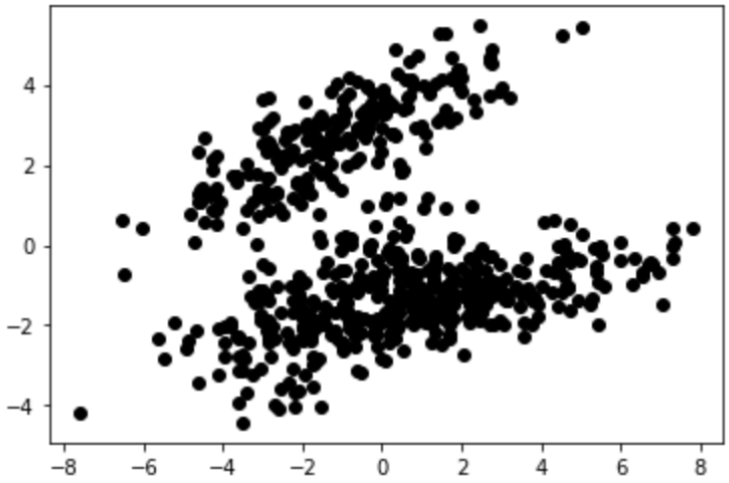
Problem 4 Solution: Yeast Colonies

By Henry Tang

This problem was a nice opportunity to combine unsupervised and supervised learning in a slightly unconventional way. Essentially, the problem is asking us to assign each data point in Sally's dataset into one of three distinct categories/labels. Then, using the information we learned from Sally's dataset, we will make predictions on Charlotte's data (which we do not have access to beforehand) that are hopefully quite good. (As we'll see in practice, they are not bad!)

1 Unsupervised Clustering (10 pts)

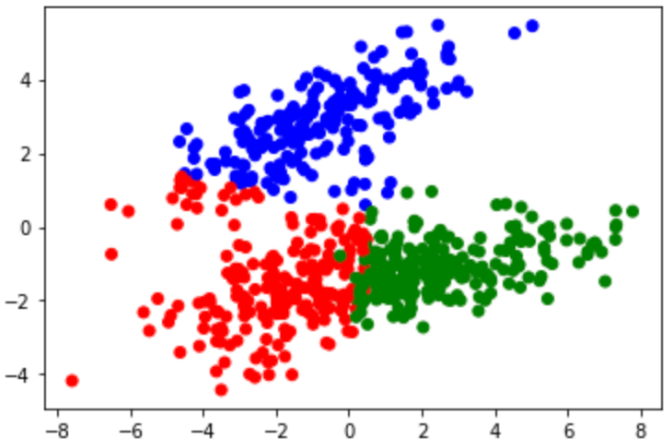
One good thing we can do whenever we encounter a problem like this is to visualize the data points that we are given. The problem is that the datapoints are in $\mathbb{R}^5$, and it is unfortunately quite difficult to visualize 5-dimensional space. One way of dealing with this is using Principal Component Analysis (PCA). Essentially, PCA is a method of changing the bases of a set of datapoints so that each axis is uncorrelated with the other ones. In 2D, think of changing from x, y to another set of axes (along which two "orthogonal" directions of variation occur). In our case, for visualization purposes, after PCA we project the points down to the first two axes (the most significant axes). This yields the diagram below:



Applying k-means. We can see that there certainly appears to be at least two (and perhaps more) distinct clusters. Using the suggestion in the problem, we can perform k-means and see what we get. A few things to keep in mind when running k-means:

- Make sure you're running the algorithm for enough epochs so that the centroids no longer change much each iteration. In other words, make sure your k-means procedure converges!
- Run the k-means algorithm multiple times, and choose the centroids from the run with the minimum total distance between each point and its corresponding centroid. Any particular k-means can be highly dependent on the initialization of its centroids, which is usually done randomly. Running multiple times can reduce this variance.

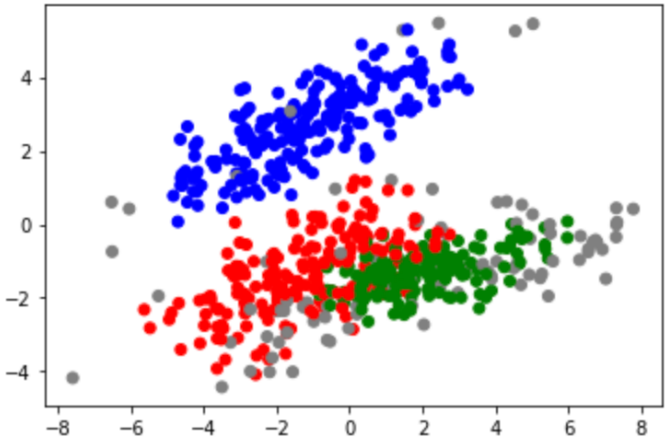
Afterwards, we can view how k-means has clustered our datapoints. It could look something like this:



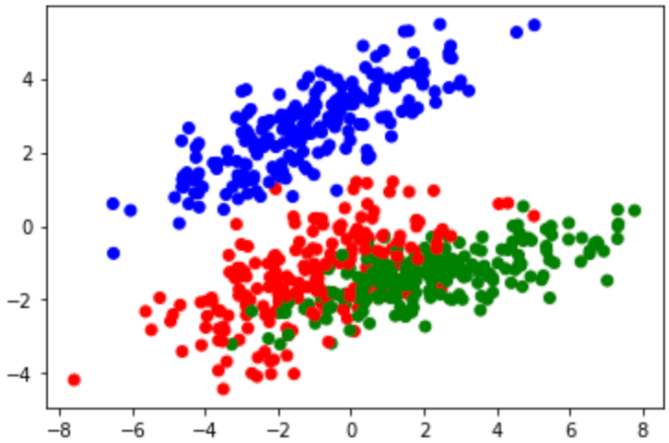
We can see that k-means has done a reasonably good job. In fact, it achieves around 84% accuracy. However, some of the boundaries certainly seem a little sharp, and don't get much better even if we try more iterations. If you want to do better, you'll have to look towards more advanced models.

Density-Based Clustering. Density-based Spatial Clustering of Applications with Noise (DBSCAN) is a model that groups points that are close together into a single region. It is quite powerful because it is able to learn many arbitrarily shaped clusters. In addition, this algorithm does not require the number of clusters as an input (unlike k-means). It's important to note that DBSCAN is highly dependent on many of its input parameters. Thus, when we run this algorithm, we should be checking to see if the points have

mostly been clustered into three distinct groups (without too many outliers and points not in the three main groups). If this is the case, then we can be reasonably confident we've found a good configuration of points. Using DBSCAN, our labelling looks like the following (where gray points are outliers/not classified in the three main groups):



Gaussian Mixture Model. Finally, inspired by how the data points seem to be normally (Gaussian) distributed, we can try using a *Gaussian Mixture Model*. Without getting in the weeds, the model tries to assign Gaussian distributions that maximize the probability a certain set of parameters actually generates the given data. This can be done using the Expectation–Maximization (EM) algorithm. Result:



In this problem, we weren't very harsh on the grading: using either a DBSCAN (with well chosen parameters) or a GMM would have netted you full marks. Note that their actual performance does differ: a DBSCAN would have given you an accuracy of around 90%, while a GMM would give you an incredibly high score of 99.7%! This is because the dataset itself was actually artificially generated by sampling points from three random Gaussians. (And some smart cookies found out...)

The official solution code (in a Jupyter Notebook), along with some extra analysis, for this part can be found here. If you're new to this stuff, we also encourage you to use the following links to learn more about the models and their implementations described in this solution: DBSCAN, GMM, and EM.

2 Further Label Prediction (5 points)

The first possible approach to this part is to literally run the same algorithm you used to predict labels for the training set, except on the test set. This will work, albeit with a slightly worse accuracy, since the test set has less data than the training set and thus the model won't generalize as well.

Instead, a better idea is to use parameters you've learned from part 1 to predict results in part 2. For instance, a common tactic was to use whatever unsupervised model that was used for in part 1 to *predict which cluster each new data point from part 2 belonged to*. For instance, in the case of k-means clustering, you would reuse the centroids from Part 1. And when given a new data point to classify, you would choose the classification corresponding to that of the nearest centroid. This would've gotten you an accuracy of around 84%; reusing the learned GMM model would've resulted in an accuracy of around 99+%.

The truth is you can do even better. For instance, you can train a classification model on your predicted labels from Part 1 in a supervised manner, then use this model to predict labels in Part 2. The intuitive reason this can result in even better predictions on the test set is that we can use these more advanced classification models to learn about certain patterns in the underlying data that aren't captured by the relatively simpler unsupervised models. For instance, one powerful type of classification model (especially when there isn't huge amounts of data) is a Support Vector Machine (SVM). If we train a SVM model on our training data using the k-means predicted labels and then use it on the test data, we can achieve an improvement of about 1.5%, yielding 85% accuracy. If we train it on the GMM predicted labels, then we can achieve 100% accuracy on the test set!

The official solution code for this part can be found here. Review it and ask us any questions!

Dataset and Verifying our Grading. The true labels of the dataset in part 1 can be found under the `TrainData` subdirectory of the folder for this problem in a file titled `YeastTrainLabels.csv`. The dataset you were tested on in part 2 (and their corresponding labels) can be found in the `TestData` subdirectory. Feel free to take a look for both parts and see if the score you were assigned matches up with what you'd expect. You can also access all the relevant files on Google Drive here.

Plaudits:

- Congratulations to Devdigvijay Singh (MAE '24)/Alan Ji (ORF '24)/Sean Wang (COS '24) and Rahul Saha (COS '22)/Alan Chung (MAT '22)/Kevin Feng (MAT '22) for achieving the highest accuracy on *both parts*! They both achieved an impressive 99.69% accuracy on the first part and a 99.64% accuracy on the second part.
- In addition, congratulations to Kiril Bangachev (MAT '22)/Aleksa Milojevic (MAT '22)/Alex Lopex (MAT '22) and Louis Viglietta (CBE '24) for achieving full scores on both parts!