3. Bagging Features

We're now making trees based on different random subsets of our initial dataset. But we can continue to add variety to the ways our trees are created by changing the features that we use.

Recall that for our car data set, the original features were the following:

- The price of the car
- The cost of maintenance
- The number of doors
- The number of people the car can hold
- The size of the trunk
- The safety rating

Right now when we create a decision tree, we look at every one of those features and choose to split the data based on the feature that produces the most information gain. We could change how the tree is created by only allowing a subset of those features to be considered at each split.

For example, when finding which feature to split the data on the first time, we might randomly choose to only consider the price of the car, the number of doors, and the safety rating.

After splitting the data on the best feature from that subset, we'll likely want to split again. For this next split, we'll randomly select three features again to consider. This time those features might be the cost of maintenance, the number of doors, and the size of the trunk. We'll continue this process until the tree is complete.

One question to consider is how to choose the number of features to randomly select. Why did we choose `3` in this example? A good rule of thumb is to randomly select the square root of the total number of features. Our car dataset doesn't have a lot of features, so in this example, it's difficult to follow this rule. But if we had a dataset with `25` features, we'd want to randomly select `5` features to consider at every split point.

Instructions

1.
We've given you access to the code that finds the best feature to split on. Right now, it considers all possible features. We're going to want to change that!

For now, let's see what the best feature to split the dataset is. At the bottom of your code, call `find_best_split()` using `data_subset` and `labels_subset` as parameters and print the results.

This function returns the information gain and the index of the best feature. What was the index?

That index corresponds to the features of our car. For example, if the best feature index to split on was `0`, that means we're splitting on the price of the car.

2.
We now want to modify our `find_best_split()` function to only consider a subset of the features. We want to pick `3` features *without replacement*.

The `random.choice()` function found in Python's numpy module can help us do this. `random.choice()` returns a list of values between `0` and the first parameter. The size of the list is determined by the second parameter. And we can choose without replacement by setting `replace = False`.

For example, the following code would choose ten unique numbers between `0` and `100` (exclusive) and put them in a list.

```
lst = np.random.choice(100, 10, replace = False)
```

Inside `find_best_split()`, create a list named `features` that contains `3` numbers between `0` and `len(dataset[0])`.

Instead of looping through `feature in range(len(dataset[0]))`, loop through `feature in features`.

Now that we've implemented feature bagging, what is the best index to use as the split index?

```
from tree import car_data, car_labels, split, information_gain
import random
import numpy as np
np.random.seed(1)
random.seed(4)

def find_best_split(dataset, labels):
    best_gain = 0
    best_feature = 0
    #Create features here
    features = np.random.choice(len(dataset[0]), 3, replace=False)
    for feature in features:
        data_subsets, label_subsets = split(dataset, labels, feature)
        gain = information_gain(labels, label_subsets)
        if gain > best_gain:
            best_gain, best_feature = gain, feature
    return best_gain, best_feature

indices = [random.randint(0, 999) for i in range(1000)]

data_subset = [car_data[index] for index in indices]
labels_subset = [car_labels[index] for index in indices]
print(find_best_split(data_subset, labels_subset))
```

```
(0.010225712539814483, 4)
```