes, our metrics aren’t too bad, but we believe that if we had the time and resources to test even more parameters of our neural network we could find an even better model.