CPSC 340 Assignment 2 Report

1 Naive Bayes

1.1 Naive Bayes by Hand

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

- $p(y = 1) = 0.6$.
- $p(y = 0) = 0.4$.

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

- $p(x_1 = 1|y = 1) = 0.5$.
- $p(x_2 = 0|y = 1) = 0.33$.
- $p(x_1 = 1|y = 0) = 1$.
- $p(x_2 = 0|y = 0) = 0.75$.

(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.)

$$\begin{aligned} p(y = 1|x_1 = 1, x_2 = 0) &= p(x_1 = 1|y = 1) * p(x_2 = 0|y = 1) * p(y = 1) \\ &= 0.5 * 0.33 * 0.6 \\ &= 0.1 \end{aligned}$$

$$\begin{aligned} p(y = 0|x_1 = 1, x_2 = 0) &= p(x_1 = 1|y = 0) * p(x_2 = 0|y = 0) * p(y = 0) \\ &= 1 * 0.75 * 0.4 \\ &= 0.3 \end{aligned}$$

As $0.3 > 0.1$, the most likely label for the test example is 0.

1.2 Bag of Words

1. Which word corresponds to column 50 of X?

   lunar

2. Which words are present in training example 500?

   car, fact, gun, video

3. Which newsgroup name does training example 500 come from?

   talk.*

1.3 Naive Bayes Implementation

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.

The code can be found at *code/naive_bayes.py*.

The random forest: 0.203
The validation error obtained (before implementing Laplace smoothing): 0.188
The sklearn validation error: 0.187

Our validation error is quite close to the sklearn validation error, meaning our implementation was mostly correct.

1.4 Laplace smoothing

1. Modify your code so that it uses Laplace smoothing, with β as a parameter taken in by the constructor.

   Done, the code can be found at *code/naive_bayes.py*.

2. Did you need to modify your fit function, predict function, or both?

   Only the fit function needed to be modified.

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?

   It uses 1 as the default value for Laplace smoothing.

   The validation error obtained (after using a value of 1 for Laplace smoothing): 0.187. We do indeed get the same results.

1.5 Runtime of Naive Bayes for Discrete Data

What is the cost of classifying t test examples with the model?

The cost of classifying t examples would be $O(tkd)$.

Assume finding the probability of k is $O(1)$ as it should be found in the training phase.

Assume finding the conditional probability of $d = c$ is $O(1)$ as it should be found in the training phase, and stored in a array where the value is accessible with d, k and c.

The explanation is as follows:

For each testing object, we must find the greatest conditional probability of each k given all d features, where some d_n equals some c, so we may classify it as that k. In order to do that, we first find the probability of k.

Then, for each feature, we find the conditional probability of $d = c$ given k, and multiply it by our probability of k. This gives us the conditional probability of our current k given all d features. If it is greater than the current greatest conditional probability, we replace it and its label.

Finally, we can classify the testing object as the label that has the current greatest conditional probability.

Looping through all the features and finding the conditional probability for them to multiply all together is $O(d)$. This is done for each k, which is $O(kd)$, to find what we should label t. Finally, this is done for each t, which is $O(tkd)$, as we must label every t.

2 Random Forests

2.1 Implementation

1. Why doesn't the random tree model have a training error of 0?

   The random tree model doesnt have a training error of 0 because the tree is not very deep. So the decision tree hasnt got specific to the point of overfitting necessarily.

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.

   code provided in: *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.

   Using 50 trees, and a depth of infinity: the training error of the Random Forest is 0, and the test error is 0.18 (lower than both Decision and Random decision trees).

   Yes, this is the result that we expected because Random forests prevent overfitting by overfitting individually but overall making independent errors.

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.

   sklearns implementation is much faster, and the average of their Random forest test error is much lower (.16) - more accurate.

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?

   It can become 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?

B, C and F.

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?

   Translate the raw audio segments of training data and use that.

3 Clustering

3.1 Selecting among k-means Initializations

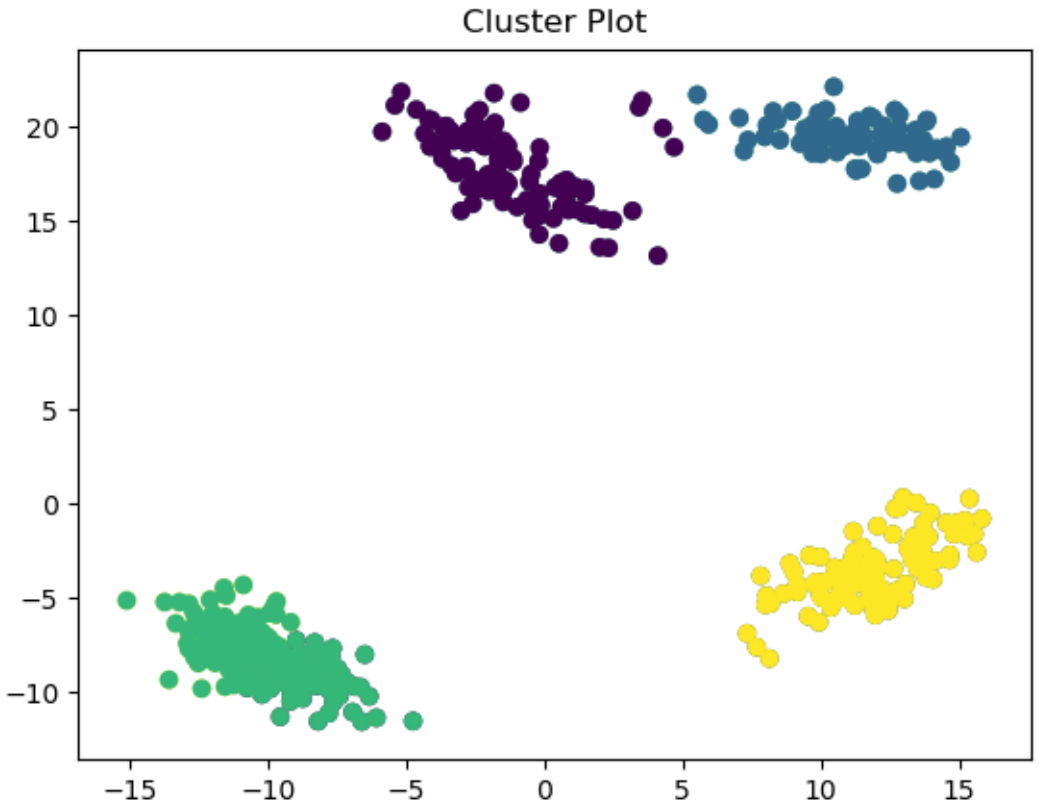
1. In the *kmeans.py* file, add a new function called *error* that takes the same input as the *predict* function but that returns the value of this above objective function. Hand in your code.

   Done, the code can be found in *code/kmeans.py*.

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

   Our error gets less and less with each iteration of the algorithm. The difference between iterations gets smaller and smaller, approaching the error found when no clusters change.

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.



4. Looking at the hyperparameters of scikit-learn's `KMeans`, explain the first four (`n_clusters`, `init`, `n_init`, `max_iter`) very briefly.

   n_clusters indicates the number of clusters that we want to classify in the end.
   init specifies how we want to choose our centroids.
   n_init specifies the number of times we want to initialize.
   max_iter specifies the max number of times we can iterate the k_mean algorithm for.

3.2 Selecting k in k-means

1. Explain why the *error* function should not be used to choose k.

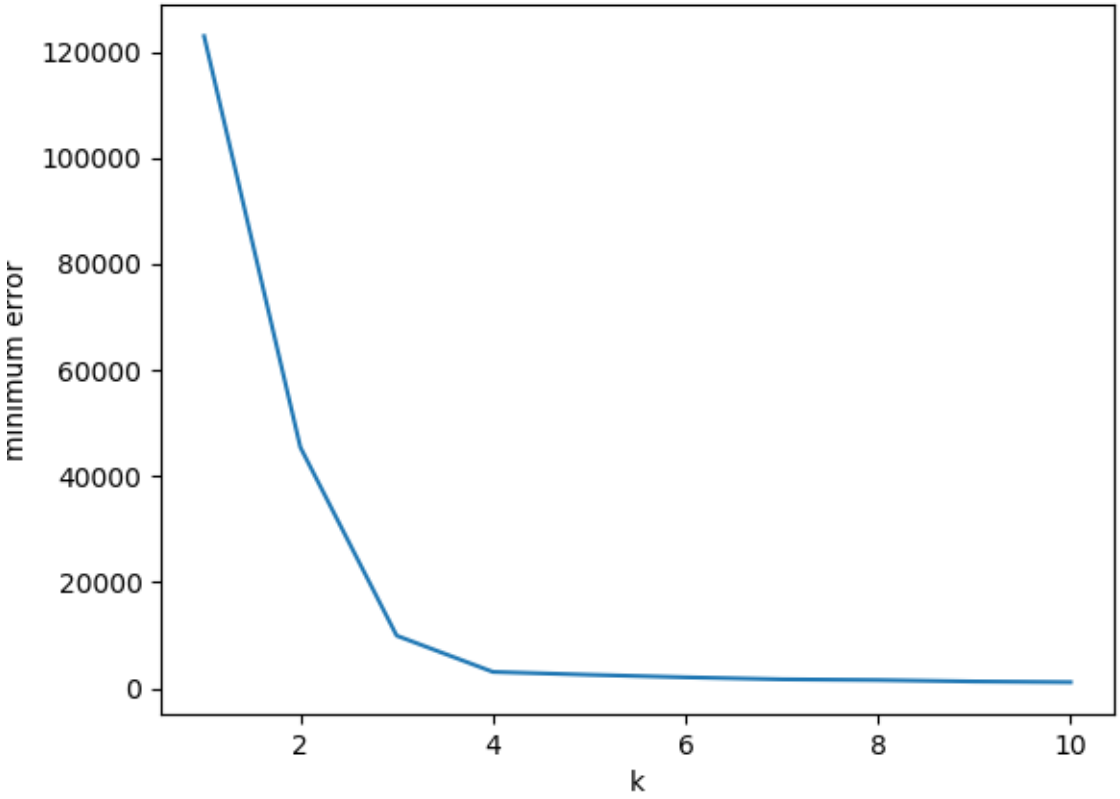
   It doesn't make sense to use error to pick k, because our most optimal value would be k = n, because then each point would be its own cluster, giving us an error of 0. But then at that point we wouldn't even be clustering.

2. Explain why even evaluating the *error* function on test data still wouldn't be a suitable approach to choosing k.

   Same thing as above. Even if we use different data, we would still get the smallest error if we set

k = n. Again, we wouldn't even be clustering.

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.



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.

   Reasonable values would include 2, 3 or 4, although 2 and 3 makes the most noticeable difference in slope.

3.3 k-medians

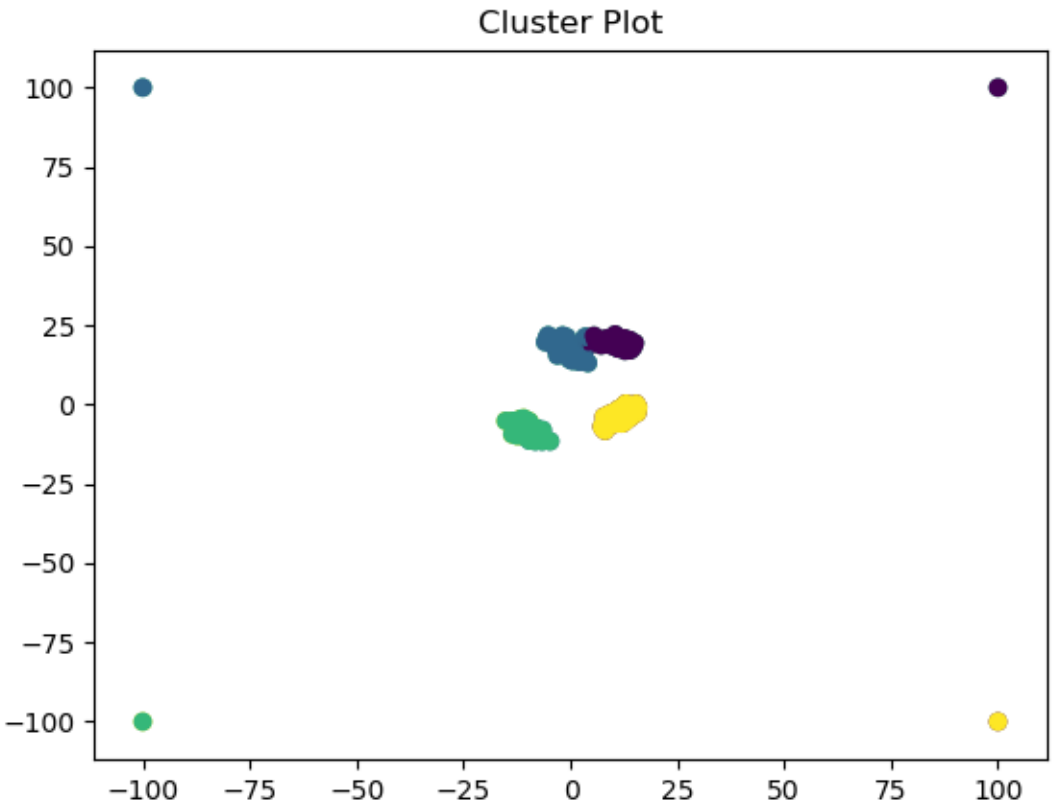
1. Using the `plot_2dclustering` function, output the clustering obtained by running k-means 50 times (with $k = 4$) and taking the one with the lowest error. Are you satisfied with the result?

   It seems to have done a fine job classifying the clusters. However, the error appears to be extremely high.

2. What values of k might be chosen by the elbow method for this dataset?

   You would pick 2 or 3 as they appear to give the sharpest slope. 4 doesn't seem to give much of a slope.

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 $k = 4$.



4. Using the L1-norm version of the error (where y_i now represents the closest median in the L1-norm), what value of k would be chosen by the elbow method under this strategy? Are you satisfied with this result?

   2, 3 or 4 would be reasonable values to pick. The difference in slope seems to be much greater as opposed to the graph created with kmeans.

3.4 Density-Based Clustering

1. The 4 "true" clusters.

eps = 3, min_samples = 3

2. 3 clusters (merging the top two, which also seems like a reasonable interpretaition).

   eps = 6, min_samples = 6

3. 2 clusters.

   eps = 15, min_samples = 2

4. 1 cluster (consisting of the non-outlier points).

   eps = 30, min_samples = 2

3.5 Very-Short Answer Questions

1. Does the standard k-means clustering algorithm always yield the optimal clustering solution for a given k?

   No, it doesn't. At times, we can get weird initializations, which will lead the means to sometimes converge to weird locations and classify weirdly. For example, if there are two clusters that are somewhat together, in some initializations, we might correctly label the two as individual clusters, in other initializations, there might be a mean smack dab in the two, causing it to be labelled as one big cluster.

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?

   k = n, where n is equal to the number of points. This value isn't useful at all, as we are no longer identifying clusters, but rather each individual point.

3. Describe a dataset with k clusters where k-means would not be able to find the true clusters.

   Imagine that we have two clusters. One that looks like a horseshoe, and the other that looks like an upside down horseshoe. They would then be positioned such that the two would almost be "linking" together. Here, k-means would not be able identify that there are two clusters. If we set k = 2, the model would guess that the "intersection" point between the two horseshoes is a cluster in itself. K-means is not able to accurately identify a pattern like this where the real cluster appears like a horseshoe - rather, it can only identify circular, blob-like clusters.

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?

   As k-means simply clusters points based on their difference in distance - each features "idea" of distance should be similar, otherwise you are going to get very skewed results that are not based on the actual distance of the points. For example, a scale of miligrams might treat a difference of 1 gram as huge - whereas a scale of kilograms might treat a difference of 1 gram as very small. Hence, you might get clusters that are entirely dependent on the scale treating the difference as very small, or clusters that are not represented due to the scale treating the difference as very big.

5. Name a key advantage and drawback of using a supervised outlier detection method rather than an unsupervised method?

   A disadvantage: experience overfitting on the training dataset - only know how to identify outliers

for a specific training dataset (the point you've seen) - can't really detect any outliers on any new data.

An advantage: unsupervised learning can only detect outliers through a difference of "distance" in the features - it isn't actually able to label points meaningfully - you might have outliers that have features similar to other data. For example - if on multiple days you had similar amounts of egg and weight, and didn't get sick - and then one day, you had the same amount as you approximately had before, and you got sick. That would be an outlier, but it wouldn't be able to be detected by unsupervised learning as it only looks for cluster.

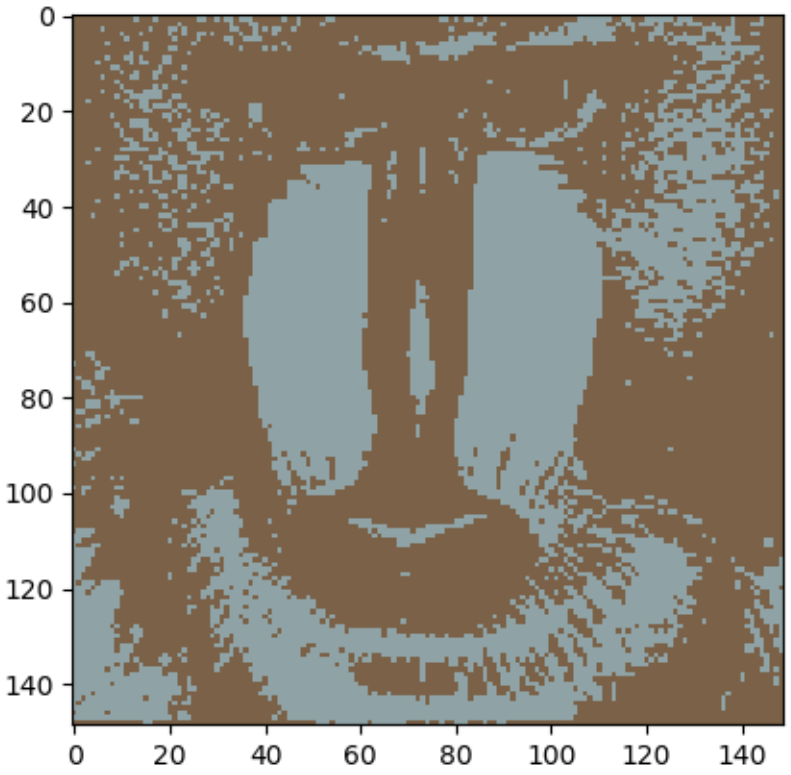
4 Vector Quantization

1. Hand in your *quantizeImage* and *deQuantizeImage* functions.

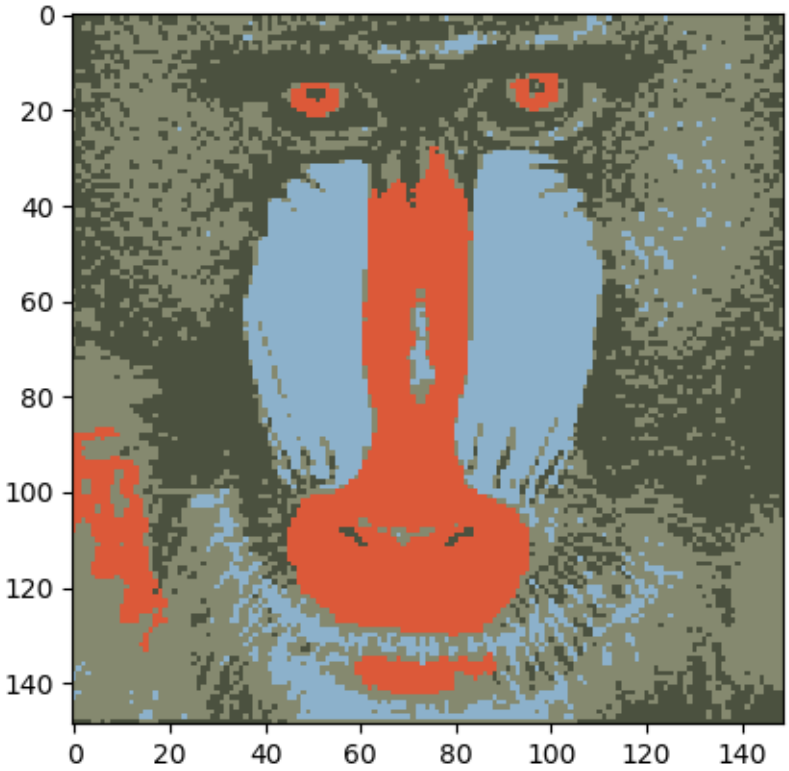
   Code provided in: *quantize_image.py*

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).

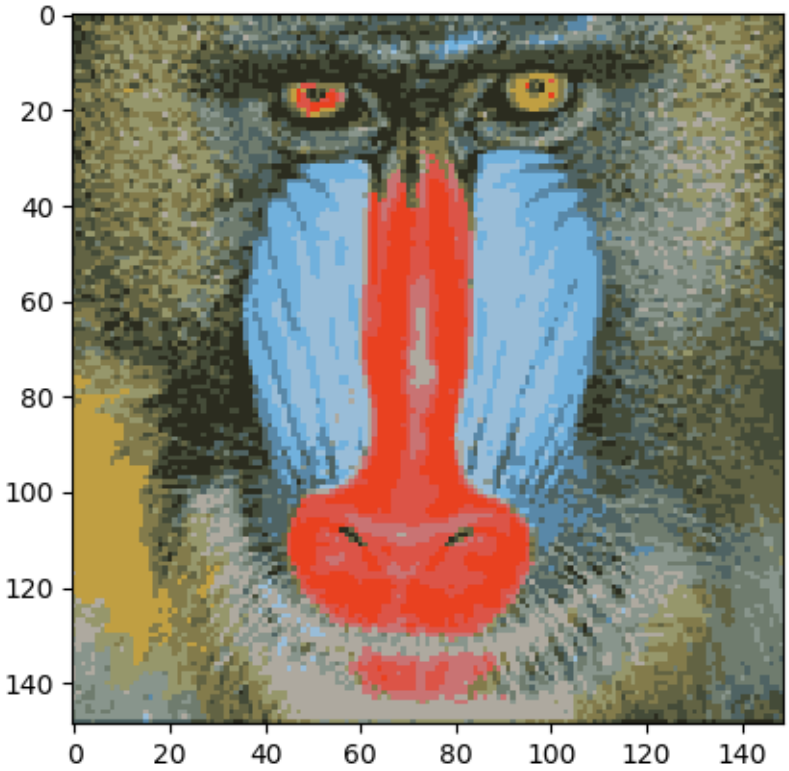
   Images provided below: can be found in the Figures folder.



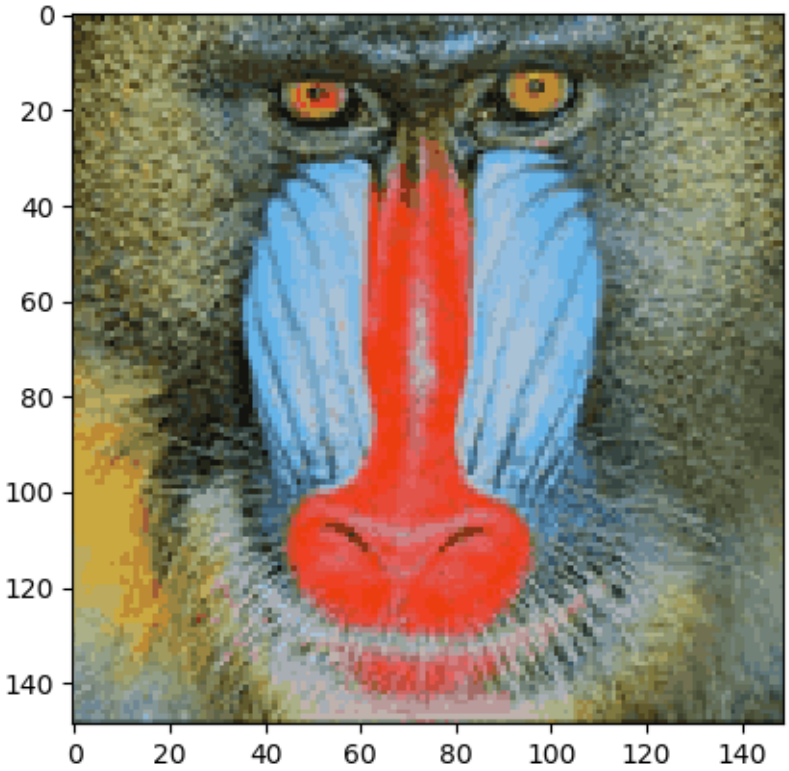
1 bits per pixel



2 bits per pixel



4 bits per pixel



6 bits per pixel

3. Briefly comment on the prototype colours learned in case each, which are saved by the code.

The dequantize function assigns each pixel to the nearest prototype colour (a cluster mean). This is a compression, meaning that we are using less bits to represent a pixel. The prototype colours that the program picks out are the colour channels that is restricted to, that is, the most representative colours of the image, given the amount of colours it has to work with (b). It then builds the image based on the nearest "cluster", or prototype color, for each pixel.