In [1]:

```
%matplotlib inline
```

SYDE 522 Assignment 2

Clustering and SVMs

Due: Friday Oct 20 at 11:59pm

As with all the assignments in this course, this assignment is structured as a Jupyter Notebook and uses Python. If you do not have Python and Jupyter Notebook installed, the easiest method is to download and install Anaconda https://www.anaconda.com/download (https://www.anaconda.com/download). There is a quick tutorial for running Jupyter Notebook from within Anacoda at https://docs.anaconda.com/free/anaconda/getting-started/hello-world/#python-exercise-jupyter (https://docs.anaconda.com/free/anaconda/getting-started/hello-world/#python-exercise-jupyter) under "Run Python in a Jupyter Notebook"

Implement your assignment directly in the Jupyter notebook and submit your resulting Jupyter Notebook file using Learn.

While you are encouraged to talk about the assignment with your classmates, you must write and submit your own assignment. Directly copying someone else's assignment and changing a few small things here and there does not count as writing your own assignment.

Make sure to label the axes on all of your graphs.

Question 1

a) [2 marks] The following code generates the same data that was used to demonstrate K-means clustering in class. Note that since this question is about clustering, which is an unsupervised technique, we will not be using the `y` variable and will instead just use `x`, which will have 500 data points, each of which is two-dimensional.

```
import sklearn.datasets
x, y = sklearn.datasets.make_blobs(
    n_samples=500, cluster_std=[1,2,0.5], random_state=8
)
```

Implement K-means clustering on this data, with k=3. Run enough iterations for it to converge to a stable classification (probably around 4-5 iterations). Generate a scatterplot that shows each learned category in a different colour. For example, if you have an array `output` which contains the learned category for each item and those values were `0`, `1`, and `2`, you could generate the plot with the following code:

```
import matplotlib.pyplot as plt
import numpy as np
plt.figure(figsize=(6,6))
plt.scatter(x[output==0,0], x[output==0,1], label='category 0')
plt.scatter(x[output==1,0], x[output==1,1], label='category 1')
plt.scatter(x[output==2,0], x[output==2,1], label='category 2')
plt.legend()
plt.xlabel('$x_1$')
```

b) [1 mark] For the same model you ran in part a), compute the "Loss" as the model is learning. The Loss is defined as $\sqrt{\frac{1}{N} \sum_k \sum_i (x_i - c_k)^2}$, where N is the number of items (500), the sum over k is over the 3 categories, the sum over i is over the items in the current category, x_i is the item itself, and c_k is the prototype item for that category. Compute the Loss after zero iterations (i.e. for the initial randomly chosen prototypes), and then after each iteration of the k-means clustering algorithm. Generate a plot with the number of iterations on the x-axis and the Loss on the y-axis.

c) [1 mark] Perform k-means clustering on the same data, but for values of k between 1 and 14, inclusive. For each k-value perform enough iterations for the clustering to be stable. After it is stable, compute the Loss. Generate a plot with the value of k on the x-axis, and the Loss on the y-axis.

Given this plot, what is a good value for k? Why?

Question 2:

a) [1 mark] The pre-written implementation of k-means clustering can be used with the following commands:

```
import sklearn.cluster
kmeans = sklearn.cluster.KMeans(n_clusters=3)
kmeans.fit(x)
output = kmeans.predict(x)
```

Use this implementation of k-means clustering to repeat question 1a) and generate the same plot.

b) [1 mark] The digit image dataset that was used in assignment 1 can be loaded with the following commands:

```
import sklearn.datasets
digits = sklearn.datasets.load_digits()
x = digits.data
```

Use the `sklearn` implementation of k-means clustering to form 10 categories from this data. Plot the results, using the following code which will show the first 12 items in each of the 10 categories.

```
plt.figure(figsize=(12,12))
for i in range(10):
    indices = np.where(output==i)[0]
    for j,index in enumerate(indices[:12]):
        plt.subplot(10,12,i*12+j+1)
        plt.imshow(digits.data[index].reshape(8,8), cmap='gray_r')
        plt.xticks([])
        plt.yticks([])
        if j==0:
            plt.ylabel(f'category {i}')
plt.show()
```

How does this clustering compare to the natural clustering into the 10 digits that a person might apply to this same data? What similarities and differences do you see? (Note: you don't need a quantitative answer to this question; I'm looking for more qualitative answers).

c) [1 mark] The following code will perform Hierarchical Clustering (also known as Agglomerative Clustering). Apply this to the digits data from the previous question and generate the same plot.

```
agglom = sklearn.cluster.AgglomerativeClustering(n_clusters=10)
agglom.fit(digits.data)
output = agglom.labels_
```

How does this clustering compare to the natural clustering into 10 digits and to the clustering in part b)? What similarities and differences do you see?

Question 3:

a) [2 marks] The following code generates the data used to demonstrate the SVM in class (notice that the two categories are now `1` and `-1` rather than `1` and `0`).

```
x, y = sklearn.datasets.make_blobs(centers=[[-2, -2], [2, 2]],
                                   cluster_std=[0.3, 1.5],
                                   random_state=0,
                                   n_samples=200,
                                   n_features=2)
y[y==0] = -1
```

Implement the version of the SVM with a learning rule. This is the version where we modify the Perceptron learning rule to produce a new learning rule that will try to minimize ω while trying to keep one category with $\omega \cdot x + b > 1$ and the other with $\omega \cdot x + b < -1$. Apply it to the data generated above. Use a learning rate of 0.01 and a λ value of 0.001. Initialize it with $\omega = [0, 0]$ and $b = 0$. Perform 200 iterations of the learning rule through all the data points (so the learning rule will be applied a total of 200 x 200 = 40000 times). This should be enough for it to stablize to the optimal decision boundary.

Generate a scatterplot showing the results. This should show the data points, coloured differently for the two categories. Also report the final ω and b values. In addition, plot the decision boundary line $\omega \cdot x + b = 0$ along with the two other boundary lines $\omega \cdot x + b = 1$ and $\omega \cdot x + b = -1$. One way to generate those lines is to use the following code:

```
xx = np.array([-4, 8])
yy = (b-xx*w0) / w1
yy_upper = ((b+1)-xx*w0) / w1
yy_lower = ((b-1)-xx*w0) / w1
plt.plot(xx, yy)
plt.plot(xx, yy_upper)
plt.plot(xx, yy_lower)
plt.xlim(-3, 6)
```

b) [1 mark] For the SVM you implemented in part a), plot the magnitude of ω over the 200 interations (i.e. plot the initial value of $|\omega| = \sqrt{\omega_0^2 + \omega_1^2}$, the value after applying the learning rule to each data point once, the value after applying it to each data point twice, and so on up to 200 times). Does the resulting value for ω converge?

c) [1 mark] The following code generates the data set with two circles, one inside the other.

```
x, y = sklearn.datasets.make_circles(n_samples=100,
                                     shuffle=True,
                                     noise=0.1,
                                     random_state=0,
                                     factor=0.3)
```

Apply your implementation of the SVM to this data and plot the result after 100 iterations. Use the same parameters as before and generate the same plot as in part a).

d) [1 mark] Repeat part c) but augment the data so that there is an additional feature computed as $x_1^2 + x_2^2$. This should make it possible for the SVM to learn a boundary between the datasets in the new 3-dimensional space.

Plot the resulting classification. You just need to plot the data points, coloured by which category they are in. You do not need to plot the decision boundaries.

Question 4:

a) [1 mark] The `sklearn` implementation of a Linear SVM (i.e. one with using the Kernel Trick) can be used as follows:

```
svm = sklearn.svm.LinearSVC(C=1)
svm.fit(x, y)
output = svm.predict(x)
value = svm.decision_function(x)
```

(note that we use `predict` if we want the output to indicate a category label, but we use `decision_function` to get the value of $\omega \cdot x + b$)

Use this implementation to repeat question 3a. Use a cost `C=100`. Instead of plotting the decision boundary lines, use the following code to compute the output for a big grid of data points and plot the output as an image (as was done in class):

```
extent = (-3, 6, -3, 6)
G = 200
XX, YY = np.meshgrid(np.linspace(extent[2],extent[3],G), np.linspace(e
xtent[0],extent[1],G))
pts = np.vstack([YY.flatten(), XX.flatten()]).T
output_pts = svm.decision_function(pts)
im = plt.imshow(output_pts.reshape((G,G)).T, vmin=-1, vmax=1, cmap='Rd
Bu',
```

b) [1 mark] Repeat question 4a) using the nested circles data set from 3c. Show the resulting plot. Now repeat this again with the same data, but using the `sklearn` implementation of an SVM with a Gaussian Radial Basis Function kernel, which can be accessed using `svm = sklearn.svm.SVC(kernel='rbf', gamma=1, C=1)`. Use $\gamma = 1$ and $C = 1$. Show the resulting plot.

c) [1 mark] Here is the code to generate the overlapping dataset discussed in class.

```
x, y = sklearn.datasets.make_blobs(centers=[[-1, -1], [1, 1]],
                                   cluster_std=[1, 1],
                                   random_state=0,
                                   n_samples=200,
                                   n_features=2)
```

Use the methodology from class to optimize γ and C. Use 20% of the data for testing. When doing cross-validation, split your training data into 80% training and 20% validation and repeat 40 times to take the average. Plot the cross-validation accuracy for different C and γ values. To generate this plot, you can use code like the following, assuming your validation accuracy scores are in a matrix `accuracy`:

```
Cs = np.logspace(-3, 5, 25)
gammas = np.logspace(-6, 3, 28)
XX, YY = np.meshgrid(np.arange(len(gammas)), np.arange(len(Cs)))
plt.contourf(XX, YY, accuracy, levels=50)
plt.colorbar()
CS = plt.contour(XX, YY, accuracy, levels=[0,0.75,0.8,0.85, 0.9, 0.9
5], colors='k')
plt.clabel(CS, CS.levels, inline=True, fontsize=8)
plt.xticks(np.arange(len(gammas))[::3], gammas[::3], rotation=90)
plt.yticks(np.arange(len(Cs))[::3], Cs[::3])
```

Given your final choice of γ and C, re-train on all the training data, generate the same plot as in parts a and b. Report the accuracy of the categorization on the test data.