5. Test Set

We're now able to create a random forest, but how accurate is it compared to a single decision tree? To answer this question we've split our data into a training set and test set. By building our models using the training set and testing on every data point in the test set, we can calculate the accuracy of both a single decision tree and a random forest.

We've given you code that calculates the accuracy of a single tree. This tree was made without using any of the bagging techniques we just learned. We created the tree by using every row from the training set once and considered every feature when splitting the data rather than a random subset.

Let's also calculate the accuracy of a random forest and see how it compares!

Instructions

1.

Begin by taking a look at the code we've given you. We've created a single tree using the training data, looped through every point in the test set, counted the number of points the tree classified correctly and reported the percentage of correctly classified points — this percentage is known as the accuracy of the model.

Run the code to see the accuracy of the single decision tree.

2.

Right below where `tree` is created, create a random forest named `forest` using our `make_random_forest()` function.

This function takes three parameters — the number of trees in the forest, the training data, and the training labels. It returns a list of trees.

Create a random forest with `40` trees using `training_data` and `training_labels`.

You should also create a variable named `forest_correct` and start it at `0`. This is the variable that will keep track of how many points in the test set the random forest correctly classifies.

3.

For every data point in the test set, we want every tree to classify the data point, find the most common classification, and compare that prediction to the true label of the data point. This is very similar to what you did in the previous exercise.

To begin, at the end of the for loop outside the if statement, create an empty list named `predictions`. Next, loop through every `forest_tree` in `forest`. Call `classify()` using `testing_data[i]` and `forest_tree` as parameters and append the result to `predictions`.

4.

After we loop through every tree in the forest, we now want to find the most common prediction and compare it to the true label. The true label can be found using `testing_labels[i]`. If they're equal, we've correctly classified a point and should add `1` to `forest_correct`.

An easy way of finding the most common prediction is by using this line of code:

```
forest_prediction = max(predictions,key=predictions.count)
```

5.

Finally, after looping through all of the points in the test set, we want to print out the accuracy of our random forest. Divide `forest_correct` by the number of items in the test set and print the result.

How did the random forest do compared to the single decision tree?

```
from tree import training_data, training_labels, testing_data, testing_labels, make_random_fo
rest, make_single_tree, classify
import numpy as np
import random
np.random.seed(1)
random.seed(1)
from collections import Counter

tree = make_single_tree(training_data, training_labels)
single_tree_correct = 0

forest = make_random_forest(40, training_data, training_labels)
forest_correct = 0

for i in range(len(testing_data)):
  prediction = classify(testing_data[i], tree)
  if prediction == testing_labels[i]:
    single_tree_correct += 1
  predictions = []
  for forest_tree in forest:
    predictions.append(classify(testing_data[i], forest_tree))
  forest_prediction = max(predictions,key=predictions.count)
  if forest_prediction == testing_labels[i]:
    forest_correct += 1

print(single_tree_correct/len(testing_data))
print(forest_correct/len(testing_data))
```

```
0.8815028901734104
0.9219653179190751
```