CPSC 340 Assignment 2 (due Friday January 26th at 9:00pm)

Instructions

Rubric: {mechanics:5}

The above points are allocated for compliance with the CPSC 340 homework submission instructions:
`https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/home/blob/master/homework_instructions.md`

NOTE: for this assignment you'll need to separately download the data from the home repo. It can't be delivered in the normal way due to a limitation of the way we're using GitHub.

1 Naive Bayes

In this section we'll implement naive Bayes, a very fast classification method that is often surprisingly accurate for text data with simple representations like bag of words.

1.1 Naive Bayes by Hand

Rubric: {reasoning:3}

Consider the dataset below, which has 10 training examples and 2 features:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Suppose you believe that a naive Bayes model would be appropriate for this dataset, and you want to classify the following test example:

$$\hat{x} = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$

(a) Compute the estimates of the class prior probabilities (you don't need to show any work):

- $p(y = 1)$.
  Answer: 6/10
- $p(y = 0)$.
  Answer: 4/10

(b) Compute the estimates of the 4 conditional probabilities required by naive Bayes for this example (you don't need to show any work):

- $p(x_1 = 1|y = 1)$.
  Answer: 1/2
- $p(x_2 = 0|y = 1)$.
  Answer: 1/3
- $p(x_1 = 1|y = 0)$.
  Answer: 1
- $p(x_2 = 0|y = 0)$.
  Answer: 3/4

(c) Under the naive Bayes model and your estimates of the above probabilities, what is the most likely label for the test example? (Show your work.)
Answer: The probability that the label for test example is 1 is:
$p(y = 1|x1 = 1, x2 = 0) = p(x1 = 1|y = 1)*p(x2 = 0|y = 1)*p(y = 1) = 1/2 * 1/3 * 3/5 = 1/10.$

The probability that the label for test example is 0 is:
$p(y = 0|x1 = 1, x2 = 0) = p(x1 = 1|y = 0)*p(x2 = 0|y = 0)*p(y = 0) = 1 * 3/4 * 2/5 = 3/10.$

Therefore, the most likely label 0.

1.2 Bag of Words

Rubric: {reasoning:3}

If you run `python main.py -q 1.2`, it will load the following dataset:

1. X: A sparse binary matrix. Each row corresponds to a newsgroup post, and each column corresponds to whether a particular word was used in the post. A value of 1 means that the word occured in the post.
2. *wordlist*: The set of words that correspond to each column.
3. y: A vector with values 1 through 4, with the value corresponding to the newsgroup that the post came from.
4. *groupnames*: The names of the four newsgroups.
5. $Xvalidate$ and $yvalidate$: the word lists and newsgroup labels for additional newsgroup posts.

Answer the following:

1. Which word corresponds to column 50 of X?
   Answer: Lunar
2. Which words are present in training example 500?
   Answer: car, fact, gun, video
3. Which newsgroup name does training example 500 come from?
   Answer: talk.*

1.3 Naive Bayes Implementation

Rubric: {code:5}

If you run `python main.py -q 1.3` it will load the newsgroups dataset and report the test error for a random forest, and also fit the basic naive Bayes model and report the test error.

The `predict()` function of the naive Bayes classifier is already implemented. However, in `fit()` the calculation of the variable `p_xy` is incorrect (right now, it just sets all values to 1/2). Modify this function so that `p_xy` correctly computes the conditional probabilities of these values based on the frequencies in the data set. Hand in your code and the validation error that you obtain. Also, briefly comment on the accuracy as compared to the random forest and scikit-learn's naive Bayes implementation.

Answer: URL for naive_bayes.py code: `https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/g5e0b_u7p1b_a2/blob/master/code/naive_bayes.py`.
Answer: The validation error obtained is 0.188. This validation error was better than Random Forest (sklearn's) validation error, which was 0.198. This validation error was slightly worse than sklearn's Naive Bayes validation error, which was 0.187

1.4 Laplace smoothing

Rubric: {code:1}

Do the following:

1. Modify your code so that it uses Laplace smoothing, with β as a parameter taken in by the constructor.
   Answer: Link to naive_bayes.py: `https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/g5e0b_u7p1b_a2/blob/master/code/naive_bayes.py`.
2. Did you need to modify your fit function, predict function, or both?
   Answer: We only needed to modify the fit function
3. Take a look at the documentation for the scikit-learn version of naive Bayes the code is using. How much Laplace smoothing does it use by default? Using the same amount of smoothing with your code, do you get the same results?
   Answer: The scikit-learn version of naive Bayes uses a smoothing value of 1 by default. Using the same amount of smoothing with our code, we obtained a validation error of 0.188 while scikit-learn's version obtained a validation error of 0.187

1.5 Runtime of Naive Bayes for Discrete Data

Rubric: {reasoning:3}

Assume you have the following setup:

- The training set has n objects each with d features.
- The test set has t objects with d features.
- Each feature can have up to c discrete values (you can assume $c \leq n$).
- There are k class labels (you can assume $k \leq n$)

You can implement the training phase of a naive Bayes classifier in this setup in $O(nd)$, since you only need to do a constant amount of work for each $X(i, j)$ value. (You do not have to actually implement it in this way for the previous question, but you should think about how this could be done). What is the cost of

classifying t test examples with the model?
Answer: The cost of classifying t test examples with the model is O(TDK)

2 Random Forests

2.1 Implementation

Rubric: {reasoning:7}

The file *vowels.pkl* contains a supervised learning dataset where we are trying to predict which of the 11 "steady-state" English vowels that a speaker is trying to pronounce.

You are provided with a `RandomStump` class that differs from `DecisionStump` in two ways: it uses the information gain splitting criterion (instead of classification accuracy), and it only considers $\lfloor\sqrt{d}\rfloor$ randomly-chosen features.[1] You are also provided with a `RandomTree` class that is exactly the same as `DecisionTree` except that it uses `RandomStump` instead of `DecisionStump` and it takes a bootstrap sample of the data before fitting. In other words, `RandomTree` is the entity we discussed in class, which makes up a random forest.

If you run `python main.py -q 2` it will fit a deep `DecisionTree` using the information gain splitting criterion. You will notice that the model overfits badly.

1. Why doesn't the random tree model have a training error of 0?
   Answer: The random tree model doesn't have a training error of 0 since only approximately 63% of the training data is present in the bootstrap sample. Therefore, not all label predictions based on the training data will be correct, resulting in a training error > 0

2. Create a class `RandomForest` in a file called `random_forest.py` that takes in hyperparameters `num_trees` and `max_depth` and fits `num_trees` random trees each with maximum depth `max_depth`. For prediction, have all trees predict and then take the mode, allowing errors to be minimized.
   Answer: Link to RandomForest: `https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/g5e0b_u7p1b_a2/blob/master/code/random_forest.py`.

3. Using 50 trees, and a max depth of ∞, report the training and testing error. Compare this to what we got with a single `DecisionTree` and with a single `RandomTree`. Are the results what you expected? Discuss.
   Answer: Using 50 trees and a max depth of infinity we got a training error of 0 and a test error of 0.174. Using a single DecisionTree we got a training error of 0 and a testing error of 0.367. Using a single RandomTree we got a training error of 0.117 and a testing error of 0.481. The results met with our expectations. The RandomTree generated a positive training error since not all of the training data were present in the bootstrap sample. The RandomTree generated the highest test error, followed by the DecisionTree, and then the RandomForest. These results were expected since the RandomTree utilized the least training data, and thus is likely to be the most biased. RandomForest had the lowest test error, which is expected since it reduces overfitting.

4. Compare your implementation with scikit-learn's `RandomForestClassifier` for both speed and accuracy, and briefly discuss. You can use all default hyperparameters if you wish, or you can try changing them.
   Answer: Our implementation of RandomForest took 21 seconds to execute for 50 trees with an infinite max depth while scikit-learns RandomForestClassifier took 0.16 seconds to execute for the same hyperparameters. The test error for our implementation was 0.189 while the test error for scikit-learns

[1] The notation $\lfloor x\rfloor$ means the "floor" of x, or "x rounded down". You can compute this with `np.floor(x)` or `math.floor(x)`.

implementation was 0.155. The training errors for both implementations were 0. Overall, scikit-learns implementation was faster and had a smaller test error.

2.2 Very-Short Answer Questions

Rubric: {reasoning:3}

1. What is a a disadvantage of using a very-large number of trees in a random forest classifier?
   Answer: A disadvantage of using a very-large number of trees is the runtime will be very slow
2. Your random forest classifier has a training error of 0 and a very high test error. Which ones of the following could help performance?
   (a) Increase the maximum depth of the trees in your forest.
   (b) Decrease the maximum depth of the trees in your forest.
   (c) Increase the amout of data you consider for each tree (Collect more data and use $2n$ objects instead of n).
   (d) Decrease the amount of data you consider for each tree (Use $0.8n$ objects instead of n).
   (e) Increase the number of features you consider for each tree.
   (f) Decrease the number of features you consider for each tree.

   Answer: In order to help performance, one could: (B) decrease the maximum depth of the trees in your forest, (C) Increase the amount of data you consider for each tree (Collect more data and use $2n$ objects instead of n), and (E) decrease the number of features you consider for each tree
3. Suppose that you were training on raw audio segments and trying to recognize vowel sounds. What could you do to encourage the final classifier to be invariant to translation?
   Answer: In order to encourage the final classifier to be invariant to translation one could add transformed data (ex: sound clips consisting of vowel sounds starting after different times (speaking after 0 seconds, 0.1 second, 0.2 seconds of starting the audio clip, etc)) during training

3 Clustering

If you run `python main.py -q 3`, it will load a dataset with two features and a very obvious clustering structure. It will then apply the k-means algorithm with a random initialization. The result of applying the algorithm will thus depend on the randomization, but a typical run might look like this: (Note that the colours are arbitrary – this is the label switching issue.) But the 'correct' clustering (that was used to make the data) is this:

3.1 Selecting among k-means Initializations

Rubric: {reasoning:5}

If you run the demo several times, it will find different clusterings. To select among clusterings for a *fixed* value of k, one strategy is to minimize the sum of squared distances between examples x_i and their means w_{y_i},

$$f(w_1, w_2, \ldots, w_k, y_1, y_2, \ldots, y_n) = \sum_{i=1}^{n} \|x_i - w_{y_i}\|_2^2 = \sum_{i=1}^{n}\sum_{j=1}^{d}(x_{ij} - w_{y_i j})^2.$$

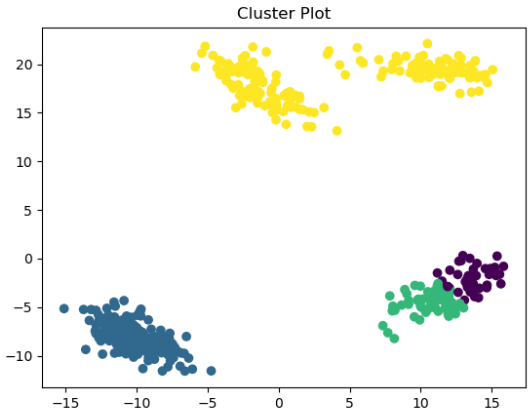


Figure 1: clustering figure 1

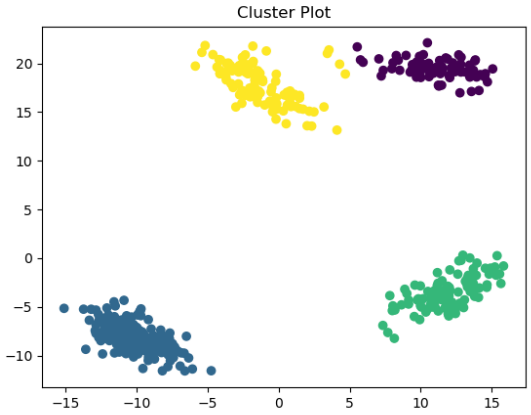


Figure 2: clustering figure 2

where y_i is the index of the closest mean to x_i. This is a natural criterion because the steps of k-means alternately optimize this objective function in terms of the w_c and the y_i values.

1. In the *kmeans.py* file, add a new function called *error* that takes the same input as the *predict* function but that returns the value of this above objective function. Hand in your code.
   Answer: the URL of the code is: `https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/g5e0b_u7p1b_a2/blob/master/code/kmeans.py`

2. What trend do you observe if you print the value of *error* after each iteration of the k-means algorithm?

   Answer: The error monotonically decreases.

3. Using `plot_2dclustering`, output the clustering obtained by running k-means 50 times (with $k = 4$) and taking the one with the lowest error.
   Answer: The minimal error is 3071.468053, and the corresponding clustering is shown below.

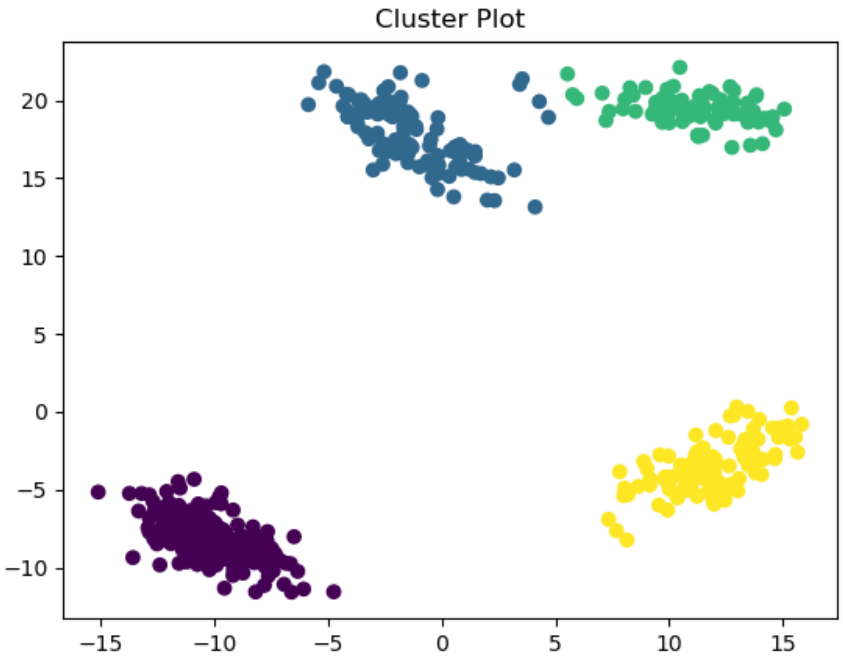


Figure 3: clustering figure 3.1

4. Looking at the hyperparameters of scikit-learn's `KMeans`, explain the first four (`n_clusters`, `init`, `n_init`, `max_iter`) very briefly.
   Answer: `n_clusters` is the number of clusters to form as well as the number of centroids to generate. `init` is method for initialization including k-means++ and random. `n_init` is number of time the k-means algorithm will be run with different centroid seeds. `max_iter` is maximum number of iterations of the k-means algorithm for a single run.

3.2 Selecting k in k-means

Rubric: {reasoning:5}

We now turn to the task of choosing the number of clusters k.

1. Explain why the *error* function should not be used to choose k.
   Answer: It is obvious that if we choose k equals to the number of examples, the error will be 0. Actually the error decreases with the increase of k, thus we only need to choose a larger k. This cannot be the reason to choose k.
2. Explain why even evaluating the *error* function on test data still wouldn't be a suitable approach to choosing k.
   Answer: When k increases, it means the number of clusters increases. For the test data with new data, to get smaller error, choosing a larger k still works.
3. Hand in a plot of the minimum error found across 50 random initializations, as a function of k, taking k from 1 to 10.

Answer: The plot reflecting the relation between k and error is given below.

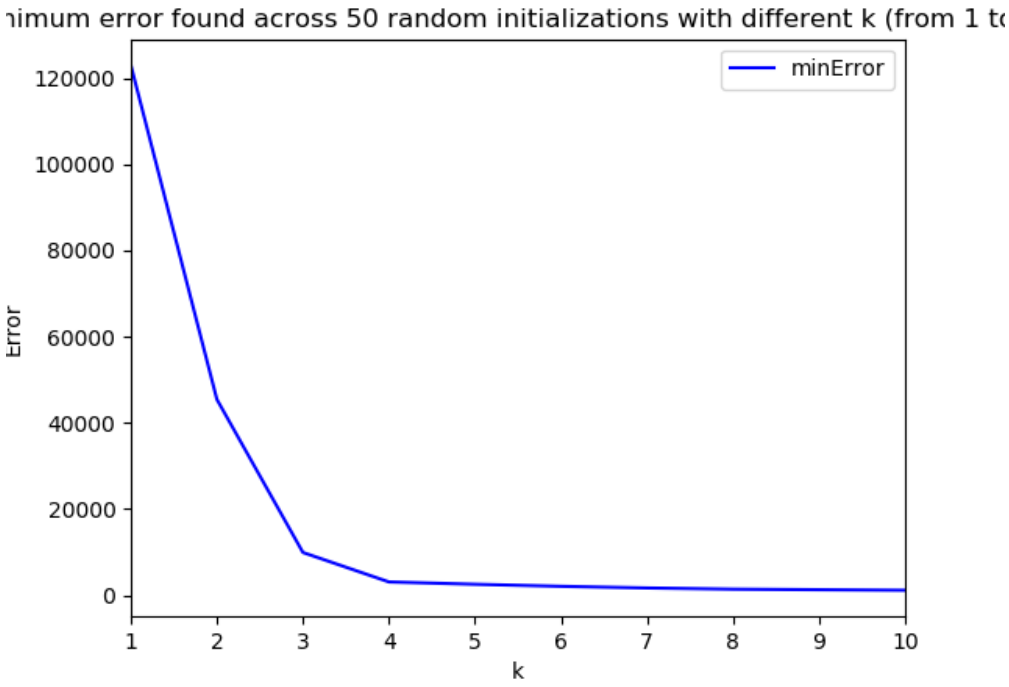


Figure 4: clustering figure 3.2

4. The *elbow method* for choosing k consists of looking at the above plot and visually trying to choose the k that makes the sharpest "elbow" (the biggest change in slope). What values of k might be reasonable according to this method? Note: there is not a single correct answer here; it is somewhat open to interpretation and there is a range of reasonable answers.
   Answer: As it is shown in the plot above, the reasonable k value should be 2, 3 or 4 with a big change in slope.

3.3 k-medians

Rubric: {reasoning:5}

The data in *clusterData2.pkl* is the exact same as the above data, except it has 4 outliers that are very far away from the data.

1. Using the `plot_2dclustering` function, output the clustering obtained by running k-means 50 times (with $k = 4$) and taking the one with the lowest error. Are you satisfied with the result?
   Answer: The plot below shows the kmeans result for clusterData2.
   The min error is 59116.295803 which is too huge. I am not satisfied with the result.
2. What values of k might be chosen by the elbow method for this dataset?
   Answer: The plot below shows the minimum error change with k increasing.

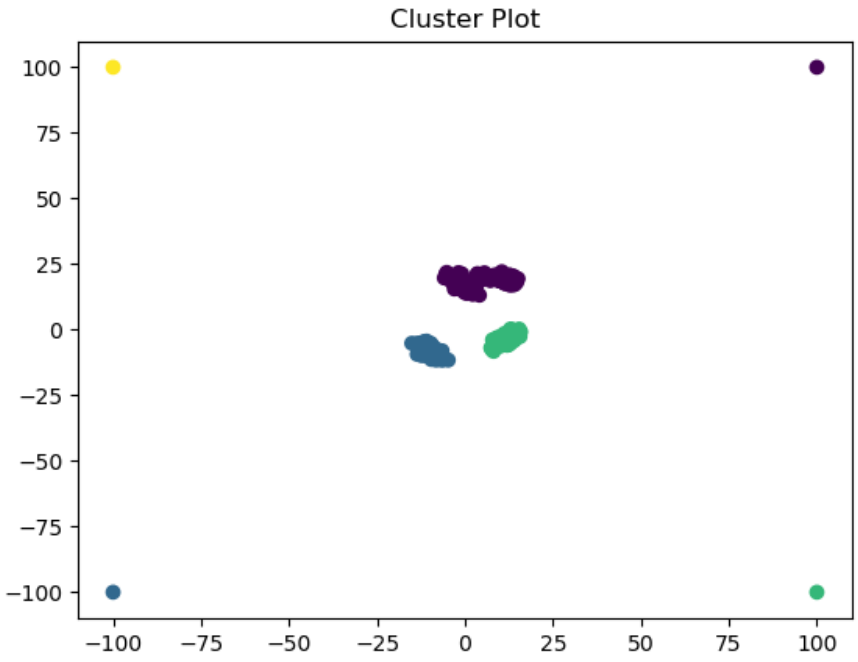


Figure 5: clustering figure 3.3

From this plot, we can find that the reasonable k is 2, 3 or 8. But when k=8, it does not make sense as there are 4 clusters only containing one point and the k=4 does not see a sharp elbow.

3. Implement the *k-medians* algorithm, which assigns examples to the nearest w_c in the L1-norm and to updates the w_c by setting them to the "median" of the points assigned to the cluster (we define the d-dimensional median as the concatenation of the median of the points along each dimension). Hand in your code and plot obtained with 50 random initializations for $k = 4$.
Answer: The URL of the code is: `https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/g5e0b_u7p1b_a2/blob/master/code/kmedians.py`. The plot of the result with k=4 is as below.

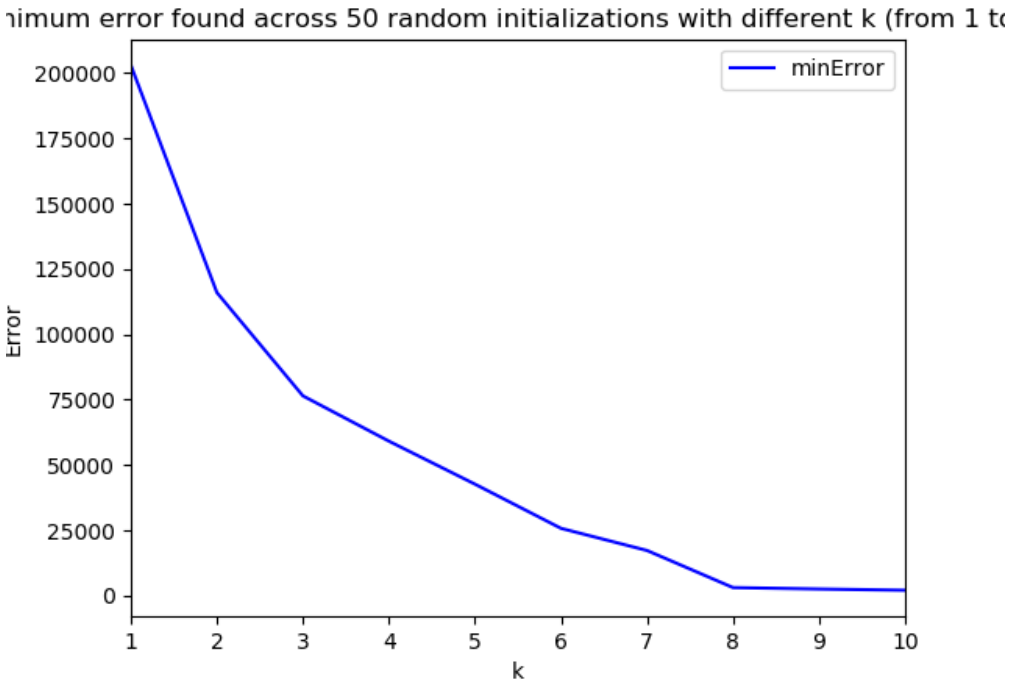


Figure 6: clustering figure 3.3error

4. Using the L1-norm version of the error (where y_i now represents the closest median in the L1-norm),

$$f(w_1, w_2, \ldots, w_k, y_1, y_2, \ldots, y_n) = \sum_{i=1}^{n} \|x_i - w_{y_i}\|_1 = \sum_{i=1}^{n} \sum_{j=1}^{d} |x_{ij} - w_{y_i j}|,$$

what value of k would be chosen by the elbow method under this strategy? Are you satisfied with this result?
Answer: The minimum error plot is shown as below.

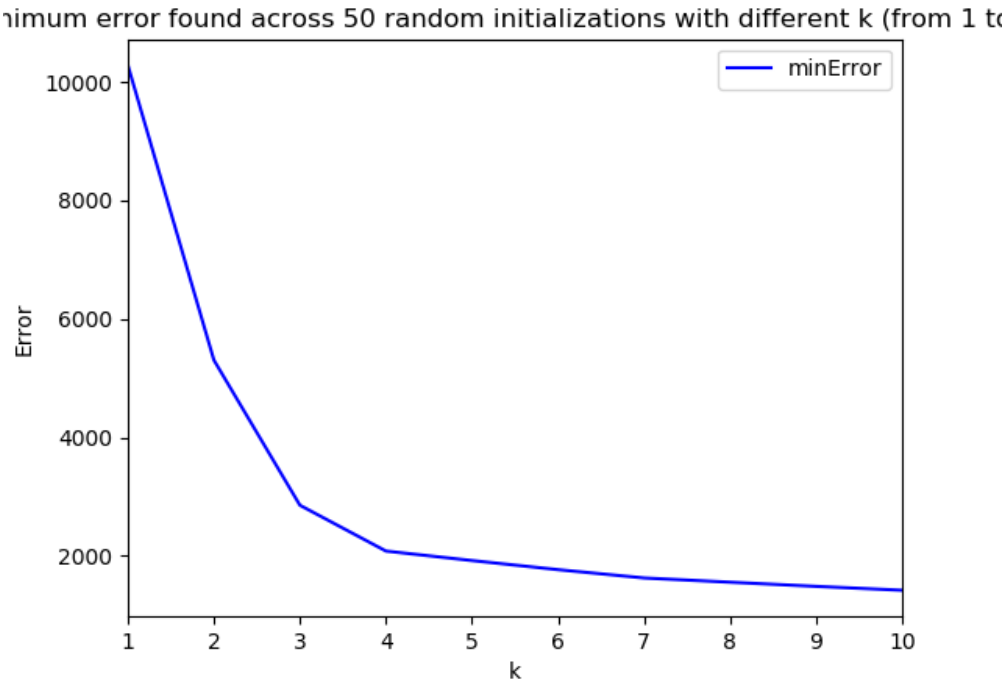


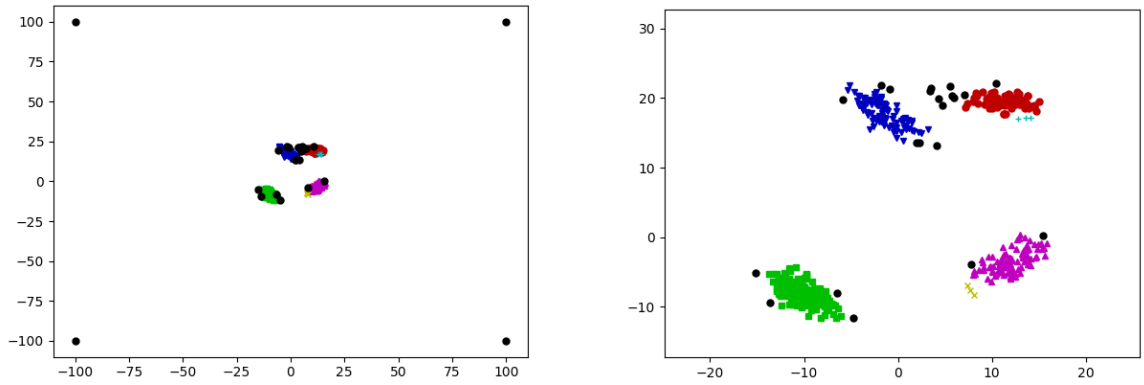
Figure 7: clustering figure 3.3 kmedians error

The reasonable k is 2, 3 or 4. The error is much smaller than the kmeans thus I am satisfied with the result.

3.4 Density-Based Clustering

Rubric: {reasoning:2}

If you run `python main.py -q 3.4`, it will apply the basic density-based clustering algorithm to the dataset from the previous part. The final output should look somewhat like this:



(The right plot is zoomed in to show the non-outlier part of the data.) Even though we know that each object was generated from one of four clusters (and we have 4 outliers), the algorithm finds 6 clusters and does not assign some of the original non-outlier objects to any cluster. However, the clusters will change if we change the parameters of the algorithm. Find and report values for the two parameters, `eps` (which we called the "radius" in class) and `minPts`, such that the density-based clustering method finds:

1. The 4 "true" clusters.
2. 3 clusters (merging the top two, which also seems like a reasonable interpretaition).
3. 2 clusters.
4. 1 cluster (consisting of the non-outlier points).

Answer: For clusterData2, we keep the minPts=3 and change eps.

1. eps=3
2. eps=9
3. eps=15
4. eps=20

3.5 Very-Short Answer Questions

Rubric: {reasoning:3}

1. Does the standard k-means clustering algorithm always yield the optimal clustering solution for a given k?
   Answer: No. For different initializations, it may not always converge to the optimal solution but a sub-optimal one.
2. If your set out to minimize the distance between each point and its mean in a k-means clustering, what value of k minimizes this cost? Is this value useful?
   Answer: When k=n, the distance is minimized to 0. But it is not useful.
3. Describe a dataset with k clusters where k-means would not be able to find the true clusters.
   Answer: Consider a condition with 2 clusters and there is a line between two points in one cluster going through the other cluster.

4. Suppose that you had only two features and that they have very-different scales (like kilograms vs. milligrams). How would this affect the result of density-based clustering?
   Answer: If the scales of features are really different, the expand of cluster may always add the one with smaller scale first as they will be much more possible to be in the neighbour cluster. This actually ignore the feature of smaller scale. Thus features should be of similar scales or should be normalized to similar scales. Actually, it is even difficult to choose proper minPts and eps in this condition.

5. Name a key advantage and drawback of using a supervised outlier detection method rather than an unsupervised method?
   Answer: For supervised outlier detection, if the outliers are similar to normal points, it can perform very well with high accuracy. However, if the outliers and the normal points are different, supervised method will not work and we have to use unsupervised method.

4 Vector Quantization

Rubric: {reasoning:6}

Discovering object groups is one motivation for clustering. Another motivation is *vector quantization*, where we find a prototype point for each cluster and replace points in the cluster by their prototype. If our inputs are images, vector quantization gives us a rudimentary image compression algorithm.

Your task is to implement image quantization in *quantize_image.py* with `quantize` and `dequantize` functions. The `quantize` function should take in an image and, using the pixels as examples and the 3 colour channels as features, run k-means clustering on the data with 2^b clusters for some hyperparameter b. The code should store the cluster means and return the cluster assignments. The `dequantize` function should return a version of the image (the same size as the original) where each pixel's orignal colour is replaced with the nearest prototype colour.

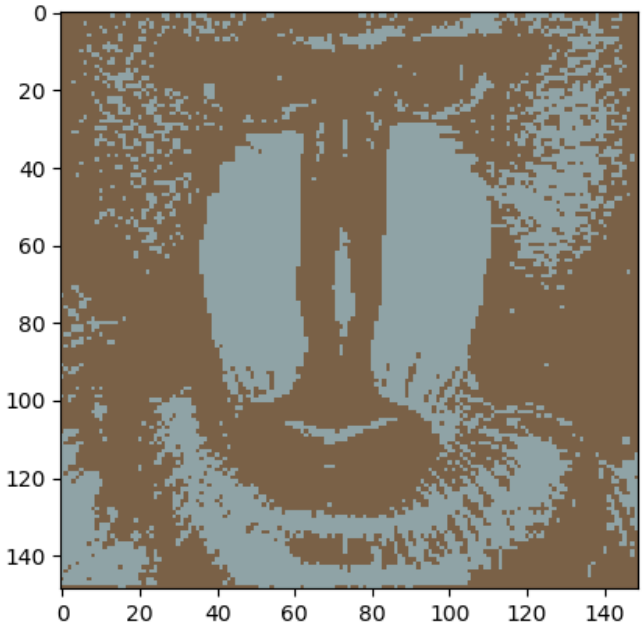
To understand why this is compression, consider the original image space. Say the image can take on the values $0, 1, \ldots, 254, 255$ in each colour channel. Since $2^8 = 256$ this means we need 8 bits to represent each colour channel, for a total of 24 bits per pixel. Using our method, we are restricting each pixel to only take on one of 2^b colour values. In other words, we are compressing each pixel from a 24-bit colour representation to a b-bit colour representation by picking the 2^b prototype colours that are "most representative" given the content of the image. So, for example, if $b = 6$ then we have 4x compression.

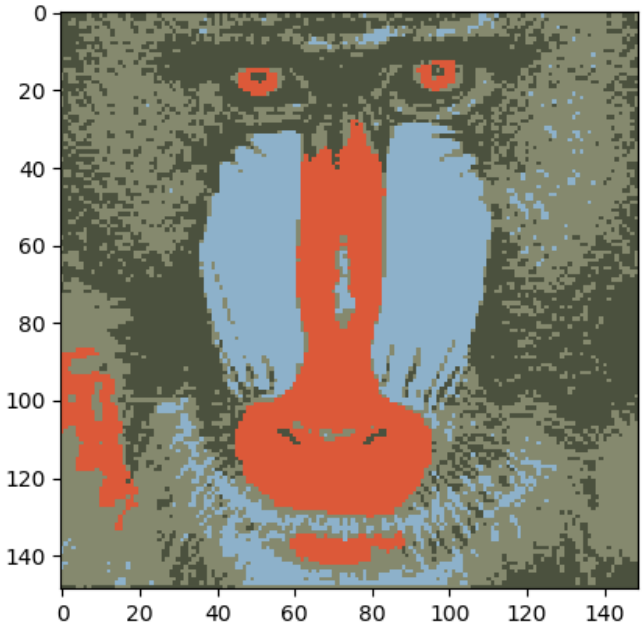
The loaded image contains a 3D-array representing the RGB values of a picture. Implement the *quantize* and *dequantize* functions and show the image obtained if you encode the colours using 1, 2, 4, and 6 bits with the provided image. You are welcome to use the provided implementation of k-means or the scikit-learn version.

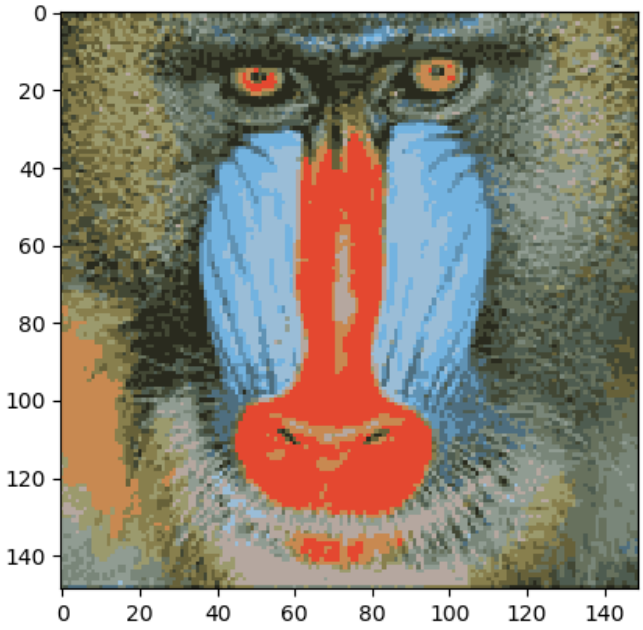
1. Hand in your *quantizeImage* and *deQuantizeImage* functions.
2. Show the image obtained if you encode the colours using 1, 2, 4, and 6 bits per pixel (instead of the original 24-bits).
3. Briefly comment on the prototype colours learned in case each, which are saved by the code.

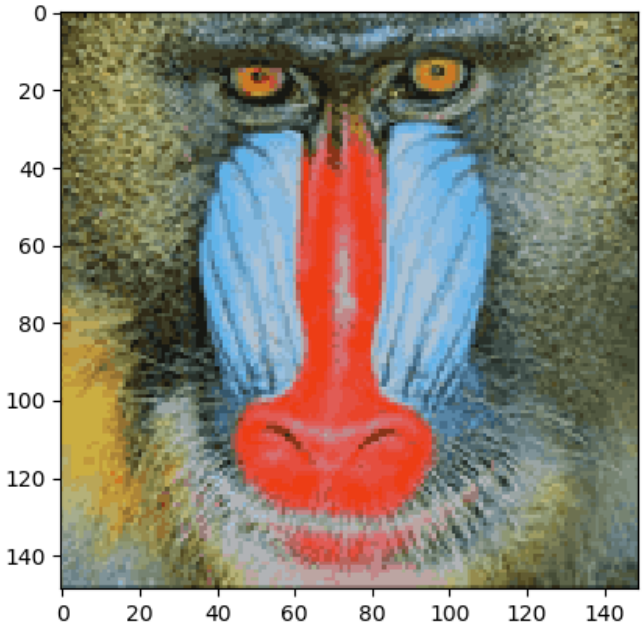
Answer:

1. The URL of the code is: `https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/g5e0b_u7p1b_a2/blob/master/code/quantize_image.py`.
2. The images obtained are shown below.

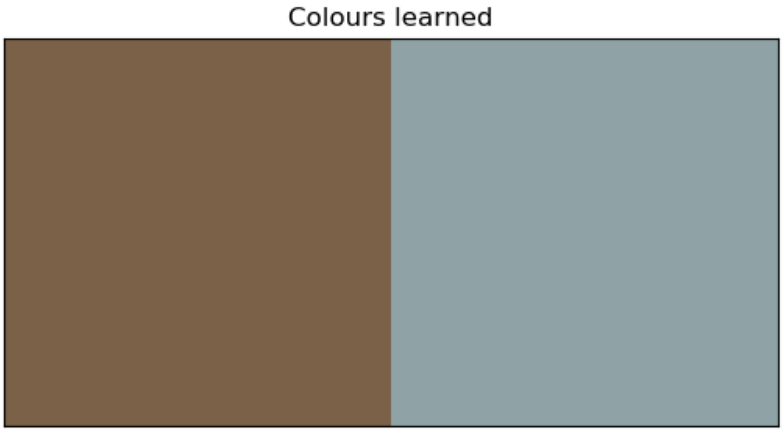

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

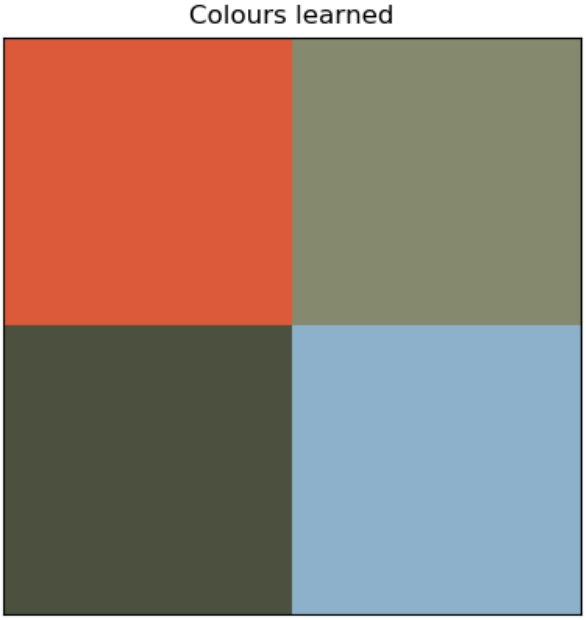

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

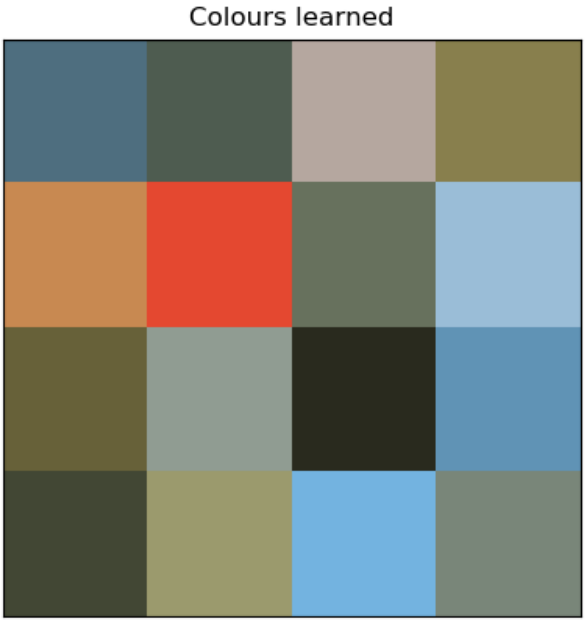


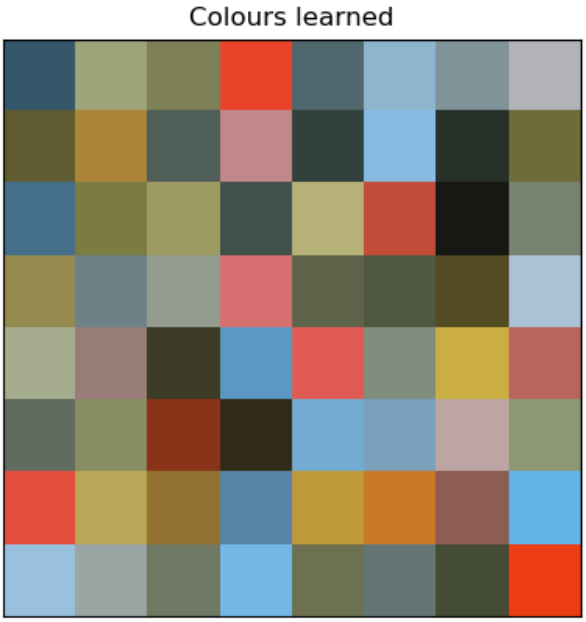


3. The prototype of the colours are shown below.


Colours learned


Colours learned





From these figures, we can conclude that with b (or the number of clusters) increasing, the details of image are better displayed and the image looks smooth and more similar to the original image as more and more pixels are colored with better means and more kinds of colors are used as means. Based on the principle, it is obvious that the clusters of colors always in the region which could be the 'center'

of this color considering its distribution of the whole image.