Ayush Patel

Problem 5

In [1]:
```
#pip install -U gensim
```

In [2]:
```
import numpy as np
from sklearn.metrics.pairwise import euclidean_distances
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
from sklearn.datasets import fetch_openml
from sklearn.metrics.pairwise import euclidean_distances
import nltk
from gensim.parsing import strip_tags, strip_numeric, strip_multiple_whitespac
from gensim.parsing import preprocess_string
from gensim import parsing
from sklearn.datasets import fetch_20newsgroups
import re
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
```

In [3]:
```
class DBSCAN:

    def __init__(self, X, eps, min_pts):
        self.eps = eps
        self.min_pts = min_pts
        self.X = X
        self.n_rows = X.shape[0]

        #  0 - unclassified
        # -1 - Noise
        # > 0 - cluster id the point belongs to
        self.label = np.zeros(self.n_rows)

        # preconpute distance matrix
        self.distances = euclidean_distances(self.X)

    def indicesInRange(self, pid):
        return np.where(self.distances[pid] <= self.eps)[0]

    def fit(self):
        c_id = 0

        for n in range(self.n_rows):
            if self.label[n] != 0:
                continue

            # get neighbours for point
            neighbours = self.indicesInRange(n)

            # density check
            if len(neighbours) < self.min_pts:
                self.label[neighbours] = -1
                continue

            c_id += 1
            self.label[n] = c_id

            # grow cluster
            index = 0
            while index < len(neighbours):
                new_n = neighbours[index]
                index += 1

                # check if already processed
                if self.label[new_n] > 0:
                    continue

                # add point to cluster
                self.label[new_n] = c_id
                new_neighbours = self.indicesInRange(new_n)

                # if core point, add to original neighbours
                if len(new_neighbours) >= self.min_pts:
                    neighbours = np.concatenate((neighbours, new_neighbours))


    def fit_and_plot(self):
```

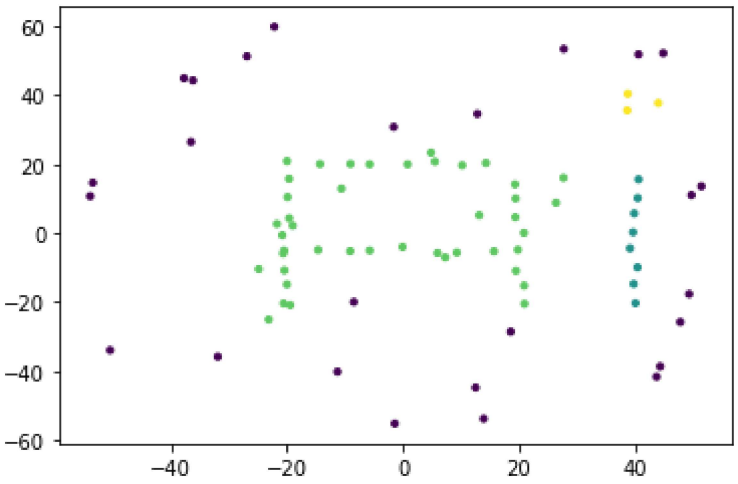
```
        self.fit()
        self.plot()

    def plot(self, size=10):
        plt.scatter(self.X[:, 0], self.X[:, 1], c = self.label, s = size)
```

In [4]:
```
df = pd.read_csv('dbscan.csv', index_col = 1)
```

In [5]:
```
X = df.iloc[:, 1:3].values
```

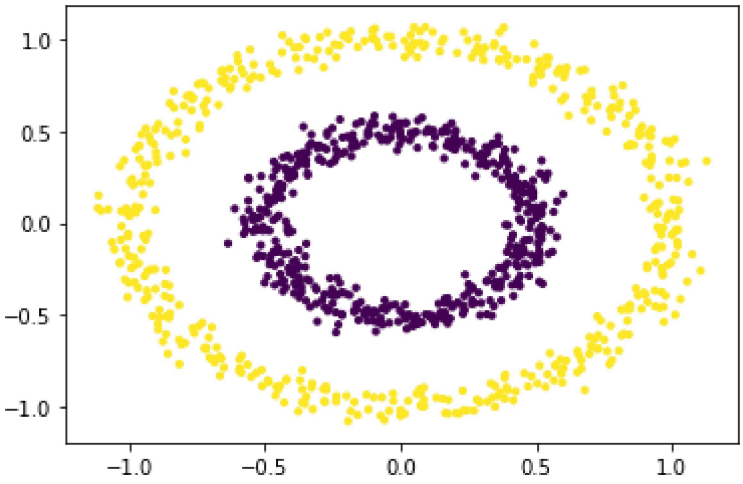
In [6]:
```
Y = DBSCAN(X, 7.5, 3)
p5_labels = Y.fit_and_plot()
```



Problem 6

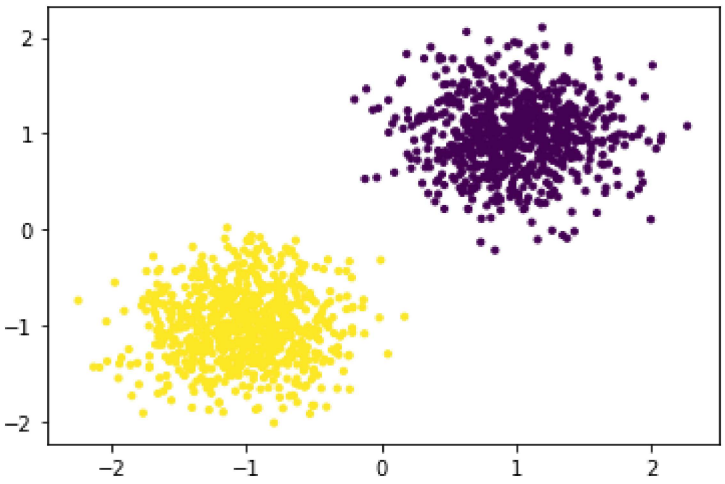
In [7]:
```
circles = pd.read_csv('circle.csv')
```

In [8]:
```
circles_X = circles.values
cir_db = DBSCAN(circles_X, eps = 0.1, min_pts = 1)
cir_db.fit_and_plot()
```



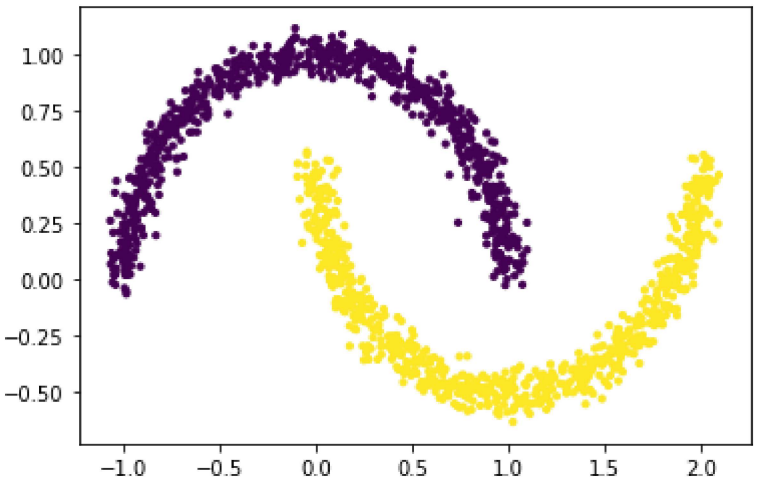
In [9]:
```
blobs = pd.read_csv('blobs.csv')
```

In [10]:
```
blobs_X = blobs.values
blobs_db = DBSCAN(blobs_X, eps = 0.31, min_pts = 1)
blobs_db.fit_and_plot()
```



In [11]:
```
moons = pd.read_csv('moons.csv')
```

In [12]:
```
moons_X = moons.values
moons_db = DBSCAN(moons_X, eps = 0.14, min_pts = 1)
moons_db.fit_and_plot()
```



Problem 7

In [13]:
```
def DBSCAN(X, eps, min_samples):
    # Compute distance matrix
    D = euclidean_distances(X)

    # Initialize list of visited points and labels
    visited = np.zeros(X.shape[0], dtype=bool)
    labels = np.zeros(X.shape[0], dtype=int)

    # Initialize label counter
    label_counter = 0

    # For each unvisited point
    for i in range(X.shape[0]):
        if not visited[i]:
            visited[i] = True

            # Find all neighbors within eps distance
            neighbors = np.where(D[i] < eps)[0]

            # If the point has less than min_samples neighbors, label it as no
            if len(neighbors) < min_samples:
                labels[i] = -1
            else:
                # Expand the cluster by visiting all neighbors
                label_counter += 1
                labels[i] = label_counter
                cluster = neighbors[visited[neighbors] == False]
                while cluster.size > 0:
                    j = cluster[0]
                    visited[j] = True
                    labels[j] = label_counter
                    j_neighbors = np.where(D[j] < eps)[0]
                    if len(j_neighbors) >= min_samples:
                        cluster = np.concatenate((cluster, j_neighbors[visited
                    cluster = cluster[1:]

    return labels
```

In [14]:
```
def calc_purity(y_pred, y_train):
    # Map cluster labels to non-negative integers
    offset = np.min(y_pred)
    y_pred_mapped = y_pred - offset
    y_train_mapped = y_train - offset

    purity_df = pd.DataFrame({'actual': y_train_mapped, 'pred': y_pred_mapped}

    # Group by actual and predicted values, and count occurrences
    count_df = purity_df.groupby(['actual', 'pred']).size().reset_index(name='

    # Drop duplicate actual values, keeping the first predicted value for each
    unique_df = count_df.sort_values(['actual', 'count_pred'], ascending=[True

    # Calculate purity as the sum of correctly predicted samples divided by th
    purity = unique_df['count_pred'].sum() / len(y_pred)

    return purity


def calc_gini(y_pred,y_train):
  y_pred = np.array(y_pred).astype(int)
  y_train = np.array(y_train).astype(int)
  gini = 0
  for label in np.unique(y_train):
      index = np.where(y_train == label)[0]
      if len(index) > 0:
          p = np.bincount(y_pred[index]).astype(int) / len(index)
          gini += len(index) / len(y_train) * (1 - np.sum(p ** 2))

  return gini
```

DBSCAN for Fashion MNIST

In [15]:
```
# Load the Fashion MNIST dataset
X, y = fetch_openml('Fashion-MNIST', version=1, return_X_y=True)

# Take a random sample of 500 images
np.random.seed(42)
sample_indices = np.random.choice(X.shape[0], size=500, replace=False)
X_sample = X[sample_indices]
y_sample = y[sample_indices].astype(int) # new code to create y_sample

# Flatten the images to obtain a 500 x 784 array
X_sample = X_sample.reshape(X_sample.shape[0], -1)

labels = DBSCAN(X_sample, eps=1100, min_samples=4)
```

In [16]:
```
np.unique(labels)
```

Out[16]:
```
array([-1,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [17]:
```
np.unique(y_sample)
```

Out[17]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [18]:
```
# Calculate purity and Gini index
mask = labels != -1
purity = calc_purity(labels[mask], y_sample[mask])
gini = calc_gini(labels[mask], y_sample[mask])

print("Purity:", purity)
print("Gini index:", gini)
```

```
Purity: 0.8255813953488372
Gini index: 0.26222420989862855
```

DBSCAN for 20 NG

In [19]:
```
transform_to_lower = lambda s: s.lower()
remove_emails = lambda s: re.sub(r'^[a-zA-Z0-9+_.-]+@[a-zA-Z0-9.-]+$', '', s)
remove_single_char = lambda s: re.sub(r'\s+\w{1}\s+', '', s)

CLEAN_FILTERS = [remove_emails,
                 strip_tags,
                 strip_numeric,
                 remove_emails,
                 strip_punctuation,
                 strip_multiple_whitespaces,
                 transform_to_lower,
                 remove_stopwords]

def cleaningPipe(document):
    processed_words = preprocess_string(document, CLEAN_FILTERS)

    return processed_words

def joinList(processed_words):
    return ' '.join(processed_words)

def basicStemming(text):
    return parsing.stem_text(text)


newsgroups_train = fetch_20newsgroups(subset='train')

ng_df_train = pd.DataFrame({"news" : newsgroups_train["data"] ,"class" : newsg
ng_df_train["cleanedText"] = ng_df_train["news"].apply(cleaningPipe).apply(joi
ng_df_train.head()
```

Out[19]:

	news	class	cleanedText
0	From: lerxst@wam.umd.edu (where's my thing)\nS...	7	lerxst wam umd edu s thing subject car nntp po...
1	From: guykuo@carson.u.washington.edu (Guy Kuo)...	4	guykuo carson u washington edu gui kuo subject...
2	From: twillis@ec.ecn.purdue.edu (Thomas E Will...	4	twilli ec ecn purdu edu thoma e willi subject ...
3	From: jgreen@amber (Joe Green)\nSubject: Re: W...	1	jgreen amber joe green subject weitek p organ ...
4	From: jcm@head-cfa.harvard.edu (Jonathan McDow...	14	jcm head cfa harvard edu jonathan mcdowel subj...

In [20]:
```
num_points = 5000

ng_df_train = ng_df_train.iloc[:num_points,:]

vectorizer = TfidfVectorizer(stop_words="english")
X_train_ng = vectorizer.fit_transform(np.array(ng_df_train["cleanedText"]))

X = pd.DataFrame(X_train_ng.toarray())
Y = np.array(ng_df_train["class"])
```

In [21]:
```
dbscan = DBSCAN(X , eps = 1.3, min_samples = 2)
```

In [22]:
```
np.unique(dbscan)
```

Out[22]:
```
array([-1,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22])
```

In [23]:
```
np.unique(Y)
```

Out[23]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

In [24]:
```
# Calculate purity and Gini index
mask = dbscan != -1
purity = calc_purity(dbscan[mask], Y[mask])
gini = calc_gini(dbscan[mask], Y[mask])

print("Purity:", purity)
print("Gini index:", gini)
```

```
Purity: 0.9893260778568439
Gini index: 0.0209281314484811
```

DBSCAN for Household

In [26]:
```
housing_df = pd.read_csv('household_power_consumption.txt', delimiter=';', na_
```

In [27]:
```
housing_df.isna().any()
```

Out[27]:
```
Date                    False
Time                    False
Global_active_power      True
Global_reactive_power    True
Voltage                  True
Global_intensity         True
Sub_metering_1           True
Sub_metering_2           True
Sub_metering_3           True
dtype: bool
```

In [28]:
```
housing_df = housing_df.drop(['Date', 'Time'], axis=1)

housing_df.head()
```

Out[28]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_r
0	4.216	0.418	234.84	18.4	0.0	
1	5.360	0.436	233.63	23.0	0.0	
2	5.374	0.498	233.29	23.0	0.0	
3	5.388	0.502	233.74	23.0	0.0	
4	3.666	0.528	235.68	15.8	0.0	

In [29]:
```
housing_df.shape
```

Out[29]:
```
(2075259, 7)
```

In [30]:
```
housing_df.dropna(inplace=True)
housing_df.shape
```

Out[30]:
```
(2049280, 7)
```

In [31]:
```
# scaler = StandardScaler()
num_points = 5000

# df_norm = pd.DataFrame(scaler.fit_transform(df.values),columns=df.columns)
X = housing_df.iloc[:num_points,:]
```

In [32]:
```
X.shape
```

Out[32]:
```
(5000, 7)
```

In [33]:
```
dbscan = DBSCAN(X , eps = 3.1 , min_pts = 15)
```

In [34]:
```
np.unique(dbscan)
```

Out[34]:
```
array([0., 1., 2., 3., 4., 5., 6., 7.])
```

In [35]:
```
np.unique(X)
```

Out[35]:
```
array([0.0000e+00, 4.6000e-02, 4.8000e-02, ..., 2.4894e+02, 2.4907e+02,
       2.4937e+02])
```

In [36]:
```
from sklearn.metrics import silhouette_score
```

In [37]:
```
silhouette_avg = silhouette_score(X,dbscan)

print("The silhouette score is:", round(silhouette_avg,3))
```

```
The silhouette score is: 0.655
```

Reason:

Although DBSCAN is frequently regarded as a successful clustering algorithm, it frequently fails when the densities of several clusters are not the same. Hence, different values of epsilon and min points are required for each clusters, which is challenging to accommodate. Moreover, DBSCAN is extremely sensitive to changes in the values of epsilon and min points.

Problem 8

In [38]:
```
from sklearn.datasets import fetch_openml

mnist = fetch_openml('mnist_784', version = 1)
X = mnist.data[:10000]
y = mnist.target[:10000].astype(int)
```

In [39]:
```
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import pairwise_distances
import numpy as np

# Perform agglomerative clustering with 20 clusters
cluster = AgglomerativeClustering(n_clusters = 10, linkage = 'ward')
cluster_labels = cluster.fit_predict(X)

# Calculate Gini index
dist = pairwise_distances(X, metric='euclidean')
gini = 0
for i in range(10):
    index = np.where(y == i)[0]
    if len(index) > 0:
        p = np.bincount(cluster_labels[index]).astype(float) / len(index)
        gini += len(index) / len(y) * (1 - np.sum(p ** 2))
print("Gini index:", gini)

# Calculate purity
purity = 0
for i in range(10):
    index = np.where(y == i)[0]
    if len(index) > 0:
        labels = cluster_labels[index]
        most_common_label = np.bincount(labels).argmax()
        purity += np.sum(labels == most_common_label)
print("Purity:", purity / len(y))
```

```
Gini index: 0.3978513388341974
Purity: 0.6636
```

In [ ]: