M11-L1-P1

November 26, 2023

0.1 M10-L1 Problem 1

In this problem you will implement the K-Means algorithm from scratch, and use it to cluster two datasets: a "blob" shaped dataset with three classes, and a "moon" shaped dataset with two classes.

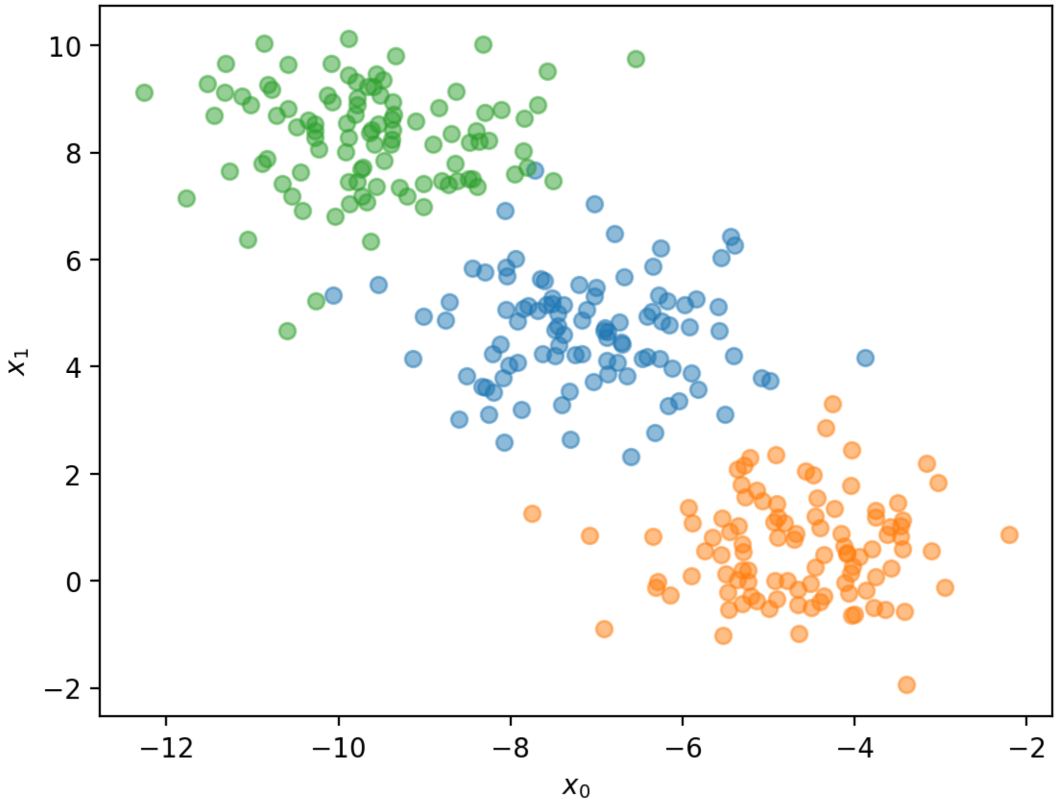
[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs, make_moons

## DO NOT MODIFY
def plotter(x, y, labels = None, centers = None):
    fig = plt.figure(dpi = 200)
    for i in range(len(np.unique(y))):
        if labels is not None:
            plt.scatter(x[labels == i, 0], x[labels == i, 1], alpha = 0.5)
        else:
            plt.scatter(x[y == i, 0], x[y == i, 1], alpha = 0.5)
    if labels is not None:
        if (labels != y).any():
            plt.scatter(x[labels != y, 0], x[labels != y, 1], s = 100, c =␣
↪'None', edgecolors = 'black', label = 'Misclassified Points')
    if centers is not None:
        plt.scatter(centers[:,0], centers[:,1], c = 'red', label = 'Cluster␣
↪Centers')
    plt.xlabel('$x_0$')
    plt.ylabel('$x_1$')
    if labels is not None or centers is not None:
        plt.legend()
    plt.show()
```

We will use `sklearn.datasets.make_blobs()` to generate the dataset. The `random_state = 12` argument is used to ensure all students have the same data.

[ ]:
```
## DO NOT MODIFY
x, y = make_blobs(n_samples = 300, n_features = 2, random_state = 12)
```

Visualize the data using the `plotter(x,y)` function. You do not need to pass the `labels` or `centers` arguments

[ ]:
```
plotter(x,y)
```



Now we will begin to create our own K-Means function.

First you will write a function `find_cluster(point, centers)` which returns the index of the cluster center closest to the given point. - `point` is a one dimensional numpy array containing the x_0 and x_1 coordinates of a single data point - `centers` is a 3×2 numpy array containing the coordinates of the three cluster centers at any given iteration - **return** the index of the closest cluster center

[ ]:
```
def find_cluster(point, centers):
    dist = np.sum(np.power(np.subtract(centers, point),2), axis=1)
    return np.where(dist == min(dist))[0][0]
```

Next, write a function `assign_labels(x, centers)` which will loop through all the points in `x` and use the `find_cluster()` function we just wrote to assign the label of the closest cluster center. Your function should return the labels - `x` is a 300×2 numpy array containing the coordinates of all the points in the dataset - `centers` is a 3×2 numpy array containing the coordinates of the three cluster centers at any given iteration - **return** a one dimensional numpy array of length 300 containing the corresponding label for each point in `x`

[ ]:
```python
def assign_labels(x, centers):
    labels = np.zeros(x.shape[0])
    for i,point in enumerate(x):
        labels[i] = find_cluster(point, centers)
    return labels
```

Next, write a function `update_centers(x, labels)` which will compute the new cluster centers using the centroid of each cluster, provided all the points in `x` and their corresponding labels - `x` is a 300×2 numpy array containing the coordinates of all the points in the dataset - `labels` is a one dimensional numpy array of length 300 containing the corresponding label for each point in `x` - **return** a 3×2 numpy array containing the coordinates of the three cluster centers

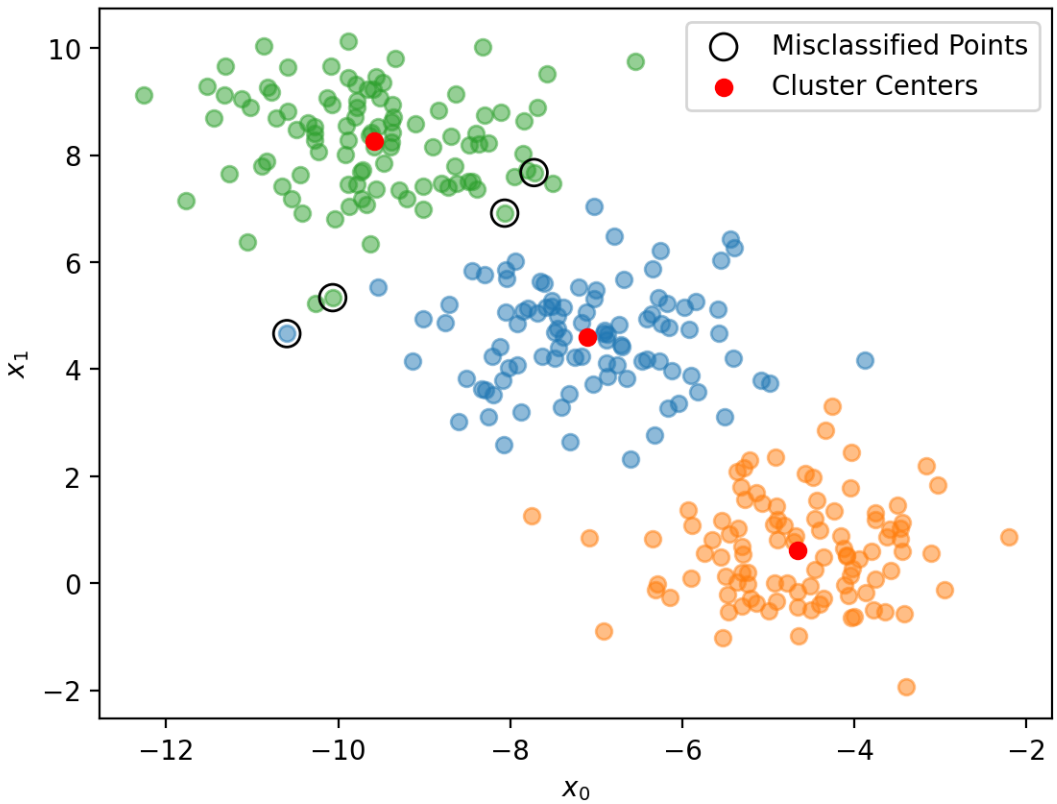
[ ]:
```python
def update_centers(x, labels):
    centers = []
    for label in np.unique(labels):
        label_idx = np.where(labels == label)[0]
        centers.append(np.average(x[label_idx], axis=0))
    return np.array(centers)
```

Finally write a function `myKMeans(x, init_centers)` which will run the KMeans algorithm, provided all the points in `x` and the coordinates of the initial cluster centers in `init_centers`. Run the algorithm until there is no change in cluster membership in subsequent iterations. Your function should return both the `labels`, the labels of each point in x, and `centers`, the final coordinates of each of the cluster centers. - `x` is a 300×2 numpy array containing the coordinates of all the points in the dataset - `init_centers` is a 3×2 numpy array containing the coordinates of the three cluster centers provided to you - **return** `labels` and `centers` as defined above

[ ]:
```python
def myKMeans(x, init_centers):
    centers = init_centers
    for i in range(100):
        labels = assign_labels(x, centers)
        new_centers = update_centers(x, labels)
        dist = 0
        for center, new_center in zip(centers, new_centers):
            dist += np.sum(np.power(np.subtract(center, new_center),2))
        centers = new_centers
        if (dist < 0.001): return labels, centers
```

Now use your `myKMeans()` function to cluster the provided data points `x` and set the initial cluster centers as `init_centers = np.array([[-5,5],[0,0],[-10,10]])`. Then use the provided plotting function, `plotter(x,y,labels,centers)` to visualize your model's clustering.

[ ]:
```python
init_centers = np.array([[-5,5],[0,0],[-10,10]])
labels, centers = myKMeans(x, init_centers)

plotter(x, y, labels, centers)
```



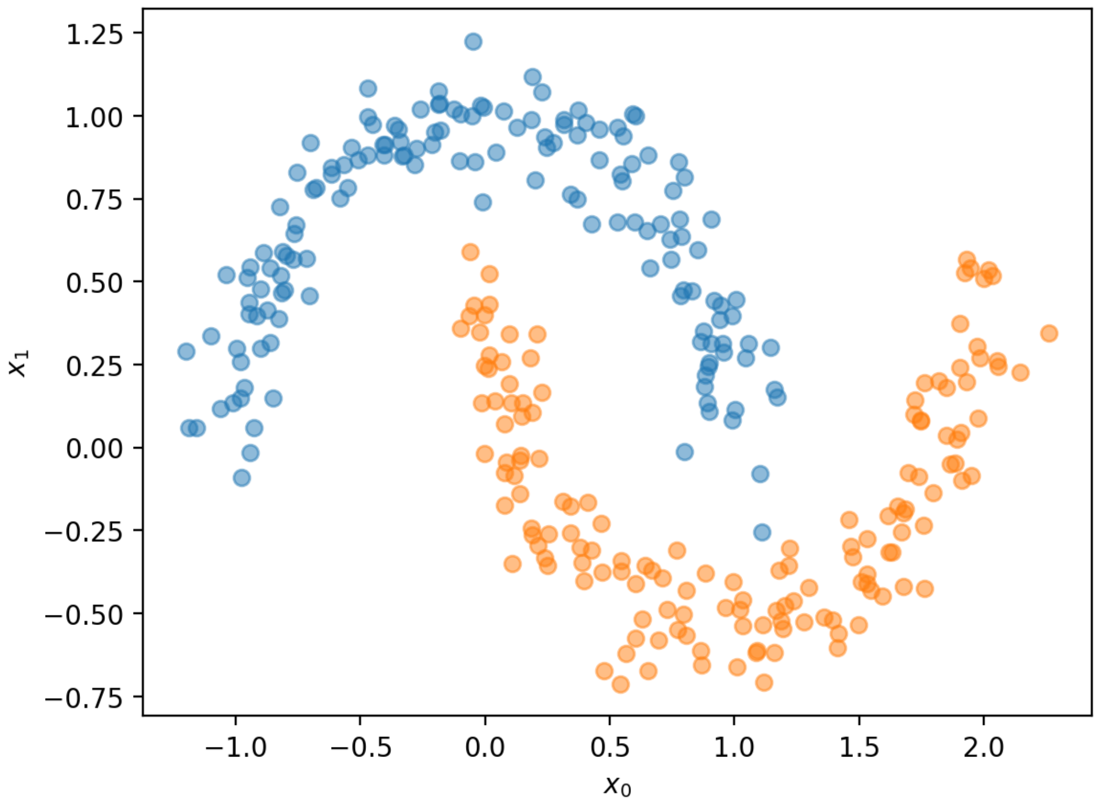
0.2 Moon Dataset

Now we will try using our `myKMeans()` function on a more challenging dataset, as generated below.

[ ]:
```
## DO NOT MODIFY
x,y = make_moons(n_samples = 300, noise = 0.1, random_state = 0)
```

Visualize the data using the `plotter(x,y)` function.

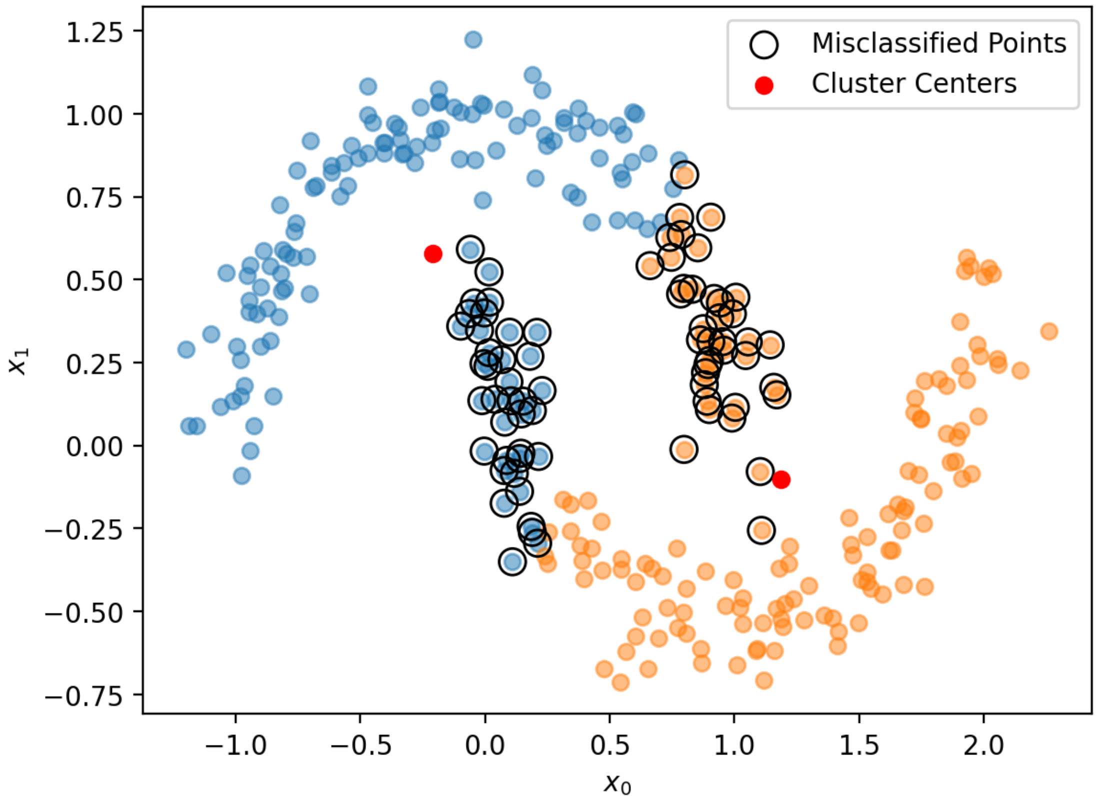
[ ]:
```
plotter(x, y)
```



Using your `myKMeans()` function and `init_centers = np.array([[0,1],[1,-0.5]])` cluster the data, and visualize the results using `plotter(x,y,labels,centers)`.

[ ]:

```
init_centers = np.array([[0,1],[1,-0.5]])
labels, centers = myKMeans(x, init_centers)

plotter(x, y, labels, centers)
```


Misclassified Points
Cluster Centers
x_0
x_1
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75