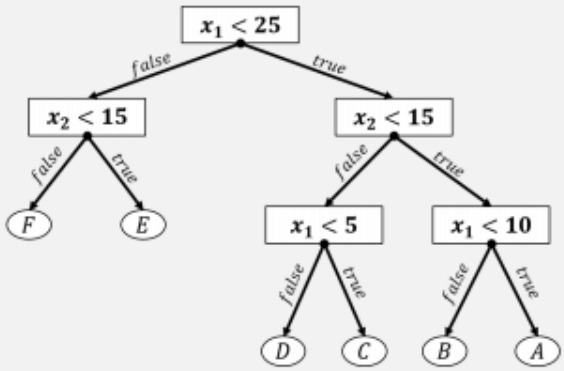
Brandon Withington
CS434 Machine Learning & Datamining

5 / 29 / 2021

▶ Q1 Drawing Decision Tree Predictions [2pts]. Consider the following decision tree:

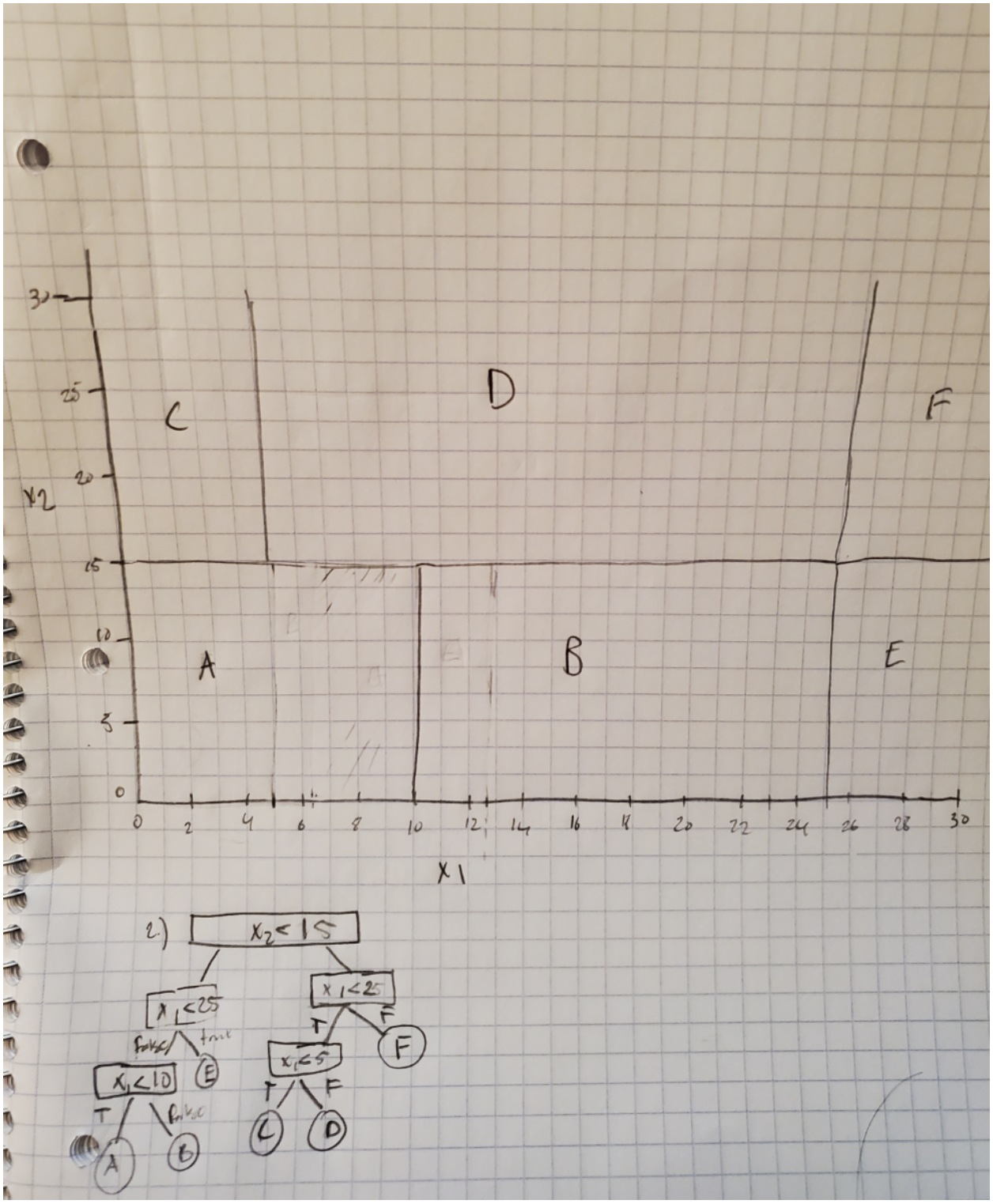


a) Draw the decision boundaries defined by this tree over the interval $x_1 \in [0, 30]$, $x_2 \in [0, 30]$. Each leaf of the tree is labeled with a letter. Write this letter in the corresponding region of input space.

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?

A and B are in the photo posted below


30
25
20
15
10
5
0
X2
C
D
F
A
B
E
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
X1
2.)
$X_2 < 15$
$X_1 < 25$
$X_1 < 25$
false
true
T
F
$X_1 < 10$
$X_1 < 5$
E
F
T
F
T
F
A
B
C
D

c.) I do not believe it would be a statistical problem. It has no impact on how the way the decision tree is drawn. The same area of decision trees can be represented both with or without the presence of redundancy. It is in fact likely to be a computational advantage as imperfect greedy heuristics could be easier to find and therefore a better solution could be found in an easier fashion.

▶ Q2 Manually Learning A Decision Tree [2pts]. 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.

Gotta make a prediction for Y in any case.

Given the data set the list of probabilities comes from the given data table :
Class prior : p(y = 1) = ½
p(y = 0) = ½
Conditional probability for y = 1 :

p(A = 0 | y = 1) = ⅓

p(B = 0 | y = 1) = ⅓

p(C = 0 | y = 1) = ⅔

Conditional probability for y = 0 :

p(A = 0 | y = 0) = ⅔

p(B = 0 | y = 0) = ⅓

p(C = 0 | y = 0) = ⅓

Predicting both y = 1 and y = 0
Setting A = 1, B = 0, C = 0.

$$P(Y = 1 \mid X) = \frac{P(Y=1) * P(A=1 \mid Y=1)P(B=0 \mid Y=1)P(C=0 \mid Y=1)}{P(A=1, B=0, C=0)} = \frac{\frac{2}{27}}{K}$$

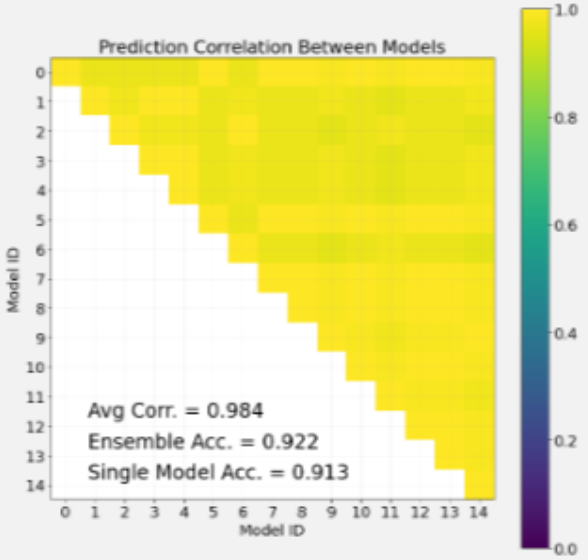
$$P(Y = 0 \mid X) = \frac{P(Y=0) * P(A=1 \mid Y=0)P(B=0 \mid Y=0)P(C=0 \mid Y=0)}{P(A=1, B=0, C=0)} = \frac{\frac{1}{54}}{K}$$

$$P(Y = 0 \mid X) + P(Y = 1 \mid X) = 1$$

$$K = \frac{2}{27} + \frac{1}{54} = 0.0925925926$$

$$P(Y = 1 \mid X) = 4/5 \quad P(Y = 0 \mid X) = 1/5$$

▶ Q3 Measuring Correlation in Random Forests [1pts]. We've provided a Python script `decision.py` that trains an ensemble of 15 decision trees on the Breast Cancer classification dataset we used in HW1. We are using the `sklearn` package for the decision tree implementation as the point of this exercise is to consider ensembling, not to implement decision trees. When run, the file displays the plot:



The non-empty cells in the upper-triangle of the figure show the correlation between predictions on the test set for each of 15 decision tree models trained on the same training set. Variations in the correlation are due to randomly breaking ties when selecting split attributes. The plot also reports the average correlation (a very high 0.984 for this ensemble) and accuracy for the ensemble (majority vote) and a separately-trained single model. Even with the high correlation, the ensemble managed to improve performance marginally.

As discussed in class, uncorrelated errors result in better ensembles. Modify the code to train the following ensembles (each separately). Provide the resulting plots for each and describe what you observe.

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.)

```python
#Change the X_data and y_data to look at np.zeroes value of either of the shapes of both X and Y training data
X_data = np.zeros(X_train.shape)
y_data = np.zeros(y_train.shape)

random_values = random.choices(range(0, X_train.shape[0]), k = X_train.shape[0])

#Doing this for X
for i in range(len(random_values)):
    X_data[i] = X_train[random_values[i], :]

#Now doing this for Y
for i in range(len(random_values)):
    y_data[i] = y_train[random_values[i]]
```

b.)

```python
clf = tree.DecisionTreeClassifier("entropy", max_features="sqrt")
```

▶ Q4 Implement k-Means [10pts]. 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) `initializeCentroids` – Given the dataset and a integer k produce k centroid vectors. We recommend doing this by selecting k random points from the dataset, but you are welcome to try alternatives.

b) `computeAssignments` – Given the dataset and a set of centroids, implement Eq.2 to produce a vector of assignments where the i'th entry is the id of the centroid nearest to point x_i. This will involve computing distances between centroids and the dataset and will take up most of the computation in k-means. To keep things running fast, we strongly recommend using similar efficient matrix calculations as in HW1's implementation of kNN. See the HW1 solutions if you did not use these fast operations.

c) `updateCentroids` – Given the dataset, the centroids, and the current assignments, implement Eq.3 to produce a new matrix of centroids where the j'th row stores the j'th centroid. Also return the count of datapoints assigned to each centroid as a vector.

d) `calculateSSE` – Given the dataset, the centroids, and the current assignments, implement Eq.1 to calculate the sum-of-squared-errors for a clustering. This will be a scalar value. We will use this to track progress in the clustering and compare across clusterings.

When executing `kmeans.py`, k-means with $k = 3$ will be applied to the toy problem shown above with three Gaussian blobs and the result will be plotted. Use this to verify your implementation is accurate – note that depending on your choice of initialization you may get different colors for clusters.

Hint: If you want help debugging your algorithm, you can set `visualize=True` in the k-means function call to have each iteration plotted to look at intermediate steps.

a.)

```
def initalizeCentroids(dataset, k):
  return dataset[np.random.choice(dataset.shape[0], size=k, replace=False), :]
```

b.)

```
def computeAssignments(dataset, centroids):
  overall_dataset = dataset.shape[0]
  assigned_vals = np.zeros((overall_dataset))
  for i in range(overall_dataset):
    assigned_vals[i] = np.argmin(np.linalg.norm(dataset[i,:] - centroids, axis = 1))
  return assigned_vals
```

c.)

```
def updateCentroids(dataset, centroids, assignments):
  #initialize these before utilizing!
  count_of_centroids = np.zeros((centroids.shape[0], 1))
  all_sums = np.zeros(centroids.shape)

  #Update centroids for the length of assignments that have occurred.
  for i in range(len(assignments)):
    count_of_centroids[int(assignments[i])] = count_of_centroids[int(assignments[i])] + 1
    all_sums[int(assignments[i])] = all_sums[int(assignments[i])] + dataset[i]

  return (all_sums / count_of_centroids), count_of_centroids
```

d.)

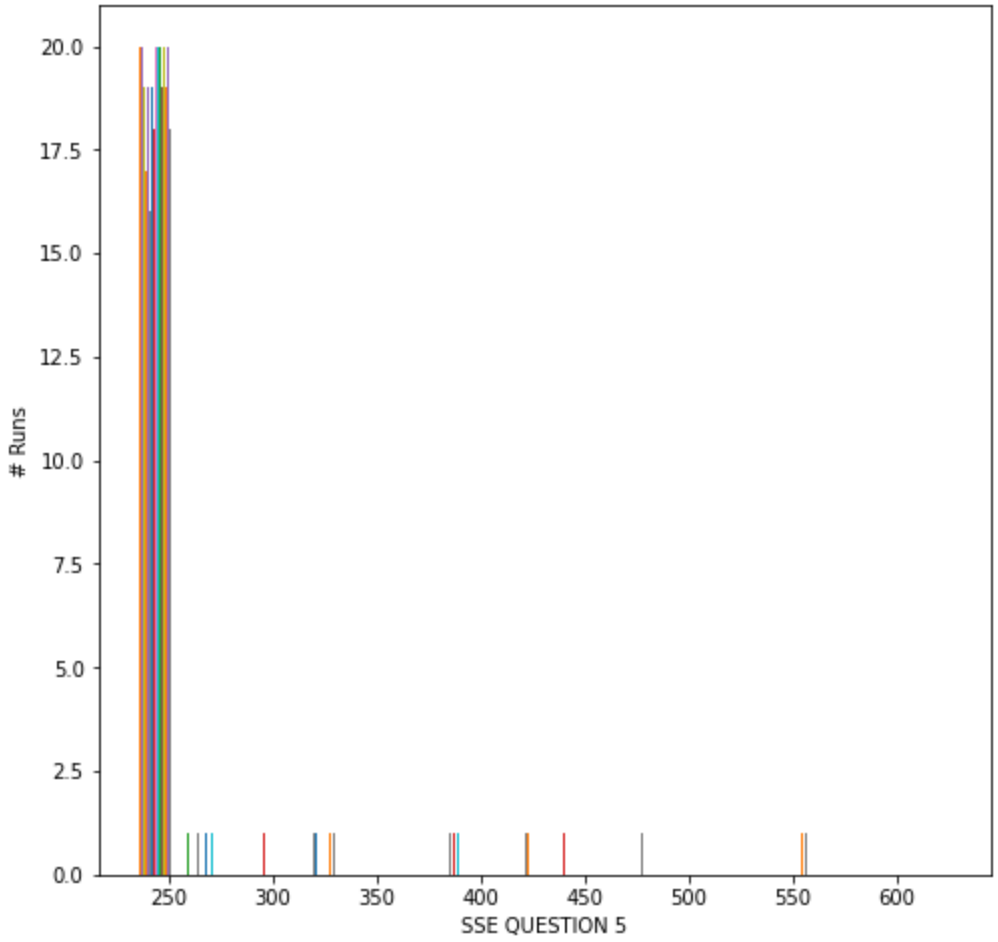
```
def calculateSSE(dataset, centroids, assignments):
  #Initialize before use.
  SSE = 0
  amount_of_centroids = 0
  for i in range(len(assignments)):
    amount_of_centroids = centroids[int(assignments[i])]
    SSE = SSE + np.linalg.norm(dataset[i, :] - amount_of_centroids)

  return SSE
```

▶ **Q5 Randomness in Clustering [2pt].** k-Means is fairly sensitive to how the centroids are initialized. Finish the following code in the `toyProblem` function to run k-means with $k = 5$ fifty times on the toy dataset and plot the final SSE achieved for each clustering.

```
##############################
# Q5 Randomness in Clustering
##############################
k = 5
max_iters = 20

SSE_rand = []
# Run the clustering with k=5 and max_iters=20 fifty times and
# store the final sum-of-squared-error for each run in the list SSE_rand.
raise Exception('Student error: You haven\'t implemented the randomness
  experiment for Q5.')

# Plot error distribution
plt.figure(figsize=(8,8))
plt.hist(SSE_rand, bins=20)
plt.xlabel("SSE")
plt.ylabel("# Runs")
plt.show()
```

Provide the resulting plot and discuss how this might affect how you apply k-means to a real dataset.

```
##############################
# Q5 Randomness in Clustering
##############################
k = 5
max_iters = 20

SSE_rand = []
# Run the clustering with k=5 and max_iters=20 fifty times and
# store the final sum-of-squared-error for each run in the list SSE_rand.

#Set k = 5, max_iters to 20 and run through the for loop 50 times.
#looking at line 19
k = 5
max_iters=20
for i in range(50):
  centroids, assignments, SSE = kMeansClustering(X, k=k, max_iters=max_iters, visualize=False)
  SSE_rand.append(SSE)
```

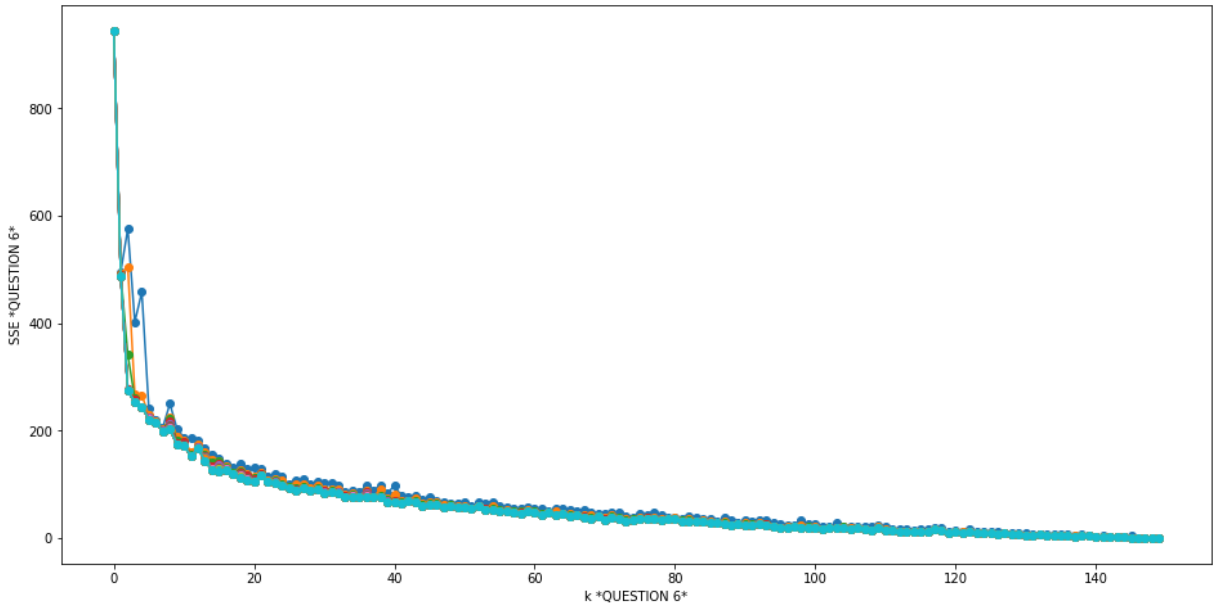


▶ Q6 Error Vs. K [2pt]. One important question in k-means is how to choose k. In prior exercises, we've chosen hyperparmeters like this based on performance on some valdiation set; however, k-means is an unsupervised method so we don't have any labels to compute error. One idea would be to pick k such that the sum-of-squared-errors is minimized; however, this ends up being a bad idea – let's see why.

Finish the following code tn the `toyProblem` function to run k-means with $k = 1, 2, ..., 150$ on the toy dataset and plot the final SSE achieved for each clustering.

```
########################
# Q6 Error vs. K
########################

SSE_vs_k = []
# Run the clustering max_iters=20 for k in the range 1 to 150 and
# store the final sum-of-squared-error for each run in the list SSE_vs_k.
raise Exception('Student error: You haven\'t implemented the k experiment for
  Q6.')

# Plot how SSE changes as k increases
plt.figure(figsize=(16,8))
plt.plot(SSE_vs_k, marker="o")
plt.xlabel("k")
plt.ylabel("SSE")
plt.show()
```

Provide the resulting plot and discuss why choosing k based on the sum-of-squared error doesn't make sense.



▶ Q7 Clustering Images. [4pt] 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 default, these are the images that are produced by the imageProblem. Most of it seems pretty accurate however there is one outlier!

Because of this I believe it is a little too high. Other clusters are largely blank which points to misclassification in a lot of subjects..

b.) Setting k = 5 still leaves some spots blank. First cluster :

Second cluster still has the panda :

Third cluster looks mostly good, it seems that roads and skyscrapers are here but ideally they would be separate :

Fourth cluster, it’s mostly ok, there are some skyscrapers that do not belong here :

Fifth cluster it looks pretty alright :

Overall this run looked pretty ok. There were some misclassified things so I believe it's still too high. Throughout the next run I'll be looking at k = 4.

First cluster, Pretty accurate! :

Second cluster, mostly accurate, the panda showed up here and it should not have :

Third cluster, pretty disorganized. There are still skyscrapers among the trees and a couple roads here and there.. :

Fourth cluster spot on :

Setting k = 2, 4 seemed to give a little more accuracy over 5, I'm going to skip over 3 and see if k = 2 can give some good looking results.

First cluster very accurate, no sign of panda :

Second cluster not as accurate, still no panda though :

I am not really satisfied with this. The second cluster seemed pretty messy, and I believe there are three categories of photos in this, city-scapes, roads and trees. Maybe setting k = to the number of categories will help it sort those into particular groups. Setting k = 3.

First cluster k = 3 :

Second cluster :

Third cluster :

This seems pretty good. There are still some things that get misclassified here and there but for the most part each cluster seems to be pretty accurate in terms of grouping similar photos together.

c.) Comparing the three clusters generated from k = 3 to the ten clusters generated from k = 10, I can firmly say that k = 3 did a better job than k = 10 in terms of clustering like-images together. The panda has stopped appearing now too.

▶ Q8 Evaluating Clustering as Classification [2pt] Suppose after you finish clustering, you want to assign a name to each cluster corresponding to what is in them.

1. Provide a "label" for each of your clusters.
2. Estimate the purity of your clusters by counting the number of examples in the plots that seem to violate your labelling, divided by the total number of images shown.

1.)

First cluster k = 3 :

Label = trees, forests and nature in general.

Second cluster :

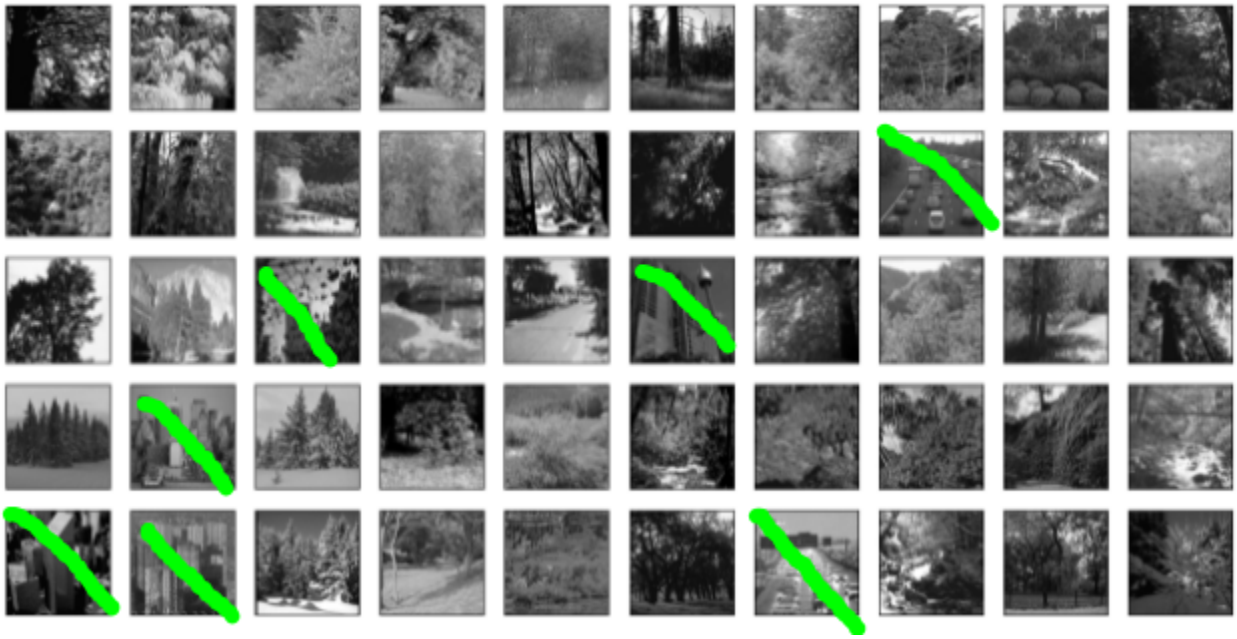
Label = roads, cars and pavement.

Third cluster :

Label = cities, skyscrapers, and other big tall buildings

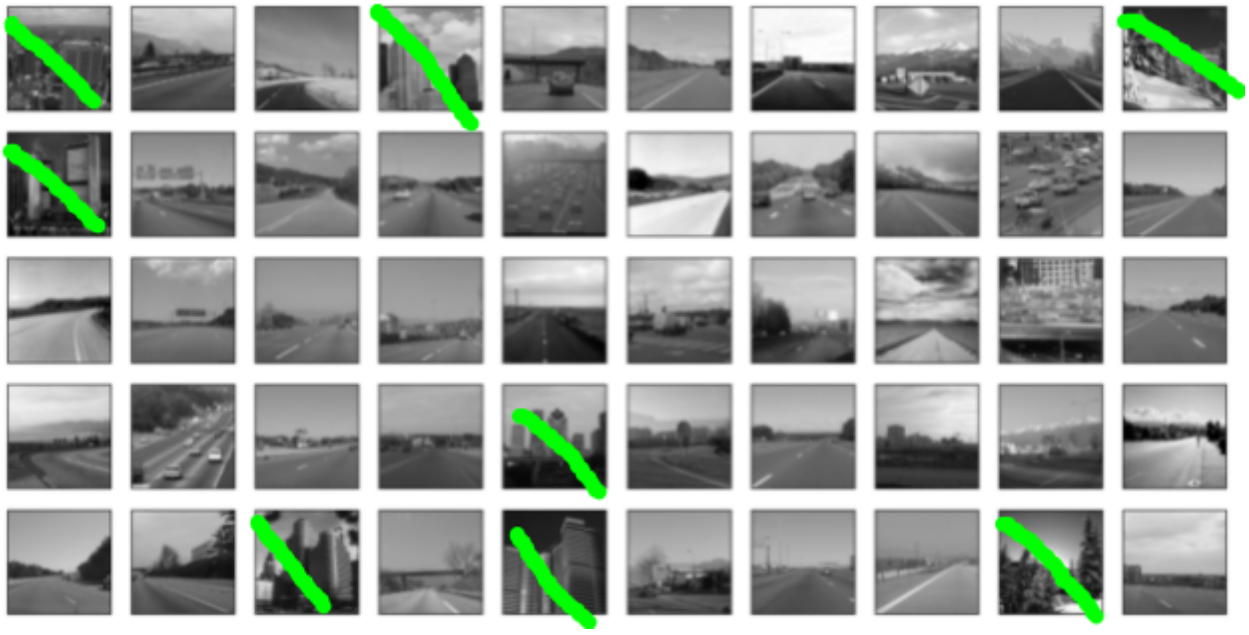
2.) In each cluster there are 50 photos. With three clusters that means there are a total of 150 photos.

Cluster 1 :



43 / 50 = 86%

Cluster 2 :



42 / 50 = 84%

Cluster 3 :

49 / 50 = 98%

Overall = 134 correct images out of 150 = 89.3333333% overall purity.

3 Debriefing (required in your report)

1. Approximately how many hours did you spend on this assignment?
2. Would you rate it as easy, moderate, or difficult?
3. Did you work on it mostly alone or did you discuss the problems with others?
4. How deeply do you feel you understand the material it covers (0%–100%)?
5. Any other comments?

1.) Approximately 13 hours.
2.) Moderate.
3.) Alone.
4.) 70%.
5.) The last part of this was pretty fun.