SENG 474: Assignment 1 Report

Nathan Woloshyn

Part 1: Processing the data

The first step is to load the data into a pandas dataframe. The data is stored in a csv file, so we can use the read_csv function to load it into a dataframe. After this, we split the data into a training set and a test set. What fraction of the data is used for training and what fraction is used for testing is a hyperparameter that can be tuned. We choose a 75/25 split as our default, but will also analyze the results of other splits. We also specify a random seed for the sampling, so that we can reproduce the results of our experiments.

```
import pandas as pd

'''Reads the data from the csv file and returns a pandas dataframe.'''
def read_data():
    df = pd.read_csv('./cleaned_adult.csv')

    return df



'''Partitions the data into train and test sets, using a taking what
percentage of the data train / test on as input. Returns the train and test'''
def partition_data(df, train_size=0.75, random_state=99):
    # Split the data into train and test sets. (0.75, 0.25) split.
    train_df = df.sample(train_size, random_state)
    test_df = df.drop(train_df.index)

    return train_df, test_df
```

Part 2: Decision Trees

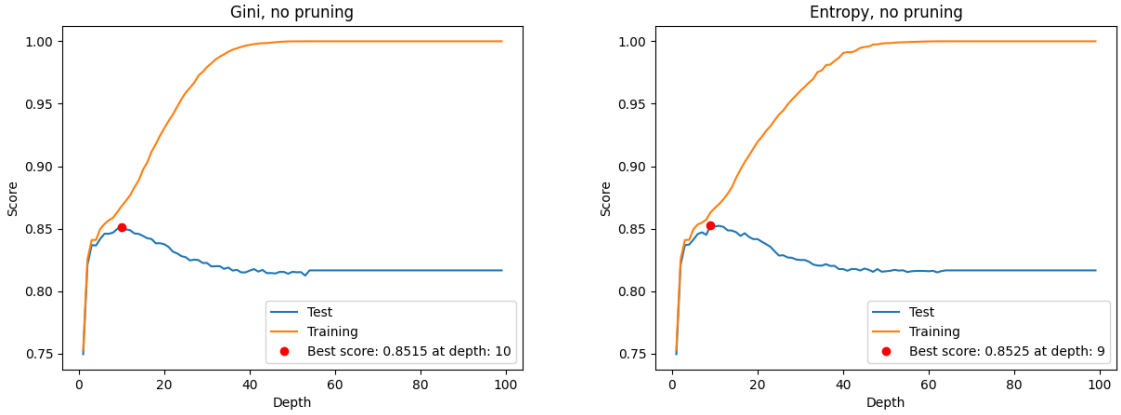
Part 2.1: No Pruning

In our first experiment we use the sklearn implementation of a decision tree classifier. We use the default parameters, which means that the tree is not pruned. We use test both entropy and Gini impurity as our criterion for splitting the tree. Using the code below we test every depth of tree from 1 to 100 using these two criteria. We use our default choice of 75/25 train/test split, giving all trees the same train/test split.

```
train, test = read_data.partition_data(read_data.read_data())


'''Test various depths, with entropy as the criterion,
store and plot the scores on both training and test sets using matplotlib'''
def test_entropy_depths():
    best_depth = 0
    best_score = 0
    test_scores = []
    training_scores = []
    for i in range(1, 100):
        eD = DecisionTreeClassifier(random_state=0, max_depth=i,
         criterion="entropy").fit(train.drop(columns=['income']),
         train['income'])


        test_scores.append(eD.score(test.drop(columns=['income']), test['income']))
        training_scores.append(eD.score(train.drop(columns=['income']), train['income']))
        if eD.score(test.drop(columns=['income']), test['income']) > best_score:
            best_score = eD.score(test.drop(columns=['income']), test['income'])
            best_depth = i
    plt.plot(range(1, 100), test_scores, label='Test')
    plt.plot(range(1, 100), training_scores, label='Training')
    plt.plot(best_depth, best_score, 'ro', label='Best score: '
        + "{:.4f}".format(best_score) + ' at depth: ' + str(best_depth))
    plt.legend(loc="lower right")
    plt.xlabel('Depth')
    plt.ylabel('Score')
    plt.title('Entropy, no pruning')
    #plt.show()
    plt.savefig('entropy_no_pruning_scores_varying_depth.png')
    plt.clf()
```

```
'''Test various depths, with gini as the criterion,
store and plot the scores on both the training and test sets using matplotlib'''
def test_gini_depths():
    best_depth = 0
    best_score = 0
    test_scores = []
    training_scores = []
    for i in range(1, 100):
        gD = DecisionTreeClassifier(random_state=0, max_depth=i,
        criterion="gini").fit(train.drop(columns=['income']), train['income'])

        test_scores.append(gD.score(test.drop(columns=['income']), test['income']))
        training_scores.append(gD.score(train.drop(columns=['income']), train['income']))
        if gD.score(test.drop(columns=['income']), test['income']) > best_score:
            best_score = gD.score(test.drop(columns=['income']), test['income'])
            best_depth = i
    plt.plot(range(1, 100), test_scores, label='Test')
    plt.plot(range(1, 100), training_scores, label='Training')
    plt.plot(best_depth, best_score, 'ro', label='Best score: '
         + "{:.4f}".format(best_score) + ' at depth: ' + str(best_depth))
    plt.legend(loc="lower right")
    plt.xlabel('Depth')
    plt.ylabel('Score')
    plt.title('Gini, no pruning')
    #plt.show()
    plt.savefig('gini_no_pruning_scores_varying_depth.png')
    plt.clf()

test_entropy_depths()
test_gini_depths()
```

Running the code above gives us the following results:



We can see that at low depth accuracy quickly grows as we allow a tree to make more splits, but around depth = 10 the models performance quickly declines. This is likely due to overfitting, as we can see that as depth increases the training accuracy continues to increase, but the test accuracy starts to decrease. This is a sign that the model is overfitting to the training data, and is not generalizing well to the test data. This is a common problem with decision trees, and why we will be using pruning in our next experiment. Overall, the two choices of criterion, entropy and Gini impurity, seem to perform similarly, with the best score being around 0.85 for both, and both having a best depth of around 10.

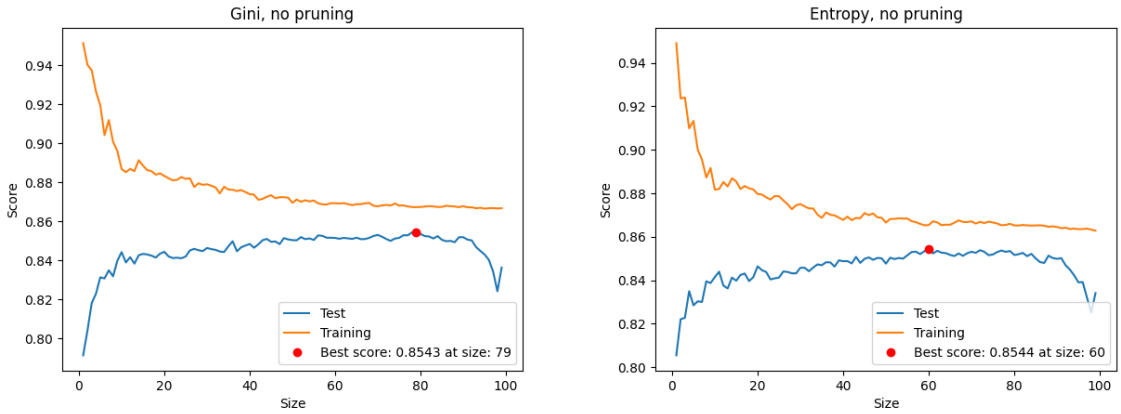
Next we test the effect of varying what percentage of the data we allocate to training and test, using the same depth of 10 for the tree. We use the code below to test the effect of varying the train/test split from 0% to 100% in 1% increments.

```
'''Test various training set sizes, with entropy as the criterion, using depth = 10'''

def test_entropy_sizes():
    best_size = 0
    best_score = 0
    test_scores = []
    training_scores = []
    for i in range(1, 100):
        train, test = read_data.partition_data(read_data.read_data(), train_size=i/100)
        eS = DecisionTreeClassifier(random_state=0, max_depth=10,
            criterion="entropy").fit(train.drop(columns=['income']), train['income'])
        test_scores.append(eS.score(test.drop(columns=['income']), test['income']))
        training_scores.append(eS.score(train.drop(columns=['income']), train['income']))
        if eS.score(test.drop(columns=['income']), test['income']) > best_score:
            best_score = eS.score(test.drop(columns=['income']), test['income'])
            best_size = i
```

```
    plt.plot(range(1, 100), test_scores, label='Test')
    plt.plot(range(1, 100), training_scores, label='Training')
    plt.plot(best_size, best_score, 'ro', label='Best score: '
         + "{:.4f}".format(best_score) + ' at size: ' + str(best_size))
    plt.legend(loc="lower right")
    plt.xlabel('Size')
    plt.ylabel('Score')
    plt.title('Entropy, no pruning')
    #plt.show()
    plt.savefig('entropy_no_pruning_scores_varying_size.png')
    plt.clf()

'''Test various training set sizes, with gini as the criterion, using depth = 10'''

def test_gini_sizes():
    best_size = 0
    best_score = 0
    test_scores = []
    training_scores = []
    for i in range(1, 100):
        train, test = read_data.partition_data(read_data.read_data(), train_size=i/100)
        gS = DecisionTreeClassifier(random_state=0, max_depth=10,
            criterion="gini").fit(train.drop(columns=['income']), train['income'])
        test_scores.append(gS.score(test.drop(columns=['income']), test['income']))
        training_scores.append(gS.score(train.drop(columns=['income']), train['income']))
        if gS.score(test.drop(columns=['income']), test['income']) > best_score:
            best_score = gS.score(test.drop(columns=['income']), test['income'])
            best_size = i
    plt.plot(range(1, 100), test_scores, label='Test')
    plt.plot(range(1, 100), training_scores, label='Training')
    plt.plot(best_size, best_score, 'ro', label='Best score: '
        + "{:.4f}".format(best_score) + ' at size: ' + str(best_size))
    plt.legend(loc="lower right")
    plt.xlabel('Size')
    plt.ylabel('Score')
    plt.title('Gini, no pruning')
    #plt.show()
    plt.savefig('gini_no_pruning_scores_varying_size.png')
    plt.clf()

test_entropy_sizes()
```

```
test_gini_sizes()
```

Running the code above gives us the following results:



We can see that the accuracy of of the model is sensitive to how we partition our data. As we increase the size of the training set, the accuracy of the model increases, but at some point the accuracy the returns become diminishing, and eventually the test set is so small that the model is not able to generalize well. However, unlike the last experiment, there seems to be meaningfully different behavior from our two criteria. We were surprised by this, given the first experiment. So we ran several trials of this experiment, varying the random seed used to partition the data, and the results were consistent. The Gini criterion sees it's best performance when around 80% of the data is allocated to training, while the entropy criterion consistently reaches peak performance when around 60% of the data is allocated to training.

Part 2.2: Pruning