**Assignment 1**

By Harsh Sharma

**Part IV. Decision Tree Learning**

1. Build a decision tree model from *party.data,* using the programs provided. Report **(1)** the **true class label** and **predicted class label** of each (training) instance, and **(2)** the **training error rate.**

Solution:

True Class Label: ['Party', 'Study', 'Party', 'Party', 'Pub', 'Party', 'Study', 'TV', 'Party', 'Study']

Predicted Class Label : ['Party', 'Study', 'Party', 'Party', 'Pub', 'Party', 'Study', 'TV', 'Party', 'Study']

The predicted class is exactly the same as the actual one and therefore the training error rate is **zero (0)**.

Also, I’ve modified parts of the code so that it runs correctly. I’ll be attaching working code as well. They are named**: party.py** and **dtree.py**

Q 2. The program uses a given dataset for both training and validation purpose. (1) Revise the program code to prepare a training dataset and a validation dataset from a user input dataset using ***bootstrap****.*

(2) Revise the program code for computing the **test error rate**.

Submit the revised program code with your explanation of the changed/added program parts.

Solution:

Here is the bootstrap function that I created to divide the dataset for training and testing:

def bootstrap\_split(data, percentage=0.8):

    """Split the data into training and validation sets using bootstrap."""

    num\_samples = len(data)

    train\_size = int(percentage \* num\_samples)

    # Generate a bootstrap sample

    indices = np.random.choice(range(num\_samples), size=train\_size, replace=True)

    train\_data = [data[i] for i in indices]

    # Validation set is the remaining data not included in the training set

    validation\_data = [data[i] for i in range(num\_samples) if i not in indices]

    return train\_data, validation\_data

To calculate test error:

error\_count = sum(1 for true\_class, predicted\_class in zip(true\_classes, predicted\_classes) if true\_class != predicted\_class)

test\_error\_rate = error\_count / len(true\_classes)

I also had to update other parts of the code to accommodate these changes. They have been incorporated in these files which I’ll also sharing: **party2.py** and **dtree2.py**

**3.** For impurity measures, three metrics are popular used: *entropy*, *gini index* and *misclassification error*

The *gini index* of a node *t* is computed as 1−Σ𝑝𝑖(𝑡)2𝑐𝑖=1 , where *c* is the number of distinct class labels and 𝑝𝑖(𝑡). *Information gain* can be also computed with the gini index.

Revise the program so that the program can chooses a test attribute based on **the *information gain* with *gini index*** instead of entropy-based information gain.

Submit the revised program code with your explanation of the changed/added program parts.

Solution:

**I modified make\_tree() function to use gini index for calculating Information gain.** This involves updating the calculation of information gain and selecting the best feature based on the Gini index.

**This is the code:**

gain = np.zeros(nFeatures)

            featureSet = list(range(nFeatures))

            for feature in featureSet:

                g = self.calc\_info\_gain(data, classes, feature)

                gain[feature] = g

            bestFeature = np.argmax(gain)

            tree = {featureNames[bestFeature]: {}}

Also, I needed to modify the calc\_info\_gain() function to compute information gain using the Gini index instead of entropy. The Gini index is calculated as 1−∑pi21−∑pi2​, where pipi​ is the proportion of samples belonging to class ii in the dataset.

This is the code for the same:

def calc\_info\_gain(self, data, classes, feature):

        # Calculates the information gain based on Gini index

        gini\_gain = 0

        nData = len(data)

        # List the values that feature can take

        values = np.unique([datapoint[feature] for datapoint in data])

        for value in values:

            value\_indices = [i for i, datapoint in enumerate(data) if datapoint[feature] == value]

            value\_classes = [classes[i] for i in value\_indices]

            # Calculate Gini index for the current value

            gini\_value = 1.0

            unique\_classes = np.unique(value\_classes)

            for cls in unique\_classes:

                p\_i = np.sum([1 for c in value\_classes if c == cls]) / len(value\_classes)

                gini\_value -= p\_i\*\*2

            gini\_gain += (len(value\_classes) / nData) \* gini\_value

        return gini\_gain

These changes have been updated to a new file and I’ll be sharing it as well. It’s called: **dtree3.py**

**4.** Rebuild a decision tree model from *party.data,* using the revised program.

Report **(1)** the **training error rate** and **(2)** the **test error rate**.

Solution:

To incorporate the changes I made in dtree3.py, I created a new file **party3.py** which then calls the previous **dtree3** class and gives the output. Also, in this file I’ve added the part to **calculate training error** which wasn’t implemented before.

One thing to note is that the test error rate varies with each run of code. It is so because the numbers of occurunces of data in the dataset varies and is random. And so the error also varies. In my various runs, it has varied from 0.25 to 0.888.

Whereas the training error rate remains constant at zero.