Machine Learning and Data Mining — Homework 4

Philip Warton

December 1, 2021

1 Exercises: Decision Trees and Ensembles

Q1

a. Draw the decision boundaries defined by this tree over the interval $x_1 \in [0, 30]$, $x_2 \in [0, 30]$. Each leaf of the tree is labeled with a letter. Write this letter in the corresponding region of input space.

b. Give another decision tree that is syntactically different (i.e., has a different structure) but defines the same decision boundaries.

c. This demonstrates that the space of decision trees is syntactically redundant. How does this redundancy influence learning — i.e., does it make it easier or harder to find an accurate tree?

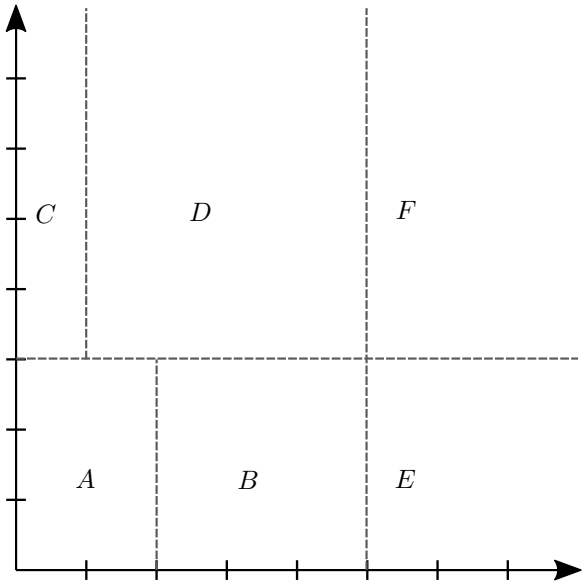

Figure 1: Decision boundary given by tree in Q1

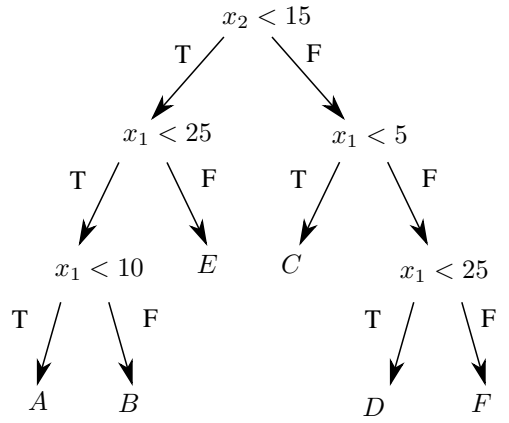

Figure 2: An equivalent – but different – binary decision tree

a.

The decision boundary determined by the given tree is illustrated in Figure 1.

b.

The equivalent tree that is syntatically different is illustrated in Figure 2.

c.

The redundancy phenomenon makes it much easier for learning, since it means that there are many different ways to get to the correct solution, so we will still have logical equivalence under different starting conditions, making learning not have to be as strict/stringent to find the exact unique solution desired.

Q2

Consider the following training set and learn a decision tree to predict Y. Use information gain to select attributes for splits.

A	B	C	Y
0	1	1	0
1	1	1	0
0	0	0	0
1	1	0	1
0	1	0	1
1	0	1	1

For each candidate split include the information gain in your report. Also include the final tree and your training accuracy.

Q3

a. Apply bagging by uniformly sampling train datapoints with replacement to train each ensemble member.

b. The sklearn API for the DecisionTreeClassifier provides many options to modify how decision trees are learned, including some of the techniques we discussed to increase randomness. When set less than the number of features in the dataset, the

`max_features`

argument will cause each split to only consider a random subset of the features. Modify line 44 to include this option at a value you decide.

a.

We apply bagging to the decision try ensemble members by modifying lines 39-42 as follows:

```
n = len(X_train)
rand_idx = np.random.choice(n, n, replace=True)
X_data = X_train[rand_idx,:]
y_data = y_train[rand_idx]
```

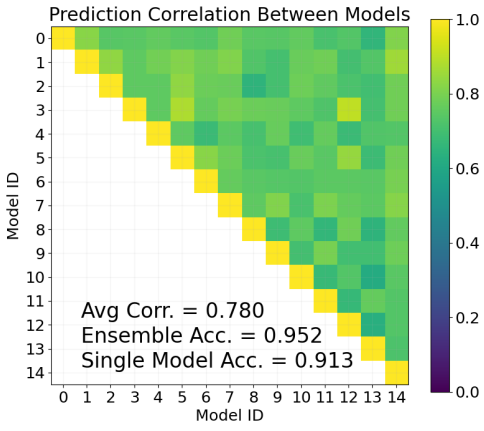

Figure 3: Prediction correlation between models with ensemble -wise bagging

b.

In order to change the maximum number of features learned, we modify line 44 to be

```
clf = tree.DecisionTreeClassifier(criterion="entropy", max_features="auto")
```

What this does is it changes the maximum number of features to be about $\sqrt{n}$ (n being the number of features total), which results in a similar kind of correlation graph as in Part a.

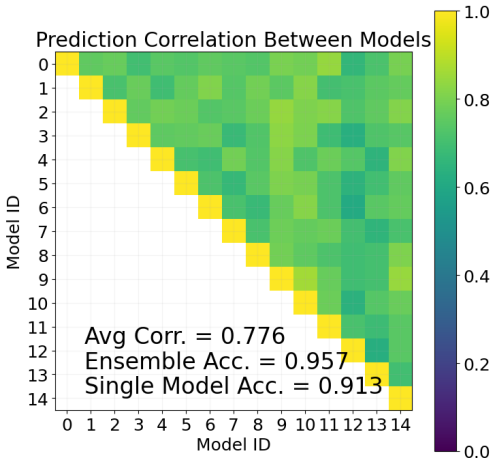

Figure 4: Prediction correlation between models with $\sqrt{n}$ max features

2 Implementation: k-Means Clustering

Q4

The provided skeleton code file kmeans.py implements the main k-means function but includes stubs for the following functions: initializeCentroids, computeAssignments, updateCentroids, and calculateSSE. Implement these functions to finish the implementation. The code provides input/output specifications

a.

```
def initalizeCentroids(dataset, k):

  # Choose k distinct random indices and set our centroids to their
  # corresponding points
  rand_idx = np.random.choice(len(dataset), size=k, replace=False)
  centroids = dataset[rand_idx,:]
  return centroids
```

b.

```
def computeAssignments(dataset, centroids):

  # Set length variable for readability
  n = len(dataset)

  # Initialize vector for assignment idx
  assignments = np.zeros((n,1))

  # Loop over each point in dataset
  for i in range(0,n):
    x_i = dataset[i]

    # Take difference of x_i and each centroid
    # and take their norms
    norms = np.linalg.norm(centroids - x_i, axis=1)

    # Get idx of closest centroid and set assign to point
    assignments[i] = np.argpartition(norms, kth=0)[0:1]
```

```
  return assignments
```

c.

```
def updateCentroids(dataset, centroids, assignments):
  counts = []
  for j in range(0, len(centroids)):

    # To count number of x_i assigned to c_j
    num_assigned = 0

    # To sum them
    sum_assigned = np.zeros(dataset[0].shape)

    # Here we preform the sum
    for i in range(0, len(dataset)):

      # If point x_i is assigned to c_j
      if j == assignments[i]:

        # Iterate count and add it to sum
        num_assigned += 1
        sum_assigned += dataset[i]

    counts.append(num_assigned)

    # Avg of assigned points becomes new centroid
    centroids[j] = sum_assigned / num_assigned

  return centroids, counts
```

d.

```
def calculateSSE(dataset, centroids, assignments):
  sse = 0
  for i in range(0, len(assignments)):
    error = np.linalg.norm(dataset[i] - centroids[int(assignments[i])])
    sq_error = error ** 2
    sse += sq_error
  return sse
```

Q5

In order to see the random distribution as desired, we modify our code to be

```
for i in range(0, 50):
  SSE = kMeansClustering(X, k=k, max_iters=max_iters, visualization=False)[2]
  SSE_rand.append(SSE)
```

This ultimately gives us the histogram for the distribution of random SSE's given by Figure 5 What we see looks like what could possibly be a uniform distribution maybe with a lot of noise. Regardless, it is clear that our SSE can vary randomly quite a bit, and perhaps it would be good to ensure that we are running the algorithm several times and choosing the best model, when we use this on real life data.

Q6

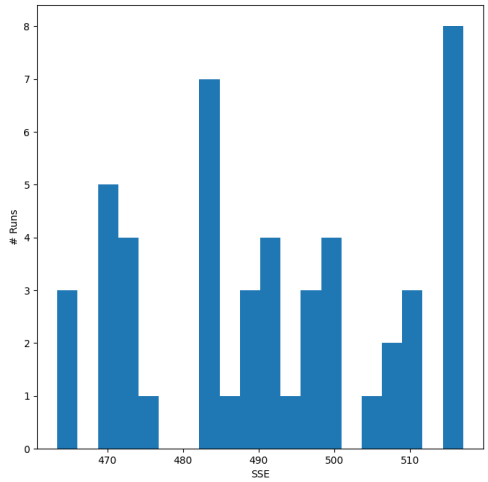

Figure 5: SSE histogram on 50 random trials

One important question in k-means is how to choose k. In prior exercises, we've chosen hyperparmeters like this based on performance on some valdiation set; however, k-means is an unsupervised method so we don't have any labels to compute error. One idea would be to pick k such that the sum-of- squared-errors is minimized; however, this ends up being a bad idea – let's see why. Finish the following code on the toyProblem function to run k-means with $k = 1, 2, \cdots, 150$ on the toy dataset and plot the final SSE achieved for each clustering.

To run this experiment, we implement the following code

```
for i in range(0,150):
    SSE = kMeansClustering(X, k=i+1, max_iters=20, visualize=False)[2][-1]
    SSE_vs_k.append(SSE)
```

Then, we see an exponentially decaying relationship between SSE and k Figure 6. This means that if we minimize SSE we will always end up with $k = n$, which is not at all useful.

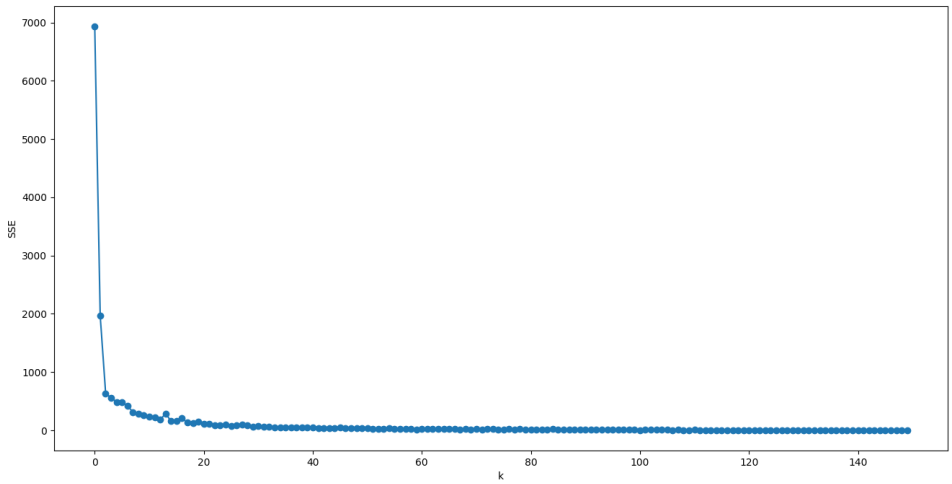

Figure 6: SSE vs k for the toy problem

Q6

Inside kmeans.py, the imageProblem function runs your k-means algorithm on this set of images and features and then visualizes the resulting clusters by showing up to 50 samples. By default, the code runs with k = 10 for this dataset. Answer:

a. From the images displayed, describe what types of images are in the dataset. Does the default parameter of k = 10 seem to be too high, too low, or fine as it is?

b. Adjust k until you are happy with the clusters you observe. Include the samples from your clusterings.

c. Compare the SSE for your chosen k against the SSE from k = 10. Is SSE a good indicator of clustering quality?

a.

By the looks of it, there are vaguely 3 different kinds of images. There are images of skyscrapers, empty roads, and lush greenery. So, I would guess that $k = 10$ is too high because there are multiple clusters that seem very similar in character.

b.

I started by adjusting k to 3, and as it turns out this was the best one that I could ultimately come up with. In Figures 7-9, a sample from each cluster is displayed.

c.

For the SSE, we have

$$k = 3 \Rightarrow \qquad SSE = 2963.57$$
$$k = 10 \Rightarrow \qquad SSE = 2375.55$$

As we can see, the SSE is much smaller for $k = 10$ than for $k = 3$. However, similar to our answer for Q6 we conclude that SSE is not a reliable metric to minimize, since more clusters are always favored. So, we say that no, SSE is not a good indicator of cluster quality.

Q8

Here we have our figures of the samples of our clusters, and the associated purities of each, given in Figures 7-9.

3 Debriefing

1. 12 hours total on this assignment
2. Assigment was moderate difficulty – Q2 was tough
3. Worked mostly alone on this, recieved some advice on Q2
4. I understand the material about 50%. I don't feel good about the decision tree stuff, but k-means I feel is quite easy to have a grasp of.
5. No other comments

Figure 7: Cluster 1 — Roads
numViolations = 4
purity = 92%

Figure 8: Cluster 2 — Buildings
numViolations = 2
purity = 96%

Figure 9: Cluster 3 — Greenery
numViolations = 10
purity = 80%