gmm_vs_kmeans_assignment_qlr-vdef

December 12, 2020

1 Gaussian Mixture Model vs. KMeans with Fashion-MNIST

Quentin Le Roux – MSc Data Science & AI 1

1.1 0. Introduction

1.1.1 0.1 Notebook Content

1.1.2 0.2 Task Description

Clustering is abour grouping similar objects together. It is a type of **unsupervised learning**: the generic application of machine learning to bunch data together based on similar criteria. Unsupervised learning algorithms will work to partition data into smaller groups of similar objects

while trying to maximize the differences between each group. **KMeans** and **Gaussian Mixture Models** are examples of clustering algorithms.

The goal of this notebook is to compare the applications of the KMeans and GMM algorithms on the Fashion-MNIST dataset. Afterward, we will explore the impacts of changing the GMM's parameters using a gridsearch-style method.

1.1.3 0.3 Importing Modules

[1]:
```
from keras.datasets import fashion_mnist

from itertools import cycle
import math
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random as rd

from scipy import ndimage
from scipy.ndimage.filters import maximum_filter

from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from sklearn import decomposition
from sklearn.metrics import homogeneity_score
from sklearn.metrics import silhouette_score
from sklearn.metrics import v_measure_score
from sklearn.mixture import GaussianMixture
from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
```

1.2 1. Importing the dataset: Fashion-MNIST

1.2.1 1.1. Importing the dataset

Our first step is to import our dataset: Fashion-MNIST. The copyright for Fashion-MNIST is held by Zalando SE and Fashion-MNIST is licensed under the MIT license.

Fashion-MNIST is a dataset of Zalando's article images. It consists of a **training set of 60,000 examples** and a **test set of 10,000 examples**. Each example is a 28x28 grayscale image, associated with a **label from 10 classes**. Zalando intends Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

Image shape:

Each image is **28 pixels in height and 28 pixels in width, for a total of 784 pixels in total**.

Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255.

Labels:

Each training and test example is assigned to one of the following labels:

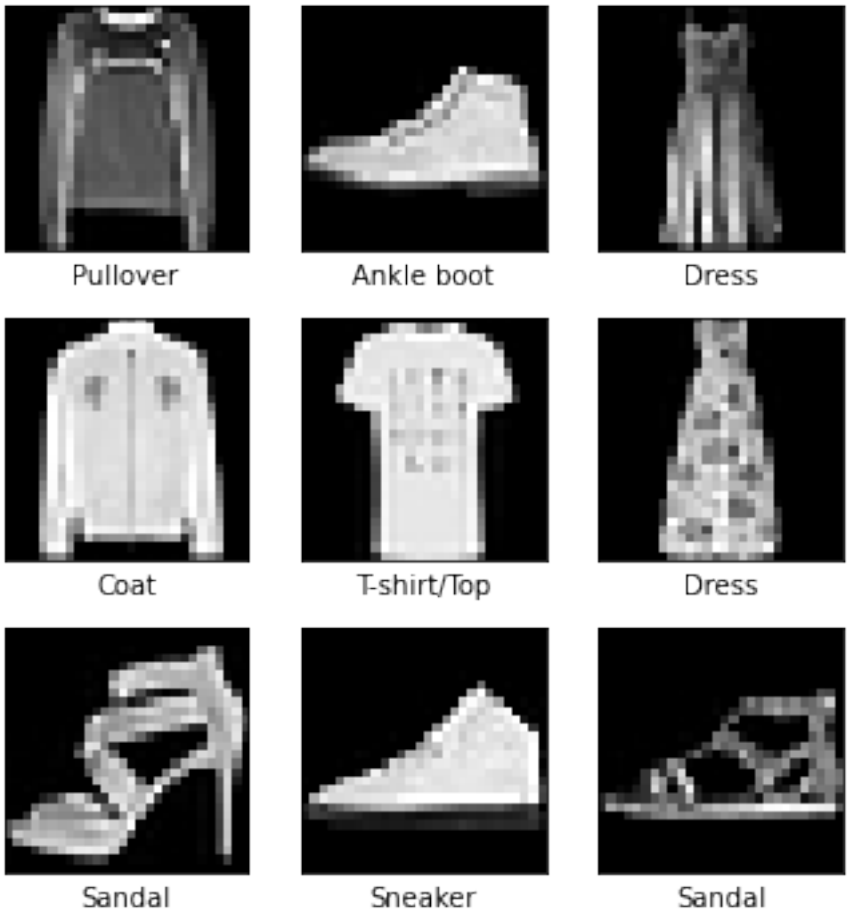
- 0 – T-shirt/top
- 1 – Trouser
- 2 – Pullover
- 3 – Dress
- 4 – Coat
- 5 – Sandal
- 6 – Shirt
- 7 – Sneaker
- 8 – Bag
- 9 – Ankle boot

[2]:
```
# We import the dataset's training and test sets and check the shape of the
# resulting arrays.

(x_train_origin, y_train), (x_test_origin, y_test) = fashion_mnist.load_data()

labelNames = ["T-shirt/Top", "Trouser", "Pullover", "Dress",
              "Coat", "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]

print(f"Shape of x_train_origin: {x_train_origin.shape}")
print(f"Shape of y_train: {y_train.shape}")
print(f"Shape of x_test_origin: {x_test_origin.shape}")
print(f"Shape of y_test: {y_test.shape}")
```

```
Shape of x_train_origin: (60000, 28, 28)
Shape of y_train: (60000,)
Shape of x_test_origin: (10000, 28, 28)
Shape of y_test: (10000,)
```

[3]:
```
# We sample some images to check the content of the dataset

length = 3
plt.figure(figsize=(5, 5))
for i in range(length * length):
    temp = rd.randint(0, len(x_train_origin)+1)
    image = x_train_origin[temp]
    plt.subplot(length, length, i+1)
    plt.imshow(image, cmap='gray')
    plt.xticks([])
    plt.yticks([])
    plt.xlabel(labelNames[y_train[temp]])
    plt.tight_layout()
```

```
plt.show()
```



To preprocess the data, we must standardardize the whole dataset. As such, we temporarily merge train and test sets.

[4]:
```
# We concatenate our training and test sets to perform our pre-processing

X = np.concatenate((x_train_origin, x_test_origin))
print(f"Shape of X: {X.shape}")
```

```
Shape of X: (70000, 28, 28)
```

1.2.2 1.2. Data Preprocessing

Data Pre-Processing is an important first step towards machine learning. Below, we will go through three different pre-processing steps.

Normalization: Each item in the dataset is a grayscale picture (i.e. 1 channel instead of 3 for RGB) where each pixel is represented by an integer value between 0 and 255. We need to rescale

each pixel value to the [0,1] range, i.e., normalize them. This rescaling is done by dividing each pixel value by 255.

[5]:
```
# Before
print(f"One pixel before normalization: {X[0][5][15]}")

# Normalization
X = X / 255.

# After
print(f"One pixel after normalization: {X[0][5][15]}")
```

```
One pixel before normalization: 204
One pixel after normalization: 0.8
```

Flattening: Since each image is a 2-dimensional picture of 28 pixels by 28 pixels, we must reshape our data so it can be fed into our models. To do so, we reshape our image into a single dimension of size 28*28, i.e. 784.

[6]:
```
# Flattening

X = X.reshape(len(X), 784)

print(f"Shape of X: {X.shape}")
```

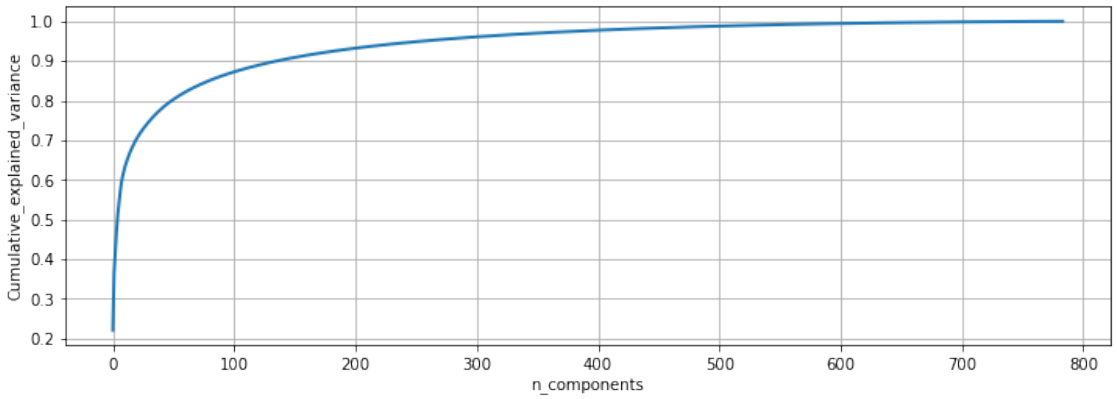
```
Shape of X: (70000, 784)
```

Feature Extraction: Principal Component Analysis for dimensionality reduction (or PCA for short) is used to reduce the complexity of a model, avoid overfitting, and visualize high dimensional data. It is an unsupervised linear transformation technique that is widely used across different fields, most prominently for feature extraction and dimensionality reduction.

We aim to use PCA reduction to reduce the complexity of the high dimensional (784) Fashion-MNIST dataset. To do so, we compute the cumulative explained variance with a number of components from 0 to 784. With this analysis, we will be able to choose a number of components that balances between explaining a high variance within the dataset and realizing a strong reduction in complexity.

[7]:
```
# Feature extraction

pca = decomposition.PCA()
X_fe = StandardScaler().fit_transform(X)

pca.n_components = 784
X_fe = pca.fit_transform(X_fe)

percentage_var_explained = pca.explained_variance_ / np.sum(pca.
 ↪explained_variance_)
cum_var_explained = np.cumsum(percentage_var_explained)
```

[8]:
```
# We plot the PCA spectrum

plt.figure(1, figsize=(12, 4))
plt.clf()
plt.plot(cum_var_explained, linewidth=2)
plt.axis("tight")
plt.grid()
plt.xlabel("n_components")
plt.ylabel("Cumulative_explained_variance")
plt.show()
```



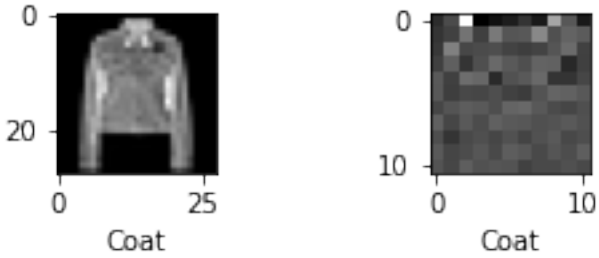
We find that 90%+ of the variance is explained once we reach above 120 components. As such we settled on a dimensionality reduction to 121 components, which provides us an easy 11*11 representation.

[9]:
```
# We reduce our number of components to 121

pca = decomposition.PCA(n_components=121)
X_fe = StandardScaler().fit_transform(X)
X = pca.fit_transform(X_fe)

print(f"Shape of x_train_processed: {X.shape}")
```

```
Shape of x_train_processed: (70000, 121)
```

[10]:
```
# We sample 1 image from the training set and pair it with its 11*11-components
# representation

for i in rd.sample(range(60000),1):
    i1 = x_train_origin[i]
    i2 = X[i].reshape(11,11)
    plt.subplot(length, length, 1)
    plt.imshow(i1, cmap='gray')
```

```
    plt.xlabel(labelNames[y_train[i]])
    plt.subplot(length, length, 2)
    plt.imshow(i2, cmap='gray')
    plt.xlabel(labelNames[y_train[i]])
    plt.tight_layout()
    plt.show()
```



Finally, we split our dataset back into a train and test set.

[11]:
```
# We retrieve our train and test sets

x_train = X[:60000]
x_test = X[60000:]

print(f"Shape of x_train: {x_train.shape}")
print(f"Shape of x_test: {x_test.shape}")
```

```
Shape of x_train: (60000, 121)
Shape of x_test: (10000, 121)
```

1.3 2. Implementation of KMeans

KMeans is an **unsupervised clustering algorithm** that tries to cluster the elements of a given dataset into a k number of clusters. It is a **centroid-based clustering**, i.e., we create k centroids, find the centres of each cluster and then assign each data point to the closest centre. This is the approach that the k-means clustering algorithm uses.

As such, it is an **iterative algorithm** meaning that we repeat multiple steps making progress each time. In more details, it proceeds as follow:

```
0. Choose a value for k
1. Selects k random points from the dataset as starting cluster centers (i.e. centroids)
2. For all data point in the dataset:
    2.1 calculates the distance (such as the euclidian distance (1)) between the point
        and the clusters
    2.2 assigns the point to the closest center (2)
3. Update each cluster center by taking the average of the points within the cluster
4. Repeat steps 2 and 3 until convergence, i.e., when the centers do not change
```

meaningfully

$$given\ x_i \in \mathbb{R}^m\ \forall i, k \in [1, ..., n]$$

$$d(x_i, center_k) = \sqrt{\sum_{j=1}^{m}(x_{i,j} - center_{k,j})^2}\ \forall i, k \in [1, ..., n] \quad (1)$$

$$cluster(x_i) = argmin\ d(x_i, center_k)\ \forall i, k \in [1, ..., n] \quad (2)$$

Performance evaluation for the KMeans algorithm can be done by calculating the sum of squared differences betweent our data points and centroids:

$$\sum_{i=0}^{n}(X_i - \bar{X})^2$$

With X_i a data point and $\bar{X}$ the centroid that point belongs to.

1.3.1 2.1. Simple Implementation From Scratch

We now have an idea about how the algorithm works. We can thus implement a simple version of it to see how it works. Using the `make_blobs` function from the SciKit-Learn library we implement a KMeans using the euclidian distance metric.

[12]:
```
def generate_dataset(number_of_points):
    """
    Generates a dataset of 2d coordinates (x, y), with x and y random values
    between 0 and 99.
    The set is generated with a random number of centers between 2 and 9.
    """
    X, _ = make_blobs(n_samples=number_of_points,
                      centers=rd.randint(2,10),
                      n_features=2,
                      random_state=0)
    return X

def compare_dict(dict1, dict2):
    """
    Checks if two dictionary are identical.
    """
    for key in dict1.keys():
        if not np.array_equal(dict1[key], dict2[key]): return False
    return True

def euclidian_distance(p1, p2):
    """
```

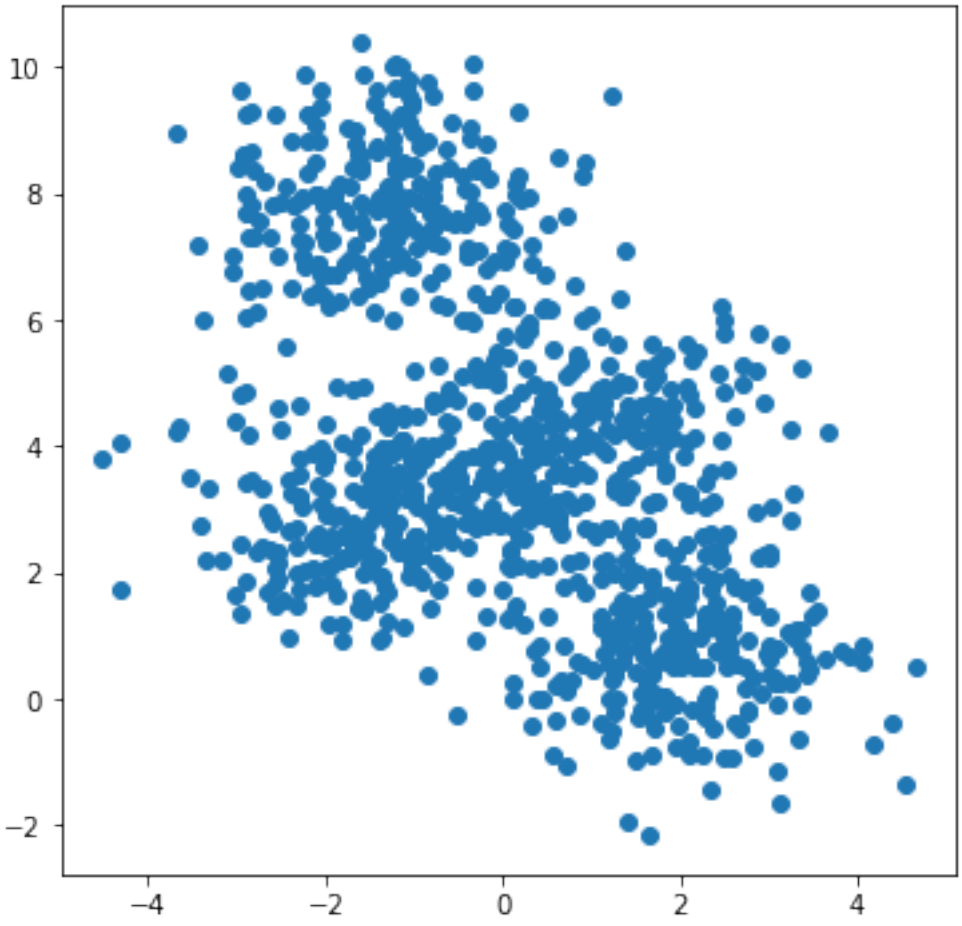
```
    Calculates the euclidian distance between two points in the R² space.
    """
    distance = 0
    for idx, item in enumerate(p1):
        distance += (item - p2[idx]) ** 2
    return math.sqrt(distance)
```

[13]:
```
class k_means():
    def __init__(self, nb_of_data_points, k=2):
        """
        Initializes the k_means class.
        """
        self.data = generate_dataset(nb_of_data_points)
        self.k = k
        self.iterated = 0
        self.colors = cycle(["g","r","b","c","m","y"])

        # Generates an empty dictionary to store each step's centroids
        self.model_centroids = {}

        # Generates an empty dictionary to store each step's classification
        self.model_classifications = {}


    def fit(self, max_iterations=100):
        """
        Fits the model with a maximum number of iterations by default of 100.
        """
        self.iterated = 0

        # Selects the initial k centroids
        centroids = rd.sample(list(self.data), 2)

        # Iterative steps to fit the model
        for iteration in range(max_iterations):
            print(f"Epoch {iteration}")

            # Generates an empty dictionary to store the iteration's
            # classification
            step_classification = {}
            for k in range(self.k): step_classification[k] = []

            # Calculates the euclidian distance between each data point
            # and each centroid
            # Records the nearest centroid
            for point in self.data:
                distances = list(map(lambda x: euclidian_distance(point, x),
```

```
                                    centroids))
                argmin = min(range(len(distances)),
                             key=distances.__getitem__)
                step_classification[argmin].append(point)

            # Records the state of the model after the iteration
            self.model_centroids[iteration] = centroids
            self.model_classifications[iteration] = step_classification

            # If no change has been identified between this iteration and the
            # last, the model will stop.
            if len(self.model_classifications)>1:
                if compare_dict(self.model_classifications[iteration],
                                self.model_classifications[iteration-1]):
                    self.iterated = iteration
                    print("No significant change has been achieved during" +\
                          f"epoch {self.iterated}. "+\
                          "Model is considered fitted.")
                    break

            # Updates the centroids
            centroids = []
            for classification in step_classification.values():
                if classification == []: centroids.append(np.zeros(3))
                else: centroids.append(np.mean(classification, axis=0))


    def plot_data(self):
        """
        Plots the distribution of the data.
        """
        # Declares the plot
        plt.figure(figsize=(6,6))

        # Plots the data without colors if the dataset was not iterated over.
        if self.model_centroids == {} or self.iterated == 0:
            plt.scatter(self.data[:,0],self.data[:,1])

        else:
            # Plots the centroids first
            for centroid in self.model_centroids[self.iterated]:
                plt.scatter(centroid[0], centroid[1],
                            marker="o", color="k", s=50, linewidths=5)

            # plots the data points
            for classification in self.model_classifications[self.iterated].
↪values():
```

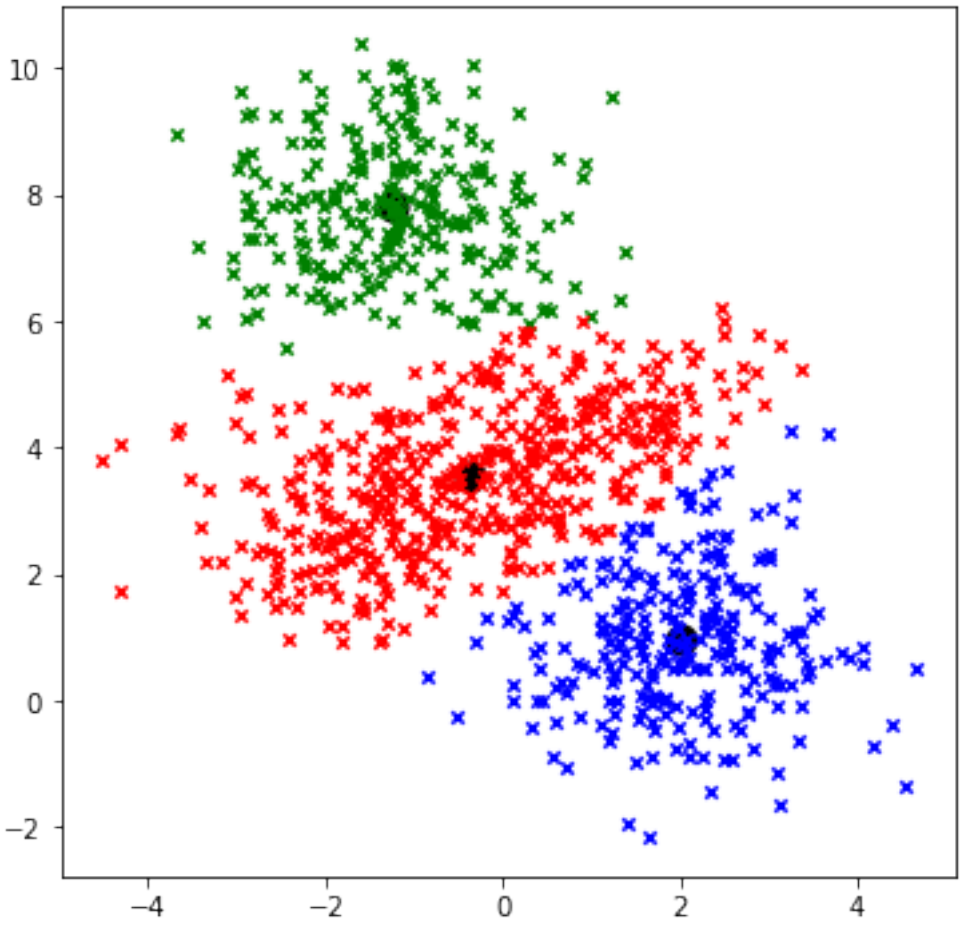
```
                color = next(self.colors)
                for feature in classification:
                    plt.scatter(feature[0], feature[1],
                                marker='x', color=color, s=20, linewidths=2)
        plt.show()
```

[14]:
```
# Example of a KMeans on a randomly generated 1000-point dataset with
# three clusters

model = k_means(1000, 3)
model.plot_data()
model.fit()
model.plot_data()
```




```
Epoch 0
Epoch 1
Epoch 2
Epoch 3
```

```
Epoch 4
Epoch 5
Epoch 6
Epoch 7
Epoch 8
Epoch 9
Epoch 10
Epoch 11
No significant change has been achieved during epoch 11. Model is considered
fitted.
```



We now see how a KMeans algorithm works.

1.3.2 2.2. Implementation using SciKit-Learn

As the simple from scratch KMeans works, and now that we have a better grasp on the different steps involved to fit the model, we can implement one using the SciKit-Learn library.

As such, we go back to our Fashion-MNIST dataset.

1. We set our hyperparameters:

[15]:
```
# Number of clusters we want to implement our model with
# We set it to match the number of classes in the Fashion-MNIST dataset
n_clusters = 10

# We choose a k-means++ initialization, a scikit-learn initialization
# method that selects initial cluster centers in a 'smart way'
# it is supposed to speed up the convergence
init = "k-means++"

# We indicate that our KMeans algorithm we be run with 50 different
# centroid seeds and our final output will be the one with the
# best results out of 50
n_init=50

# We indicate we want the model will not go over 100 iterations
max_iter = 100
```

2. We declare our model:

[16]:
```
model = KMeans(n_clusters=n_clusters,
               init=init,
               n_init=n_init,
               max_iter=max_iter,
               verbose=0,
               random_state=0).fit(x_train)
```

3. We predict labels on our test dataset using the resulting trained model:

[17]:
```
label_predictions = model.predict(x_test)
```

4. We output our resulting metrics:

Remark:

- The **homogeneity score** is a metric that checks whether the resulting clusters contain only data points which are members of a single class. It is *independent of the absolute values of the predicted labels*. The resulting score yields a real value between 0 and 1 with 1 standing for perfect homogeneous labeling.
- The **v-measure score** takes the homogeneity score and another underlying one (the completeness score, which measures whether all data points that belong to the same class are clustered within the same cluster) and provides a weighted measure: $\frac{(1+\beta).hc}{\beta.h+c}$ with β a factor that favors either the homogeneity or the completeness. The value of this evaluation metric is that it is independent of the number of class lavels, clusters, and of the size of the data. The resulting score yields a real value between 0 and 1 with 1 standing for perfectly complete labeling.
- The **silhouette** score measures the overlapping of clusters over all the samples. The resulting score yields a real value between -1 and 1 with values near 0 indicating overlapping clusters,

and a negative value indicating that a sample has been assigned to the wrong cluster.

[18]:
```
kmeans_h_score = homogeneity_score(y_test, label_predictions)
kmeans_v_score = v_measure_score(y_test, label_predictions)
kmeans_s_score = silhouette_score(x_test, label_predictions)

print(kmeans_h_score, kmeans_v_score, kmeans_s_score)
```

```
0.4930005331280772 0.5052442593457442 0.15585513066658788
```

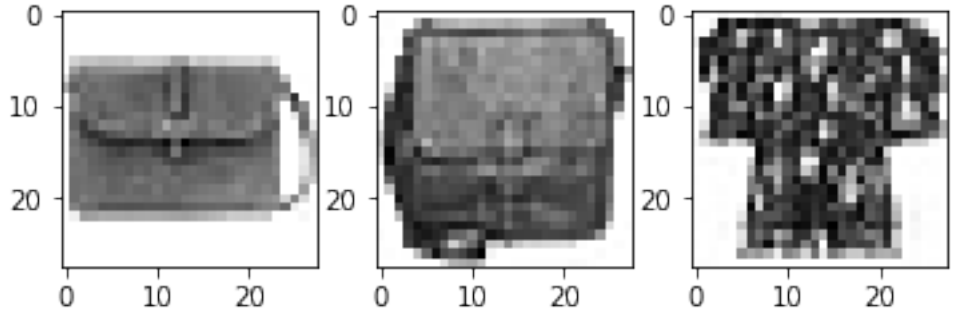
See our detailed observations below.

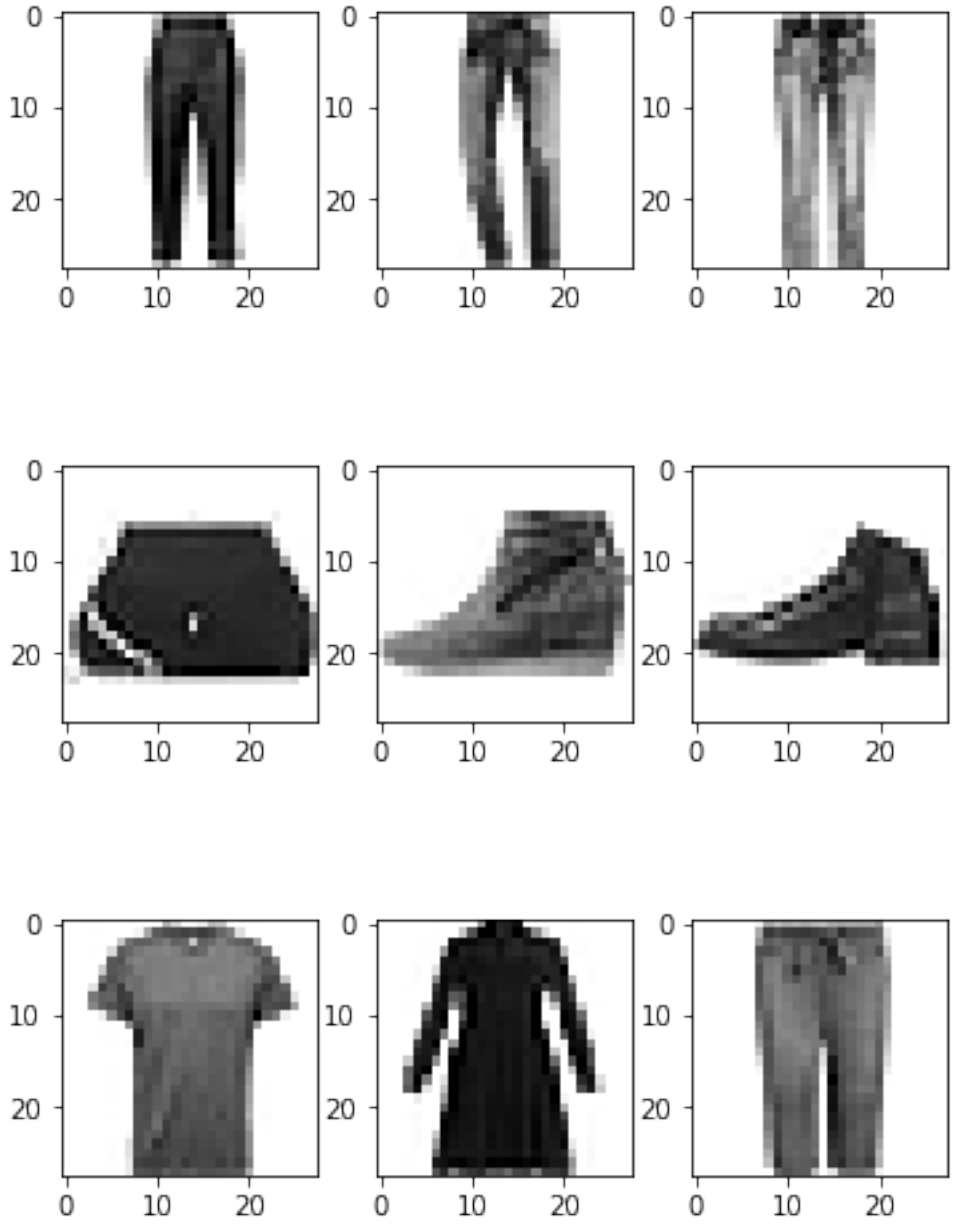
1.3.3 2.3. Observations

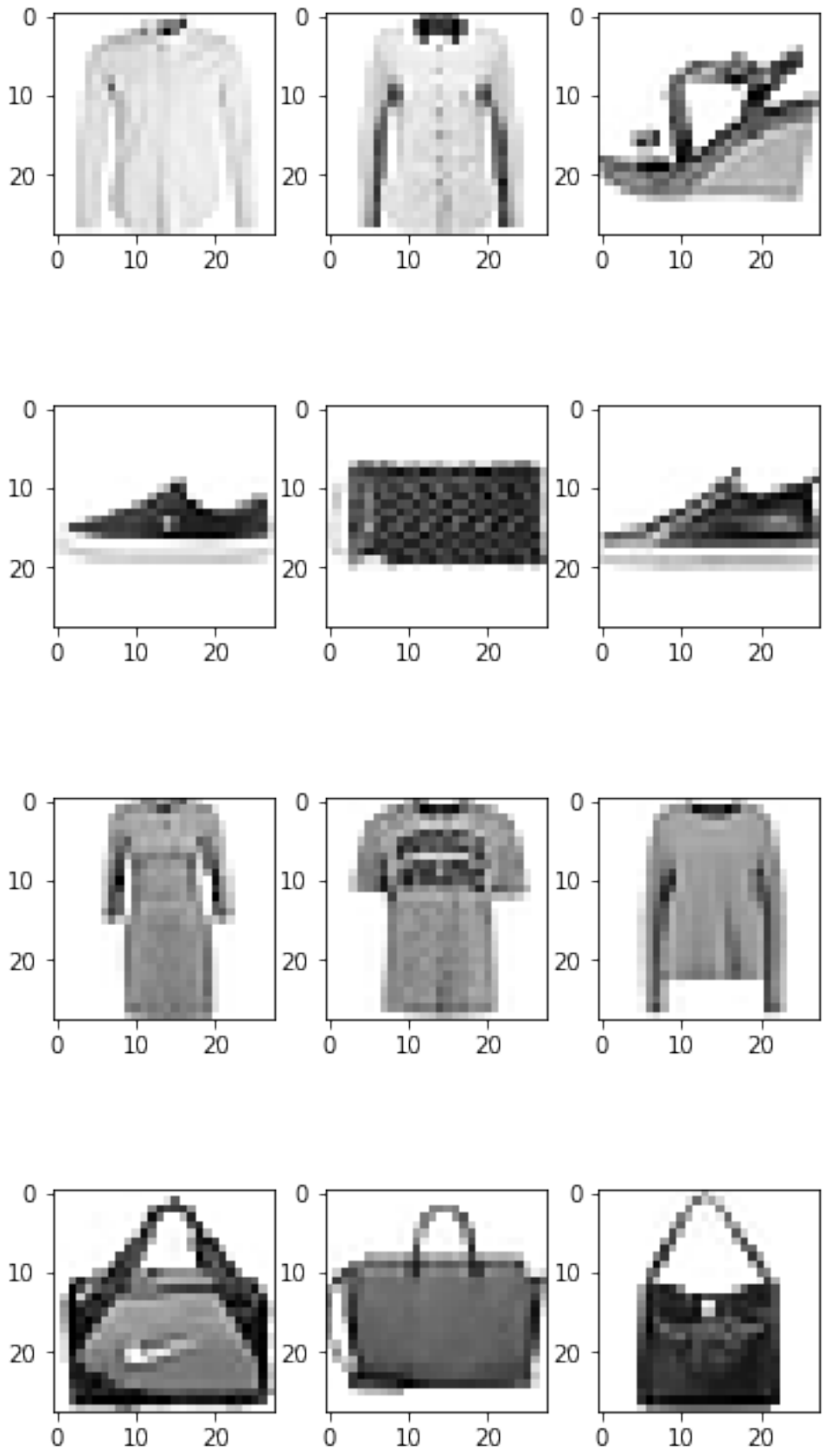
We visualize the resulting clustering on our test set.

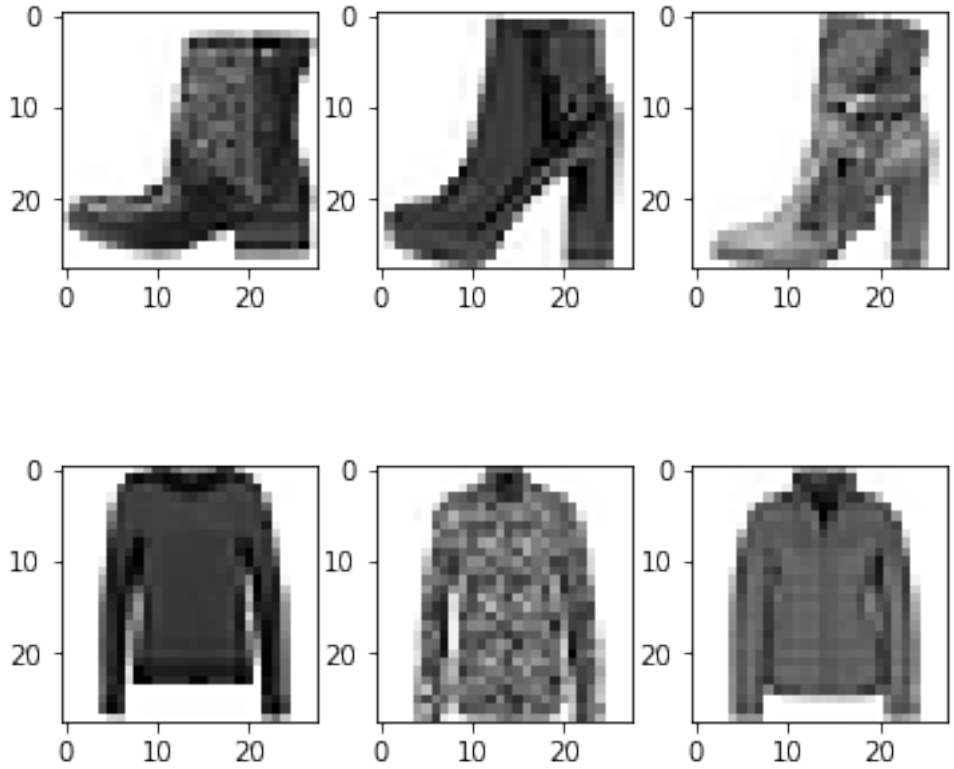
Each row presents a trio of pictures that were assigned to the same cluster.

[19]:
```
label_number = 10

cluster_index= [[] for i in range(label_number)]
for i, label in enumerate(label_predictions):
    for n in range(label_number):
        if label == n:
            cluster_index[n].append(i)
        else:
            continue

for j, _ in enumerate(labelNames):
    plt.figure(figsize=(6,6));
    clust = j
    num = 10
    for i in range(1,4):
        plt.subplot(3, 3, i)
        plt.imshow(x_test_origin[cluster_index[clust][i+20]],
                   cmap = plt.cm.binary);

plt.show()
```



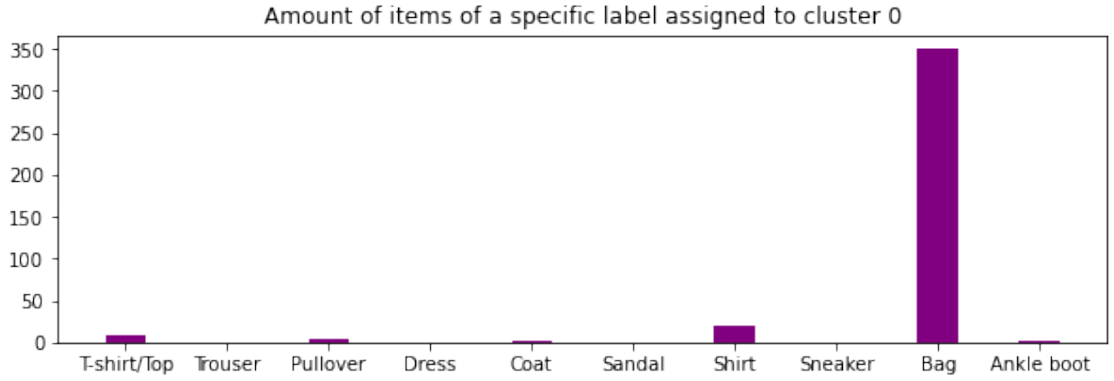


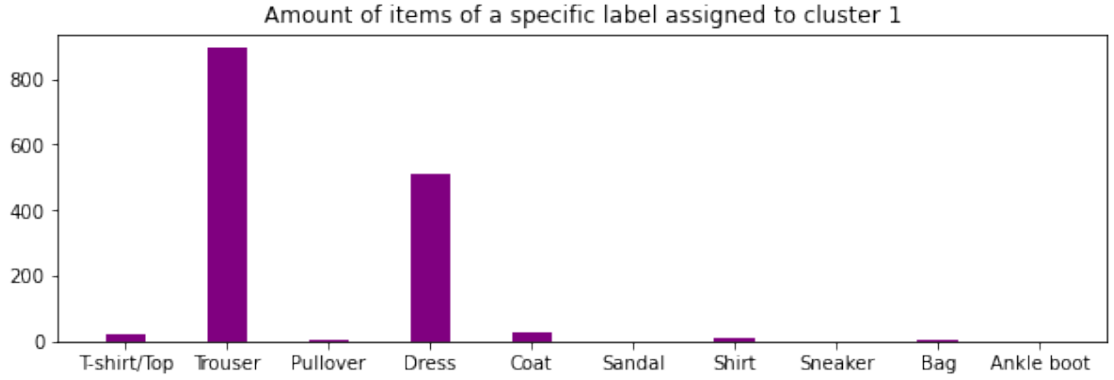


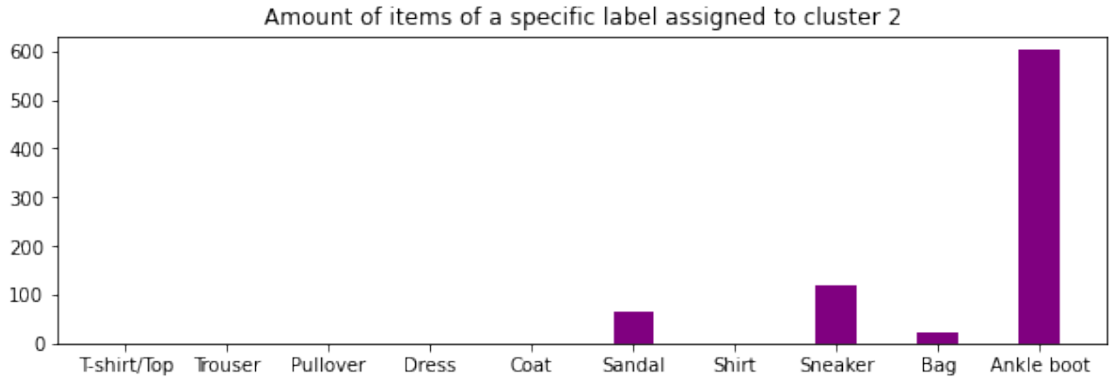


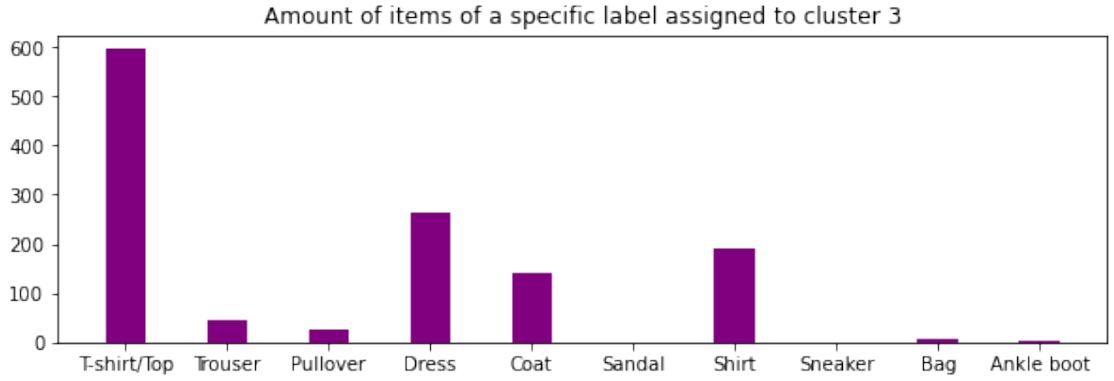
To provide a better view of what is happening, we visualize the repartition of the test set's items between each clusters.

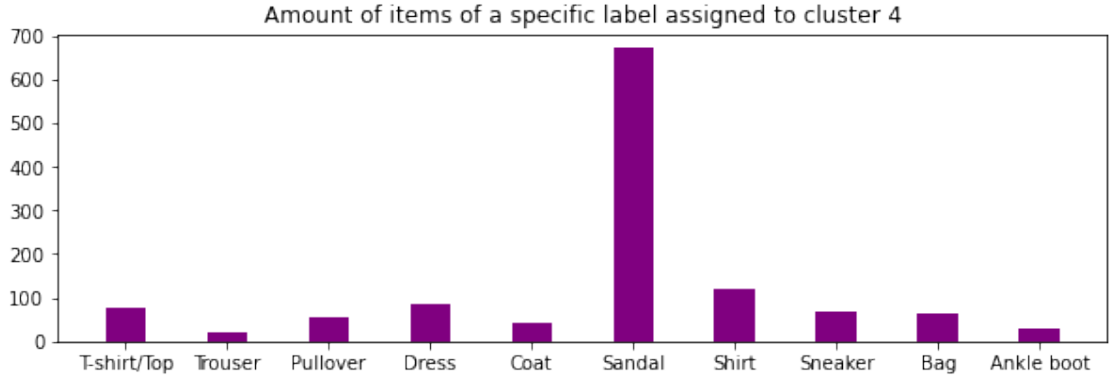
[20]:
```
clusters={}
for i in range(10):
    clusters[i]=[0]*10

for index, item in enumerate(label_predictions):
    clusters[item][y_test[index]] += 1

for i in range(len(clusters)):
    fig = plt.figure(figsize=(10,3))
    plt.bar(labelNames, clusters[i], color="purple", width=0.4)
    plt.title(f"Amount of items of a specific label assigned to cluster {i}")
    plt.show()
```

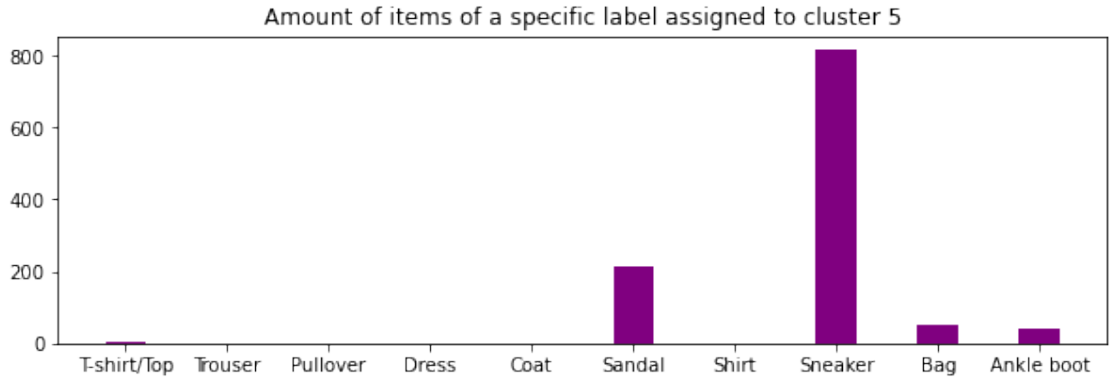

Amount of items of a specific label assigned to cluster 0
350
300
250
200
150
100
50
0
T-shirt/Top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot

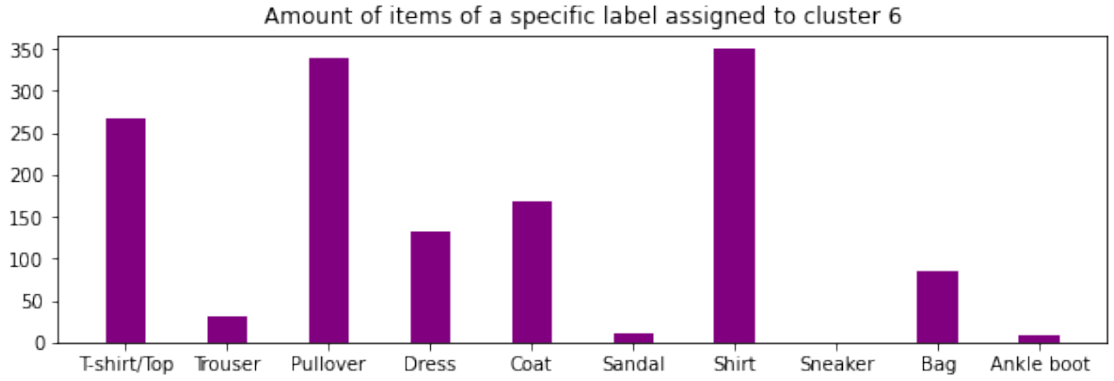

Amount of items of a specific label assigned to cluster 1
800
600
400
200
0
T-shirt/Top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot

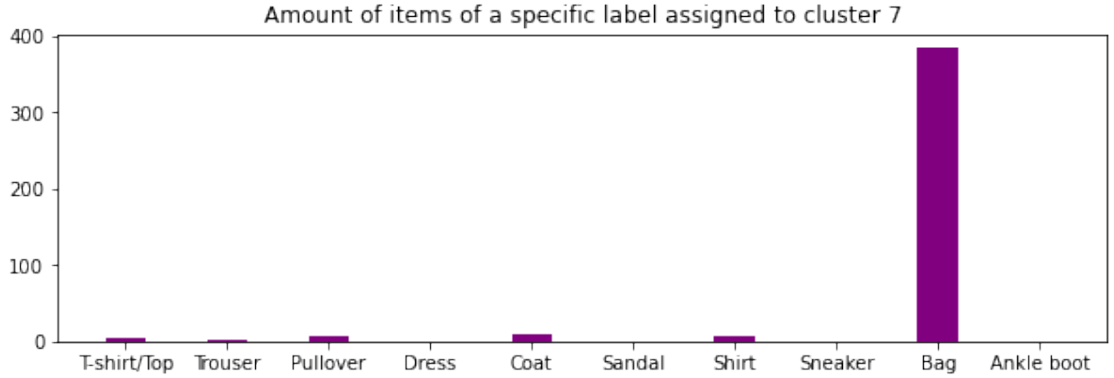

Amount of items of a specific label assigned to cluster 2
600
500
400
300
200
100
0
T-shirt/Top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot

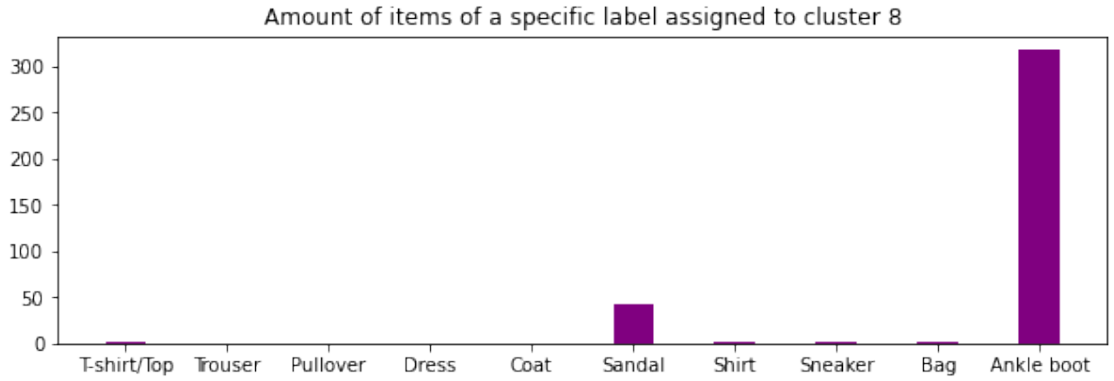

Amount of items of a specific label assigned to cluster 3
600
500
400
300
200
100
0
T-shirt/Top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot

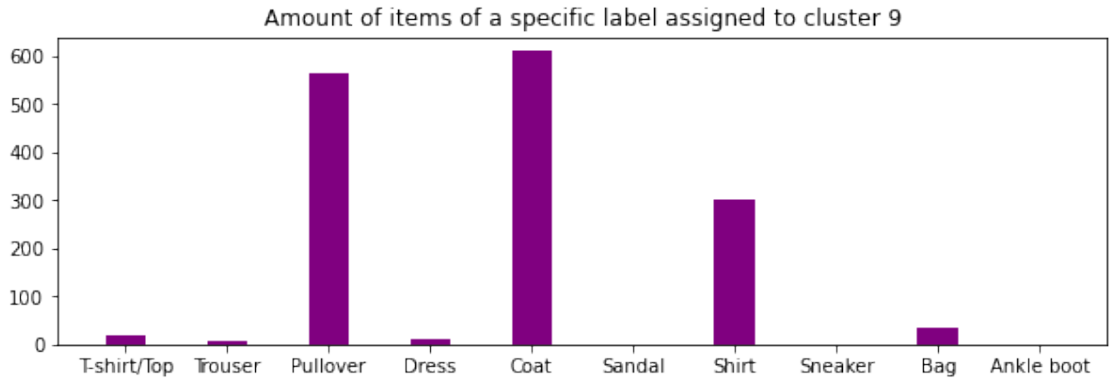

Amount of items of a specific label assigned to cluster 4
700
600
500
400
300
200
100
0
T-shirt/Top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot


Amount of items of a specific label assigned to cluster 5
800
600
400
200
0
T-shirt/Top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot


Amount of items of a specific label assigned to cluster 6
350
300
250
200
150
100
50
0
T-shirt/Top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot


Amount of items of a specific label assigned to cluster 7
400
300
200
100
0
T-shirt/Top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot


Amount of items of a specific label assigned to cluster 8
300
250
200
150
100
50
0
T-shirt/Top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot

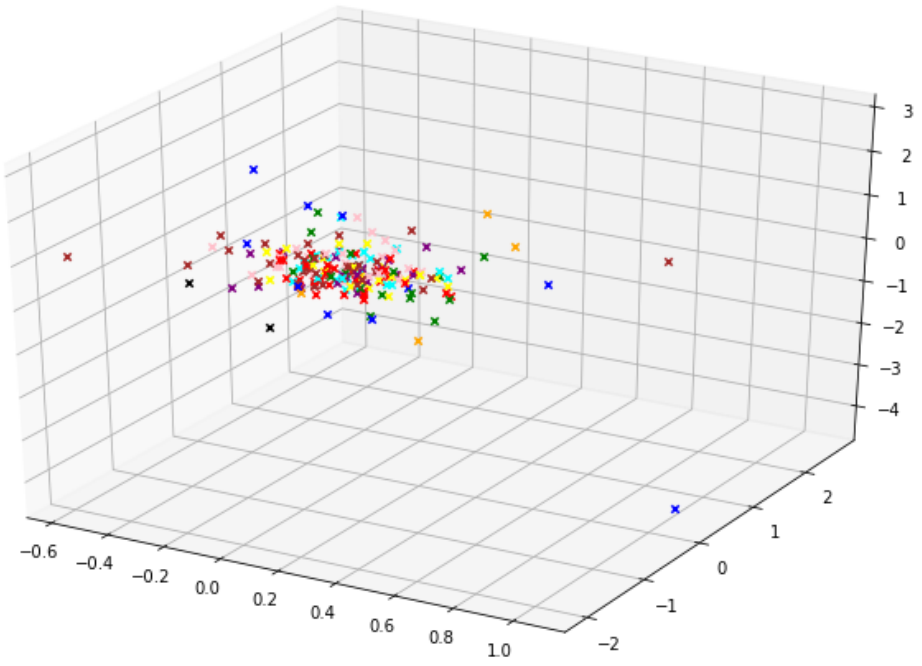


Finally, we look into reducing the dimensionality of our dataset to 3, and produce a 3-d visualization of the clustering.

The goal is to see if we can find patterns.

[21]:
```
# Dimensionality reduction to 3 dimensions

pca = decomposition.PCA(n_components = 3)
x_test_viz = StandardScaler().fit_transform(x_test)
x_test_viz = pca.fit_transform(x_test_viz)
```

[22]:
```
colors = ["blue","red","green","pink","purple","cyan",
          "yellow","orange","black","brown"]

fig = plt.figure(figsize=(12,8))
ax = fig.add_subplot(111, projection='3d')
for index, item in enumerate(x_test_viz[:200]):
    ax.scatter(item[0],
               item[1],
               item[2],
               marker="x",
               color = colors[label_predictions[index]])

plt.show()
```



Summary of observations:

The above visualizations highlight a few key behaviors:

1. KMeans is **quite able to distinguish bags** (clusters 0 and 7), **ankle boots** (clusters 2 and 8), and **sandals** and **sneakers** (clusters 4 and 5) from the rest of the dataset
2. KMeans **struggles identifying trousers from dresses**, pooling them in the same cluster (cluster 1)
3. KMeans **struggles identifying all top clothes from each other** (clusters 3, 6, and 9)

We obtained a **homogeneity score of c. 0.49**, a **v-measure score of c. 0.51**, and a **silhouette score of c. 0.16**.

As such, we can confidently say that the KMeans algorithm is able to distinguish larger granularities of clothes (top, bag, shoes) than the ones provided by the Fashion-MNIST dataset.

1.4 3. Implementation of GMM

Gaussian Mixture Models, or GMM, are a class of **unsupervised machine learning algorithms** that try to extract normally distributed subpopulations from an overall population. Like KMeans, GMM are a type of unsupervised machine learning.

A GMM is parametrized by two types of values: the **mixture component weights**, and the **component means and variances**.

Given k the number of component C_k of a GMM, each component has a mean μ_k and a variance σ_k^2 (for the univariate case, otherwise μ and σ are vectors). The mixture component wieghts are defined as ϕ_k for a component C_k, with the constraint that $\sum_{i=1}^{K} \phi_i = 1$ so that the total probability distribution normalizes to 1.

One-Dimensional Model

$$p(x) = \sum_{i=1}^{K} \phi_i . \mathcal{N}(x|\mu_i, \sigma_i^2)$$

$$\mathcal{N}(x|\mu_i, \sigma_i^2) = \frac{1}{\sigma_i . \sqrt{2.\pi}} . exp(\frac{-(x-\mu_i)^2}{2.\sigma_i^2})$$

$$\sum_{i=1}^{K} \phi_i = 1$$

Multi-Dimensional Model

$$p(x) = \sum_{i=1}^{K} \phi_i . \mathcal{N}(\vec{x}|\vec{\mu_i}, \sum_i)$$

$$\mathcal{N}(\vec{x}|\vec{\mu_i}, \sum_i) = \frac{1}{\sqrt{(2.\pi)^K . |\sum_i|}} . exp(-\frac{1}{2} . (\vec{x} - \vec{\mu_i})^T) . \sum_i^{-1} . (\vec{x} - \vec{\mu_i}))$$

$$\sum_{i=1}^{K} \phi_i = 1$$

Expectation Maximization or EM is the most common technique to estimate the mixture model's parameters when the number of components k is known. EM is an **iterative numerical technique for maximum likelihood estimation** and **is guaranteed to approach a local maximum** or saddle point.

Expectation Maximization As it is an iterative algorithm, it proceeds as follow:

1. `Initialization step 1: We randomly assign k samples without replacement from the dataset to k component mean estimates`
2. `Initialization step 2: We set all the component variance estimates to the sample variance (1)`
3. `Initialization step 3: We set all component distribution prior estimates to the uniform distribution (2)`
4. `E(xpectation)-Step: We calculate the probability that each data point is generated by a component C (3)`
5. `M(aximization)-Step: We maximize the expectations calculated in the E step with respect to the model parameters. The model parameters are then updated (4)`

6. The steps 4 and 5 are repeated until convergence

$$\sigma_k^2 = \sum_{i=1}^{N}(x_i - \bar{x})^2 \quad (1)$$

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\phi_i = \frac{1}{K} \quad (2)$$

$$\forall i, k \quad \hat{\gamma}_{i,k} = \frac{\hat{\phi}_k . \mathcal{N}(x_i|\hat{\mu}_k, \hat{\sigma}_k^2)}{\sum_{j=1}^{K} \hat{\phi}_j . \mathcal{N}(x_i|\hat{\mu}_j, \hat{\sigma}_j^2)} = p(C_k|x_i, \hat{\phi}, \hat{\mu}, \hat{\sigma}^2) \quad (3)$$

$$\forall k \quad \hat{\phi}_k = \sum_{i=1}^{N} \frac{\hat{\gamma}_{i,k}}{N} \quad (4.1)$$

$$\forall k \quad \hat{\mu}_k = \frac{\sum_{i=1}^{N} \hat{\gamma}_{i,k} . x_i}{\sum_{i=1}^{N} \hat{\gamma}_{i,k}} \quad (4.2)$$

$$\forall k \quad \hat{\sigma}_k^2 = \frac{\sum_{i=1}^{N} \hat{\gamma}_{i,k} . (x_i - \hat{\mu}_k)^2}{\sum_{i=1}^{N} \hat{\gamma}_{i,k}} \quad (4.3)$$

1.4.1 3.1. Simple Implementation From Scratch

We now have an idea about how the algorithm works. We can thus implement a simple version of it to see how it works.

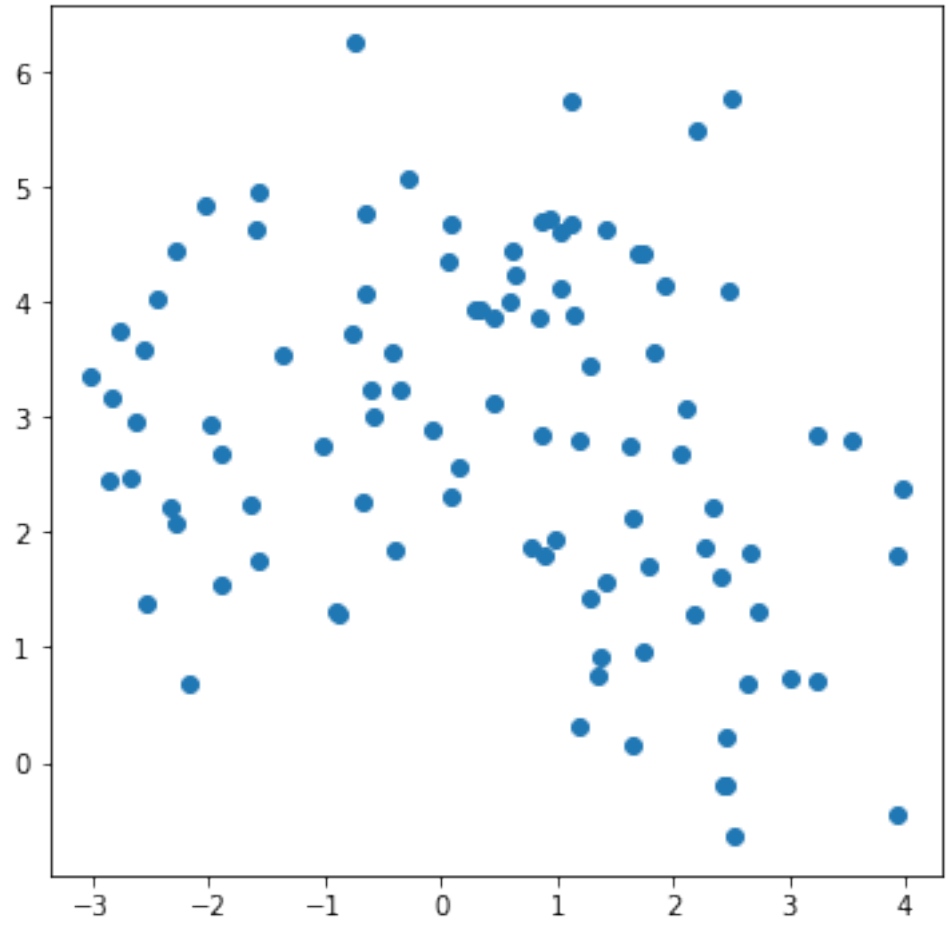
[23]:
```
class GMM():
    def __init__(self, nb_of_data_points, k=2):
        """
        Initializes the GMM class.
        """
        self.data = generate_dataset(nb_of_data_points)
        self.k = k
        self.colors = ["g","r","b","c","m","y"]
        self.phi = [1/self.k for _ in range(self.k)]
        self.cluster_names = list(range(10))
        self.clusters = []
```

```
    def multivariate_normal(self, X, mean_vector, covariance_matrix):
        """
        Formula of the multivariate normal distribution.
        """
        first_element = ((2*np.pi)**len(X) * np.linalg.
↪det(covariance_matrix))**(-1/2)
        exponential = np.exp(-1/2*np.dot(np.dot((X-mean_vector),
                                                np.linalg.
↪inv(covariance_matrix)).T,
                                            (X-mean_vector)))
        return first_element * exponential

    def fit(self, max_iterations=100):
        """
        Fits the model with a maximum number of iterations by default of 100.
        """
        # Splits the data in k subsets
        X = np.array_split(self.data, self.k)

        # Initializes the computation of the mean vector and the
        # covariance matrix
        self.mean_vector = [np.mean(x, axis=0) for x in X]
        self.covariance_matrices = [np.cov(x.T) for x in X]

        for iteration in range(max_iterations):
            if iteration %10 == 0: print(f"Epoch {iteration}")

            # E-Step

            # Calculates the gammas
            self.p = np.zeros((len(self.data), self.k)) #, dtype=object)

            # Calculates the phi matrix
            for n in range(len(self.data)):
                for k in range(self.k):
                    self.p[n][k] = self.phi[k] * self.multivariate_normal(
                        self.data[n],
                        self.mean_vector[k],
                        self.covariance_matrices[k]
                    )
                    self.p[n][k] /= sum([
                        self.phi[j]*self.multivariate_normal(
                            self.data[n],
                            self.mean_vector[j],
                            self.covariance_matrices[j]
                        )
                        for j in range(self.k)
```

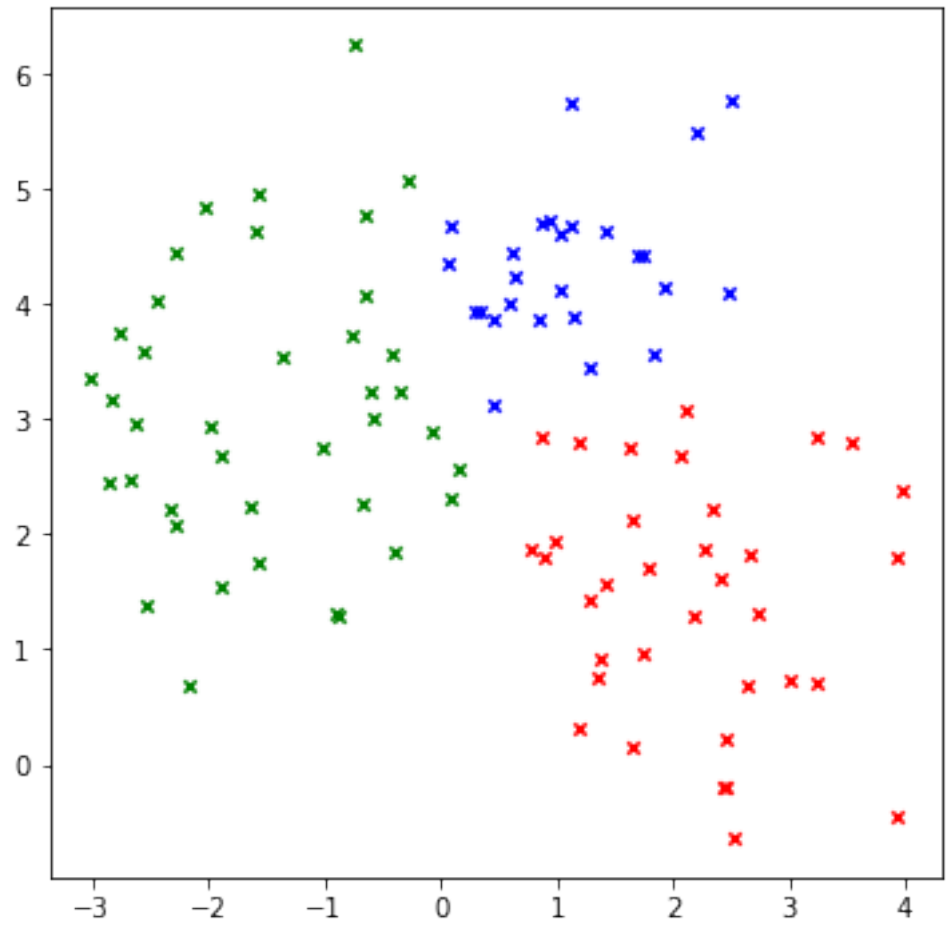
```
                ])
        N = np.sum(self.p, axis = 0)

        # M-Step

        # Initializes the mean vector as a zero vector
        self.mean_vector = np.zeros((self.k, len(self.data[0])))

        # Updates the mean vector
        for k in range(self.k):
            for n in range(len(self.data)):
                self.mean_vector[k] += self.p[n][k] * self.data[n]
        self.mean_vector = [1/N[k]*self.mean_vector[k]
                            for k in range(self.k)]

        # Initializes the list of the covariance matrices
        self.covariance_matrixes = [np.zeros((len(self.data[0]),
                                              len(self.data[0])))
                                    for k in range(self.k)]

        # Updates the covariance matrices
        for k in range(self.k):
            self.covariance_matrices[k] = np.cov(self.data.T,
                                                 aweights=(self.p[:,k]),
                                                 ddof=0)
        self.covariance_matrices = [1/N[k]*self.covariance_matrices[k]
                                    for k in range(self.k)]

        # Updates the phi list
        self.phi = [N[k]/len(self.data) for k in range(self.k)]

    def predict(self):
        """
        Predicts the data.
        """
        probabilities = []
        for n in range(len(self.data)):
            probabilities.append([self.multivariate_normal(
                self.data[n],
                self.mean_vector[k],
                self.covariance_matrices[k]) for k in range(self.k)])
        self.clusters = []
        for prob in probabilities:
            self.clusters.append(self.cluster_names[prob.index(max(prob))])
        return self.clusters

    def plot_data(self):
```

```
        """
        Plots the distribution of the data
        """
        # Declares the plot
        plt.figure(figsize=(6,6))

        # Plots the data without colors if the dataset was not iterated over.
        if self.clusters == []:
            plt.scatter(self.data[:,0],self.data[:,1])
        else:
            for index, datum in enumerate(self.data):
                plt.scatter(datum[0], datum[1],
                            marker='x',
                            color=self.colors[self.clusters[index]],
                            s=20,
                            linewidths=2)
        plt.show()
```

[24]:
```
model = GMM(100, 3)
model.plot_data()
model.fit()
model.predict()
model.plot_data()
```




```
Epoch 0
Epoch 10
Epoch 20
Epoch 30
Epoch 40
Epoch 50
Epoch 60
Epoch 70
Epoch 80
Epoch 90
```



We now see how a GMM algorithm works.

1.4.2 3.2. Implementation using SciKit-Learn

As the simple from scratch GMM works, and now that we have a better grasp on the different steps involved to fit the model, we can implement one using the SciKit-Learn library.

As such, we go back to our Fashion-MNIST dataset.

1. We set our hyperparameters:

[25]:
```
# Number of components we want to implement our model with
# We set it to match the number of classes in the Fashion-MNIST dataset
n_components = 10

# We indicate we want the model will not go over 100 iterations
max_iter = 100
```

2. We declare our model:

[26]:
```
model = GaussianMixture(n_components=n_components,
                        max_iter=max_iter,
                        verbose=0,
                        random_state=0).fit(x_train)
```

3. We predict labels on our test dataset using the resulting trained model:

[27]:
```
label_predictions = model.predict(x_test)
```

4. We output our resulting metrics:

[28]:
```
gmm_h_score = homogeneity_score(y_test, label_predictions)
gmm_v_score = v_measure_score(y_test, label_predictions)
gmm_s_score = silhouette_score(x_test, label_predictions)

print(gmm_h_score,gmm_v_score,gmm_s_score)
```

```
0.5480860311853569 0.5551891023091904 0.08121377084408062
```

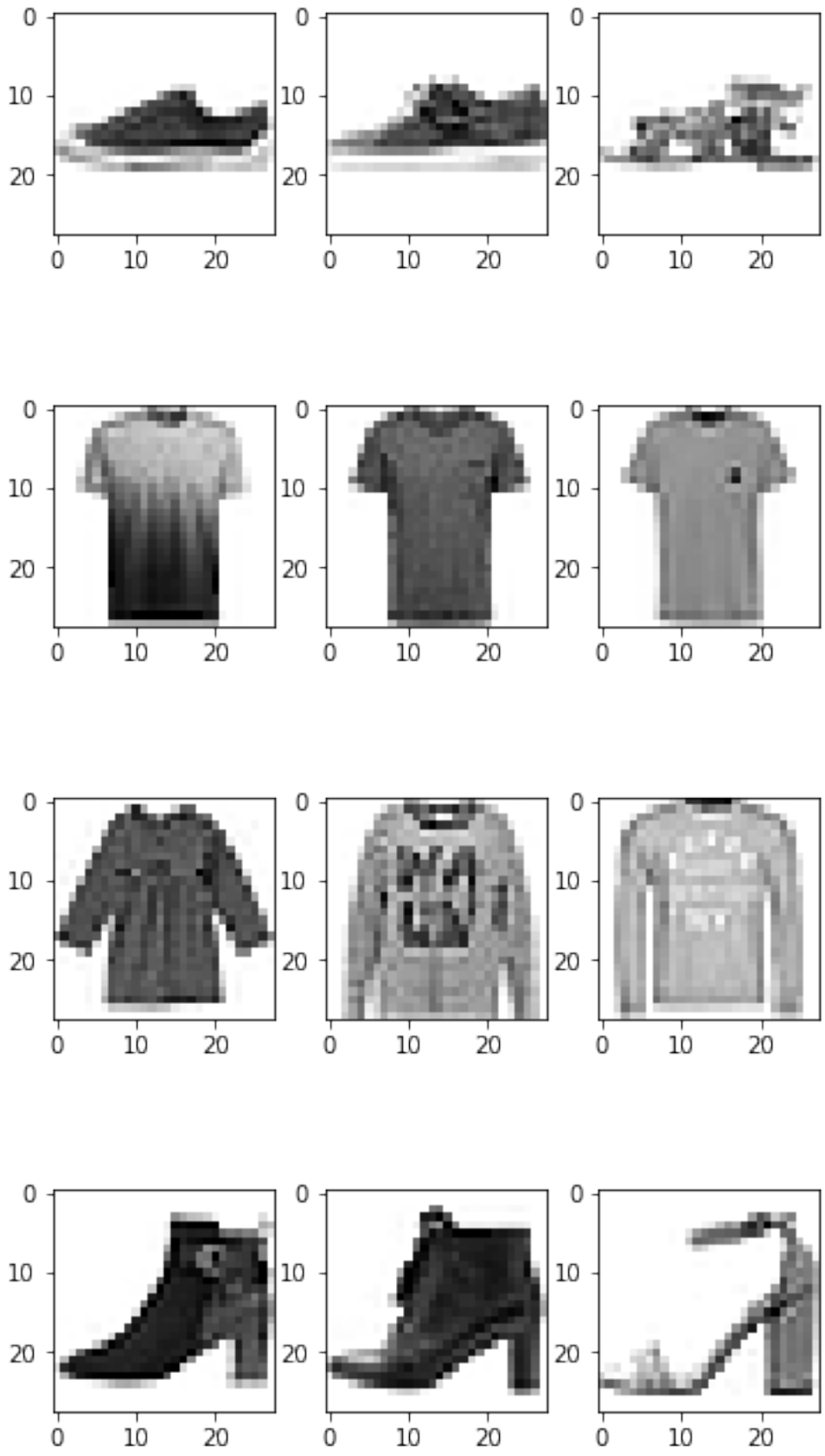
See our detailed observations below.

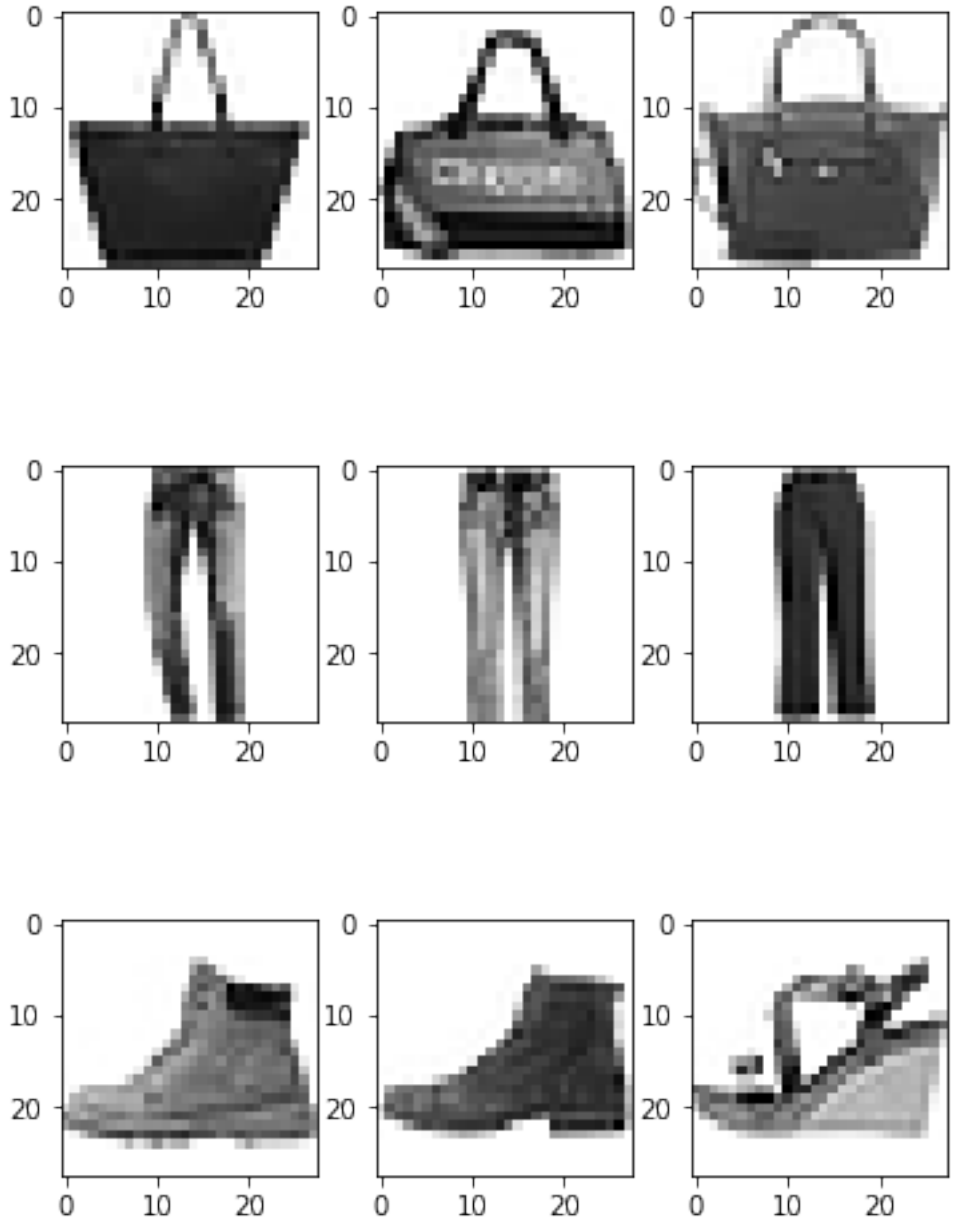
1.4.3 3.3. Observations

We visualize the resulting clustering on our test set.

Each row presents a trio of pictures that were assigned to the same cluster.

[29]:
```
label_number = 10

cluster_index= [[] for i in range(label_number)]
for i, label in enumerate(label_predictions):
    for n in range(label_number):
        if label == n: cluster_index[n].append(i)
        else: continue

for j, _ in enumerate(labelNames):
    plt.figure(figsize=(6,6));
    clust = j
    num = 10
    for i in range(1,4):
        plt.subplot(3, 3, i)
        plt.imshow(x_test_origin[cluster_index[clust][i+20]],
                   cmap = plt.cm.binary);

plt.show()
```

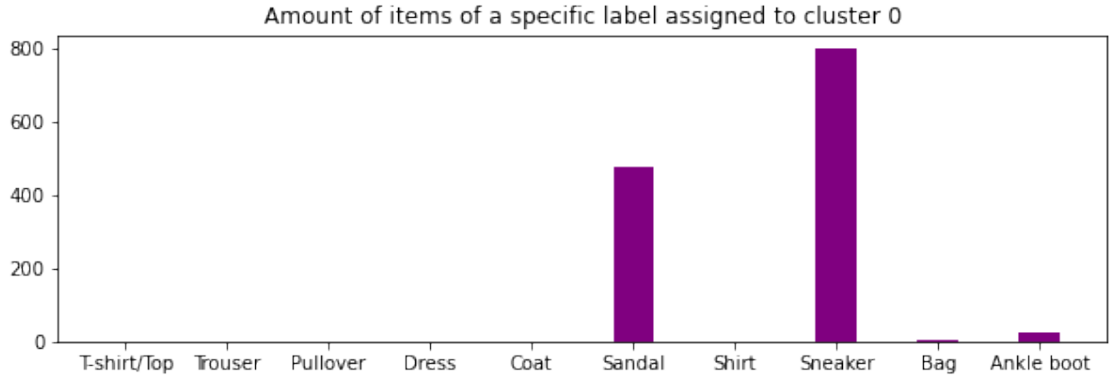


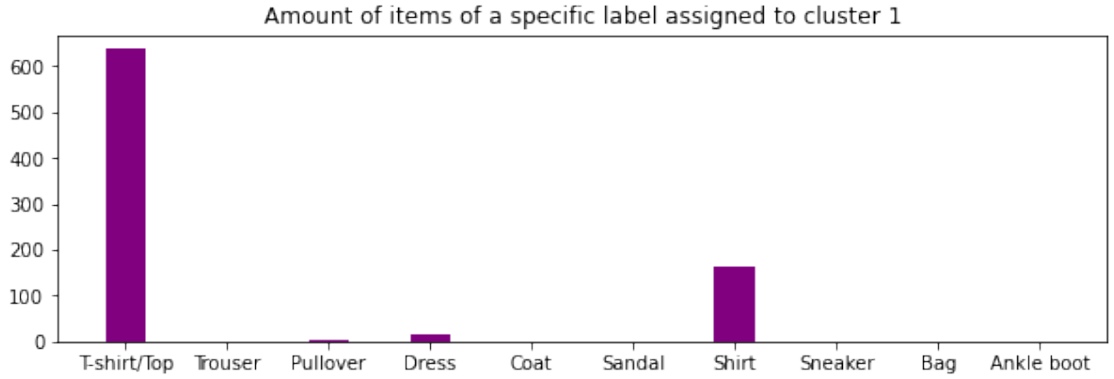


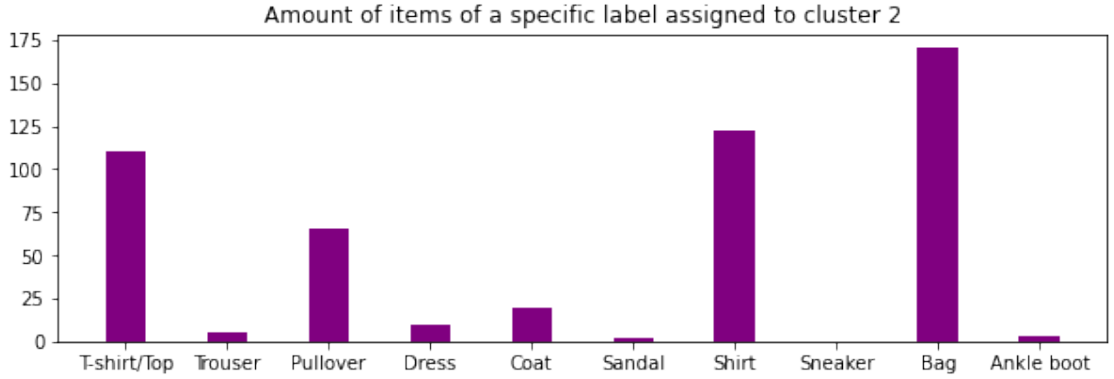
To provide a better view of what is happening, we visualize the repartition of the test set's items between each clusters.

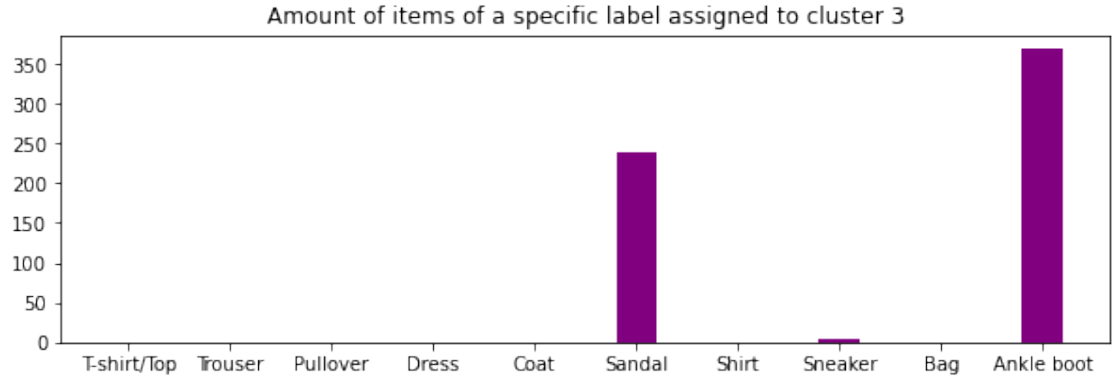
[30]:
```
clusters={}
for i in range(10):
    clusters[i]=[0]*10

for index, item in enumerate(label_predictions):
    clusters[item][y_test[index]] += 1
```

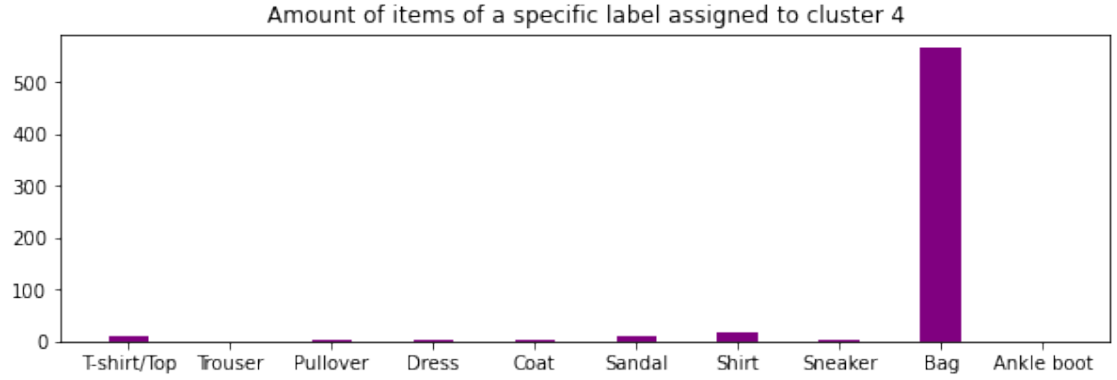
```python
for i in range(len(clusters)):
    fig = plt.figure(figsize=(10,3))
    plt.bar(labelNames, clusters[i], color="purple", width=0.4)
    plt.title(f"Amount of items of a specific label assigned to cluster {i}")
    plt.show()
```

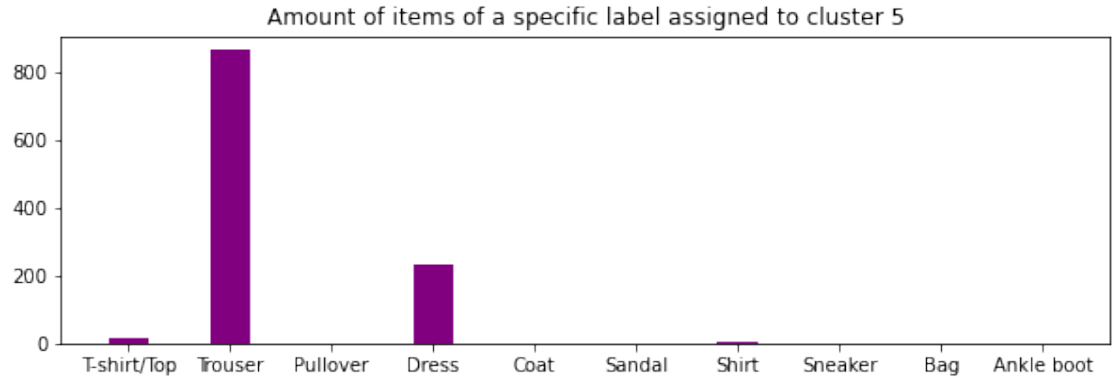


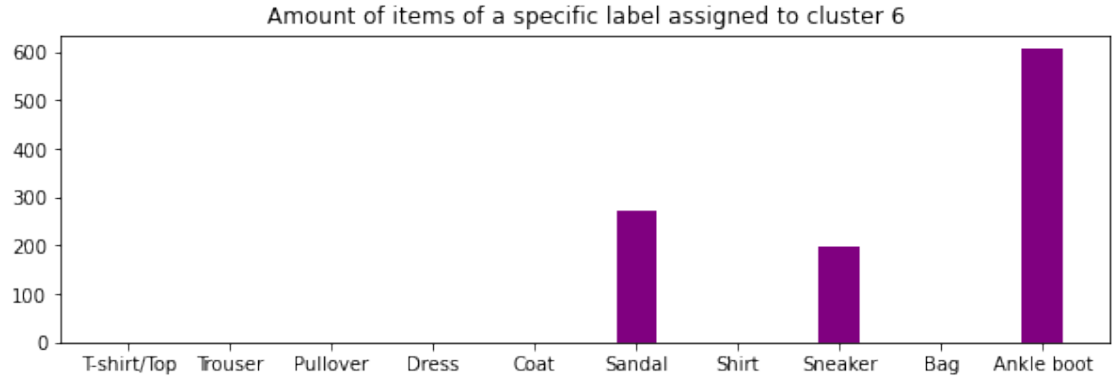


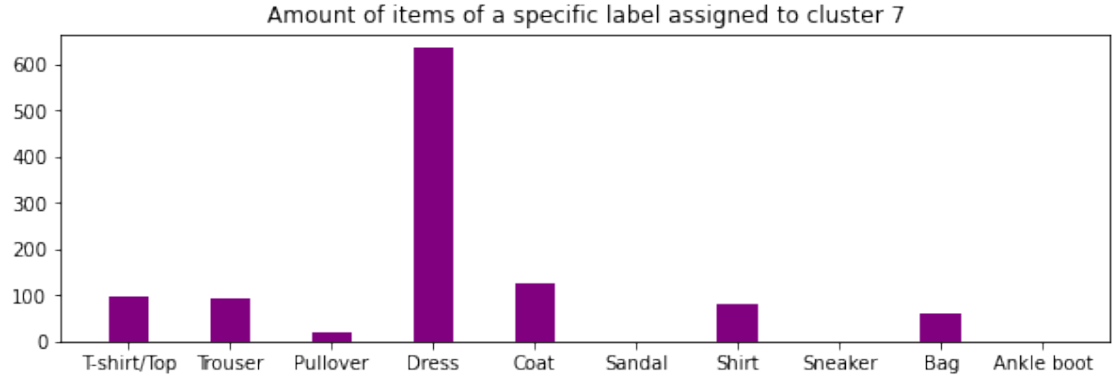


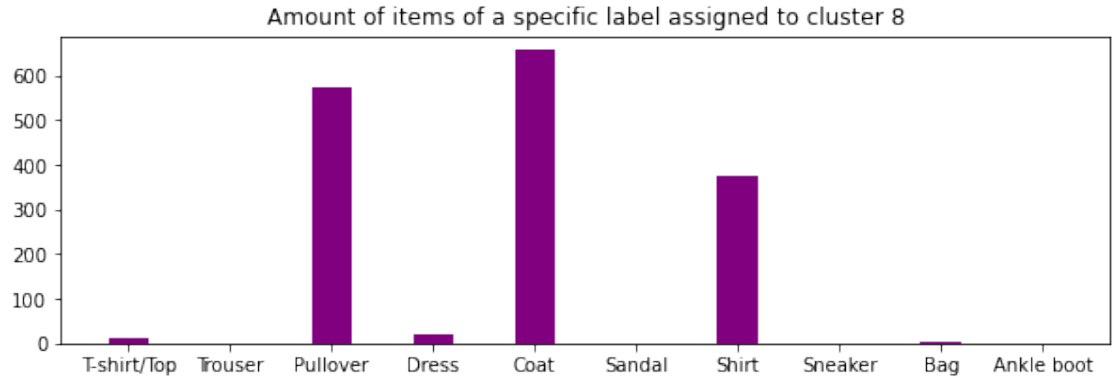

Amount of items of a specific label assigned to cluster 3
350
300
250
200
150
100
50
0
T-shirt/Top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot

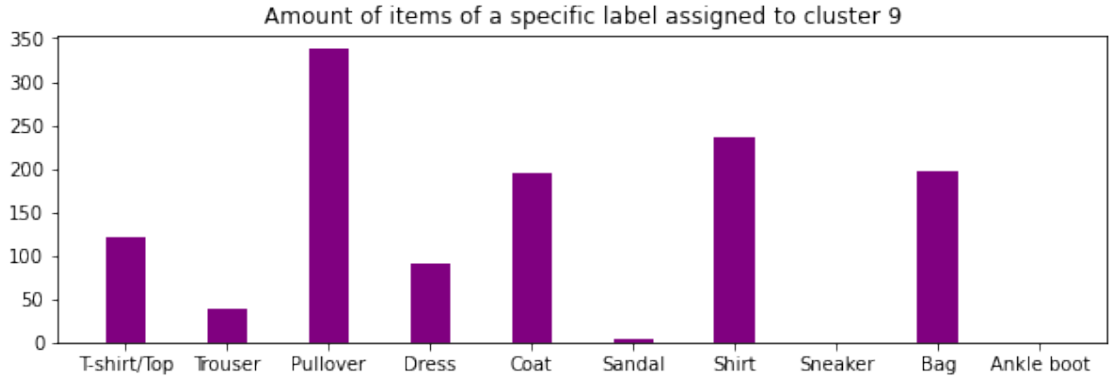

Amount of items of a specific label assigned to cluster 4
500
400
300
200
100
0
T-shirt/Top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot


Amount of items of a specific label assigned to cluster 5
800
600
400
200
0
T-shirt/Top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot


Amount of items of a specific label assigned to cluster 6
600
500
400
300
200
100
0
T-shirt/Top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot


Amount of items of a specific label assigned to cluster 7
600
500
400
300
200
100
0
T-shirt/Top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot


Amount of items of a specific label assigned to cluster 8
600
500
400
300
200
100
0
T-shirt/Top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot

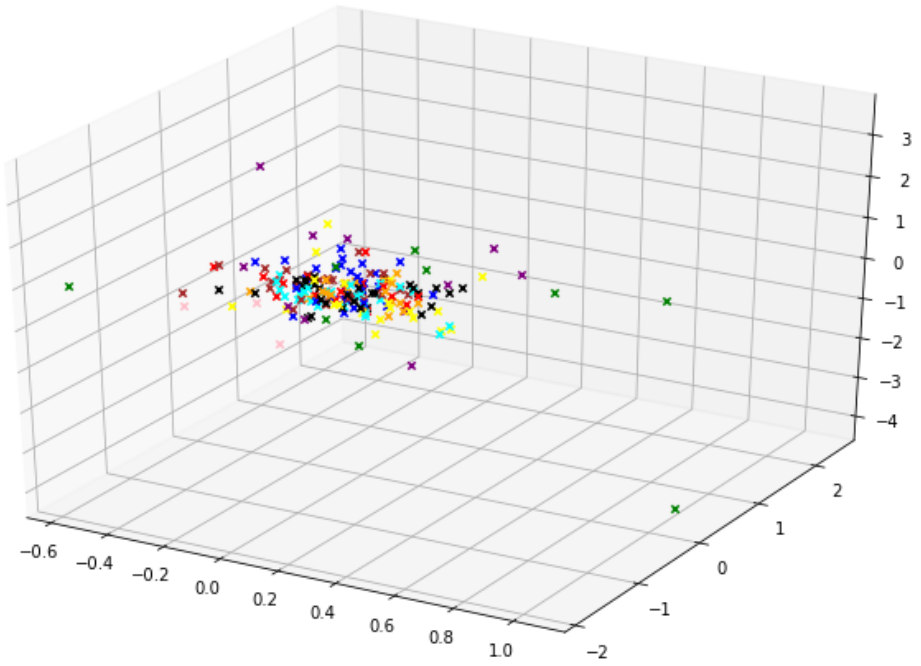


Finally, we look into reducing the dimensionality of our dataset to 3, and produce a 3-d visualization of the clustering.

The goal is to see if we can find patterns.

[31]:
```
pca = decomposition.PCA(n_components = 3)
x_test_viz = StandardScaler().fit_transform(x_test)
x_test_viz = pca.fit_transform(x_test_viz)
```

[32]:
```
colors = ["blue","red","green","pink","purple",
          "cyan","yellow","orange","black","brown"]

fig = plt.figure(figsize=(12,8))
ax = fig.add_subplot(111, projection='3d')
for index, item in enumerate(x_test_viz[:200]):
    ax.scatter(item[0],
               item[1],
               item[2],
               marker="x",
               color = colors[label_predictions[index]])

plt.show()
```



Summary of observations:

The above visualizations highlight a few key behaviors:

1. GMM is able to distinguish **ankle boots**, **sandals** and **sneakers** (clusters 0, 3, and 6) from the rest of the dataset
2. GMM seems **marginally worse than KMeans with regards to bags** as, while cluster 4 specifically focuses on bags, we see that a non-negligeable amount of bags were assigned to the clusters 2 and 9 with a lot of top clothes
3. Like KMeans, GMM **struggles identifying trousers from dresses** albeit marginally less so at first glance (cluster 5)
4. Like KMeans, GMM **struggles identifying all top clothes from each other** (clusters 1, 7, 8, and 9)

We obtained a **homogeneity score of c. 0.54**, a **v-measure score of c. 0.56**, and a **silhouette score of c. 0.08**.

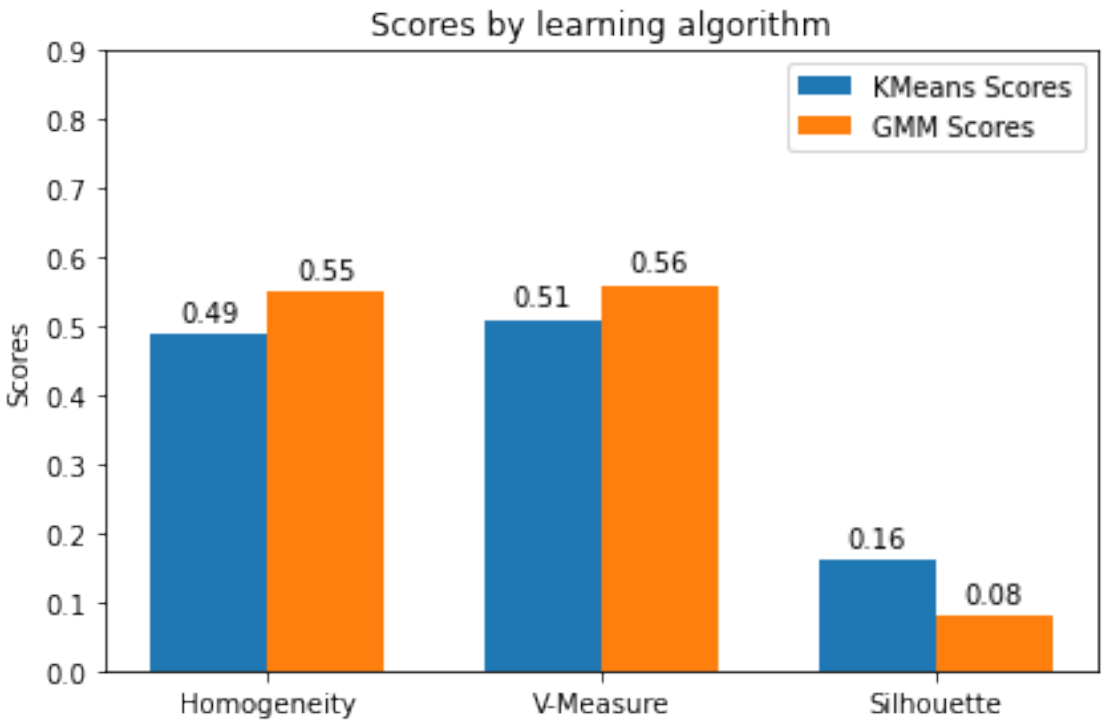
As such, we can confidently say that the GMM algorithm is able to distinguish larger granularities of clothes (top, bag, shoes) than the ones provided by the Fashion-MNIST dataset.

1.5 4. Comparing the results of both KMeans and GMM

1.5.1 4.1. Comparison

To compare the results between KMeans and GMM, we plot the different scores we retrieved from each models.

[42]:
```
gmm_scores = list(map(lambda x: round(x,2),
                      [gmm_h_score, gmm_v_score, gmm_s_score]))
kmeans_scores = list(map(lambda x: round(x,2),
                         [kmeans_h_score, kmeans_v_score, kmeans_s_score]))
labels = ['Homogeneity', 'V-Measure', 'Silhouette']

x = np.arange(0, len(labels))  # the label locations
y = np.arange(0, 1,0.1) # the y-axis labels
width = 0.35  # the width of the bars

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, kmeans_scores, width, label='KMeans Scores')
rects2 = ax.bar(x + width/2, gmm_scores, width, label='GMM Scores')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Scores')
ax.set_title('Scores by learning algorithm')
ax.set_xticks(x)
ax.set_yticks(y)
ax.set_xticklabels(labels)
ax.legend()

def autolabel(rects):
    """
    Attach a text label above each bar in *rects*, displaying its height.
    """
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

autolabel(rects1)
autolabel(rects2)

fig.tight_layout()

plt.show()
```



Conclusion:

GMM marginally improves improves clustering performance on the Fashion-MNIST dataset compared to the KMeans algorithm.

However, we observe that **we have a lower silhouette score with the GMM algorithm**, indicating that the algorithm tends to lead to an increasing amount of overlapping clusters.

1.5.2 4.2. Other Explorations: Trying out MaxPooling and a Sobel Filter instead of PCA

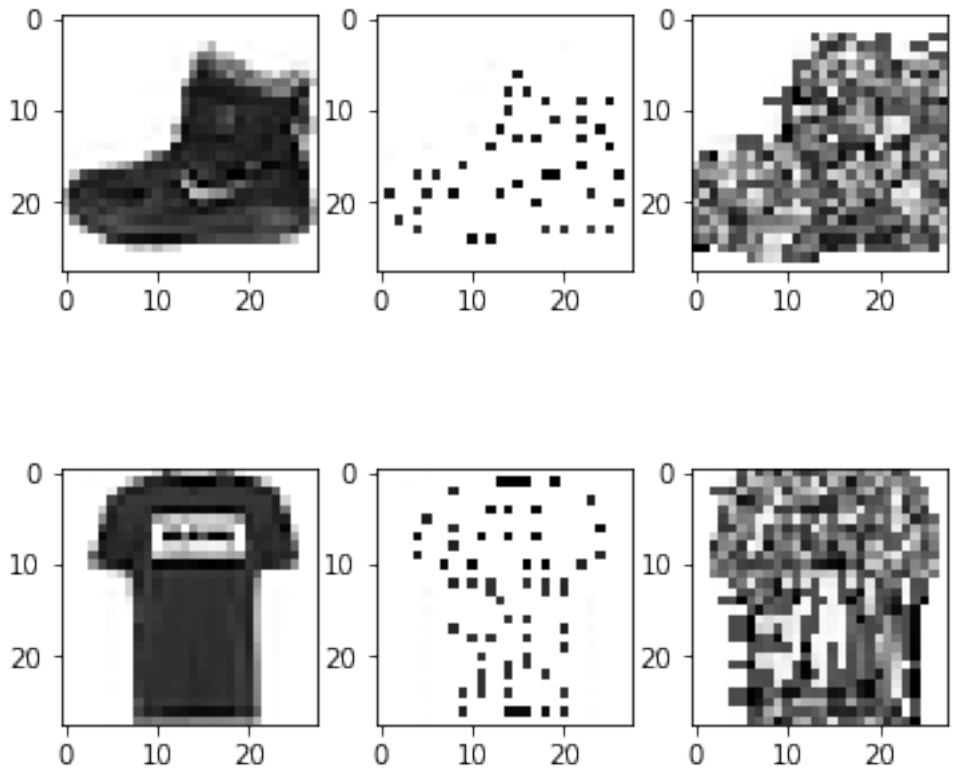
As we have used PCA in order to reduce the dimensionality of the dataset, we can also test out other ways to pre-process the Fashion-MNIST dataset, including:

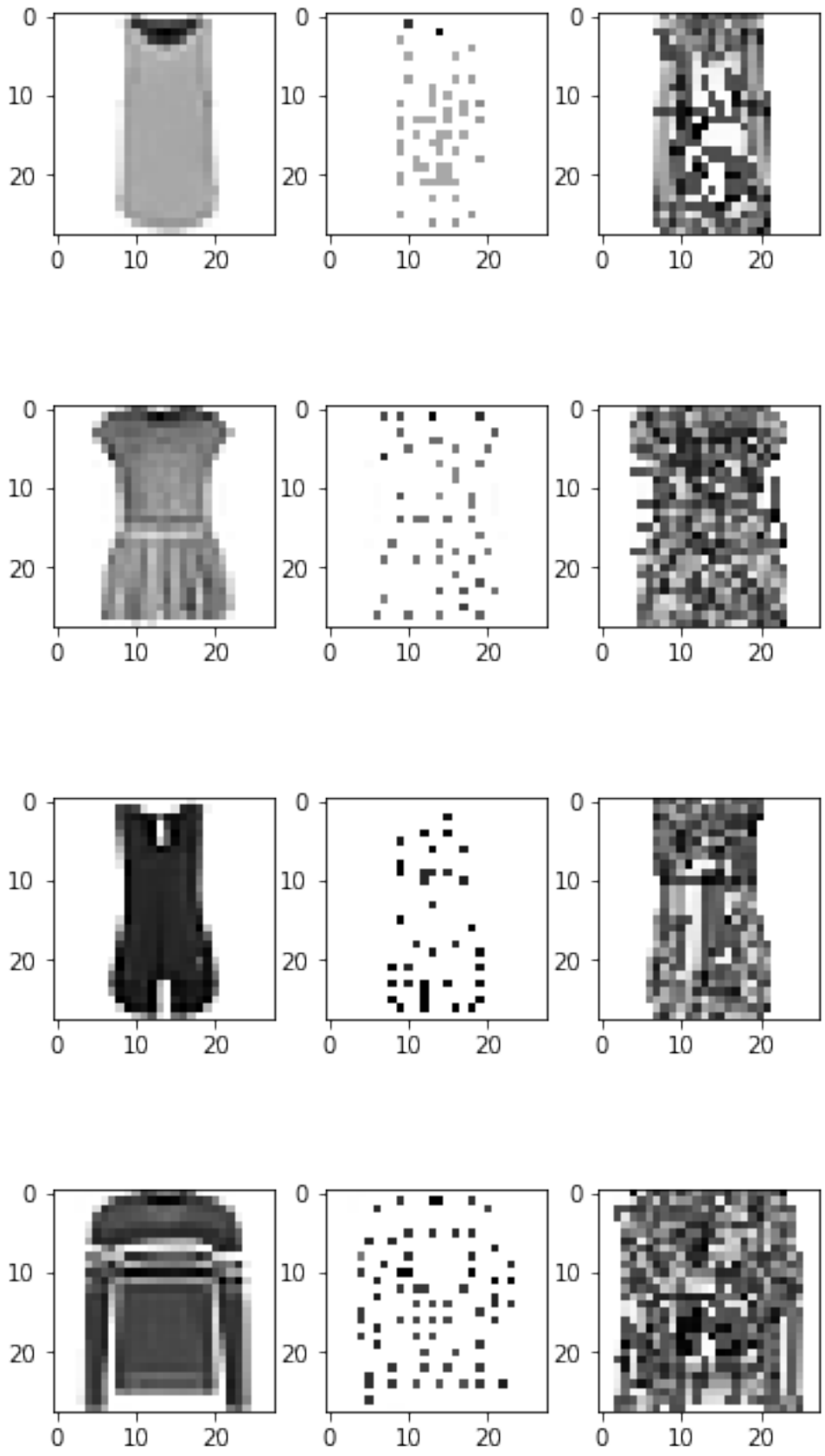
- **MaxPooling**:The goal is to calculate the maximum value for each patch of the feature map.
- **Sobel Filtering**: Sobel filtering aims at finding edges on a picture.

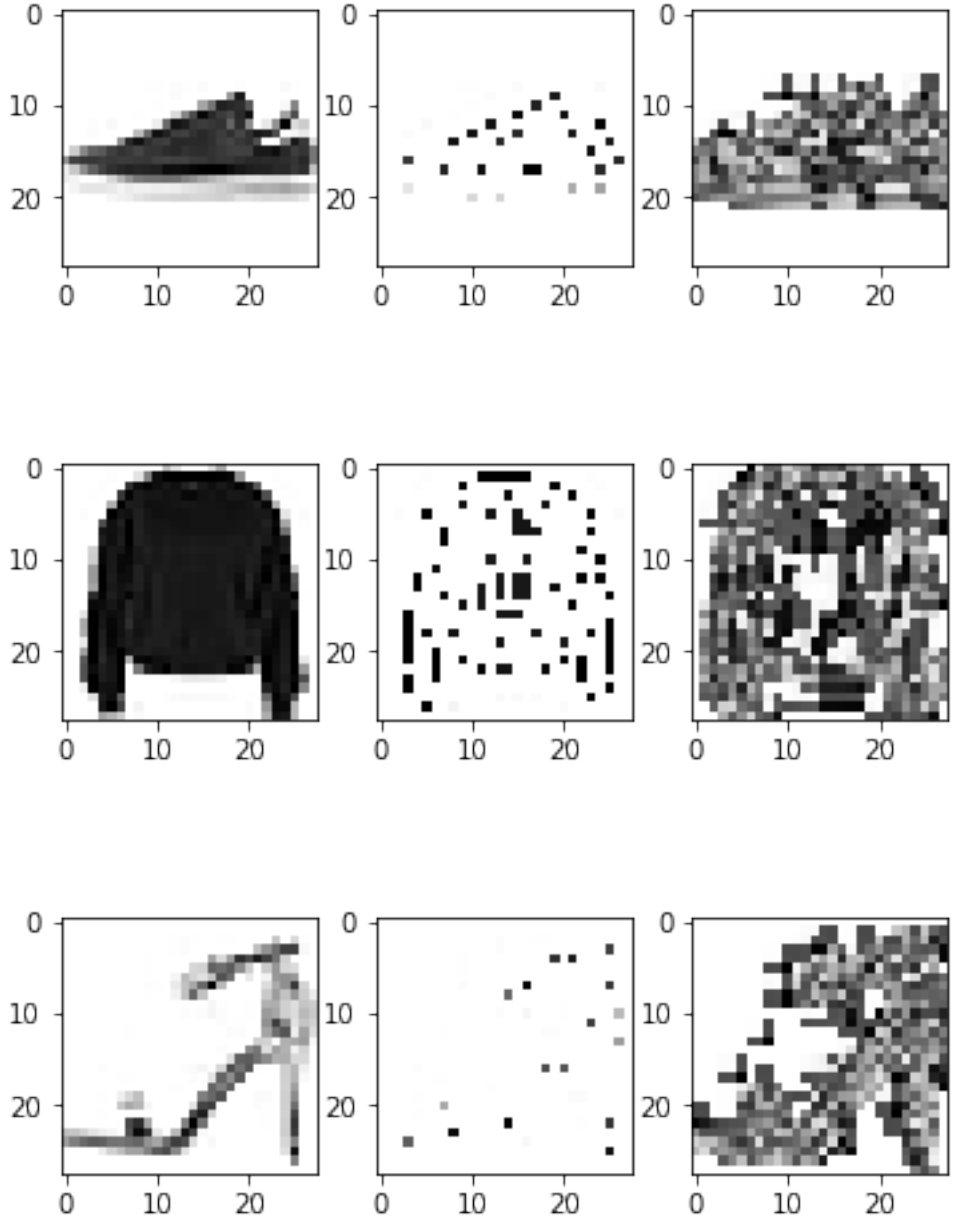
The goal here is to go back to the raw training and test sets and apply those two different operations instead of our previous preprocessing and see what results we can yield.

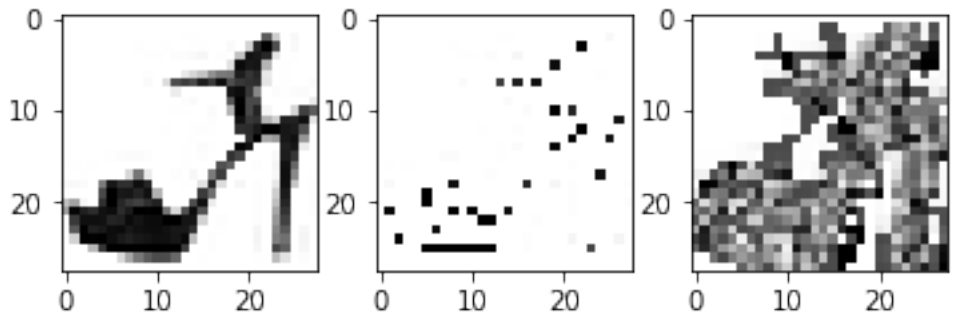
[44]:
```
def max_pool(picture):
    """
    (3,3) kernel max-pooling of a picture with stride and padding 1,
    with normalization
    """
    return picture*(picture == maximum_filter(picture,
```

```
                                                footprint=np.ones((3,3))))/255.

def sobel_filter(picture):
    """
    Sobel filtering of a picture along the x and y axes, with normalization
    """
    dx = ndimage.sobel(pic, 0)  # horizontal derivative
    dy = ndimage.sobel(pic, 1)  # vertical derivative
    mag = np.hypot(dx, dy)  # magnitude
    mag *= 255.0 / np.max(mag)  # normalize
    return np.asarray(mag, dtype=np.float32)/255.

for i in range(10):
    pic = x_train_origin[i]
    sobel_pic = sobel_filter(pic)
    maxpooled_pic = max_pool(pic)
    plt.figure(figsize=(6,6))
    plt.subplot(3, 3, 1); plt.imshow(pic, cmap = plt.cm.binary)
    plt.subplot(3, 3, 2); plt.imshow(maxpooled_pic, cmap = plt.cm.binary)
    plt.subplot(3, 3, 3); plt.imshow(sobel_pic, cmap = plt.cm.binary)

plt.show()
```









1. Training and test sets preprocessing using the sobel and maxpool functions

[35]:
```
def sobel_fMNIST(dataset):
    return np.array(list(map(sobel_filter, dataset))).reshape(len(dataset), 784)

def maxpool_fMNIST(dataset):
    return np.array(list(map(max_pool, dataset))).reshape(len(dataset), 784)

x_train_sobel = sobel_fMNIST(x_train_origin)
x_train_maxpool = maxpool_fMNIST(x_train_origin)
x_test_sobel = sobel_fMNIST(x_test_origin)
x_test_maxpool = maxpool_fMNIST(x_test_origin)

print(x_train_sobel.shape, x_train_maxpool.shape,
      x_test_sobel.shape, x_test_maxpool.shape)
```

```
(60000, 784) (60000, 784) (10000, 784) (10000, 784)
```

2. Declaring our models

[36]:
```
def run_kmeans(x_train, x_test, y_test):
    model = KMeans(n_clusters=10, random_state=0).fit(x_train)
    label_predictions = model.predict(x_test)
    kmeans_h_score = homogeneity_score(y_test, label_predictions)
    kmeans_v_score = v_measure_score(y_test, label_predictions)
    print(kmeans_h_score, kmeans_v_score)
```

[37]:
```
def run_gmm(x_train, x_test, y_test):
    model = GaussianMixture(n_components=10, random_state=0).fit(x_train)
    label_predictions = model.predict(x_test)
    kmeans_h_score = homogeneity_score(y_test, label_predictions)
    kmeans_v_score = v_measure_score(y_test, label_predictions)
    print(kmeans_h_score, kmeans_v_score)
```

3. Running our models

[38]:
```
run_gmm(x_train_maxpool, x_test_maxpool, y_test)
```

```
0.37893416779996025 0.42943222404667425
```

[39]:
```
run_kmeans(x_train_maxpool, x_test_maxpool, y_test)
```

```
0.367940095375142 0.3896772340322883
```

[40]:
```
run_gmm(x_train_sobel, x_test_sobel, y_test)
```

```
0.0 0.0
```

[41]:
```
run_kmeans(x_train_sobel, x_test_sobel, y_test)
```

```
0.0 0.0
```

Summary of observations:

The results are definitely worse.

Maxpooling instead of our previous preprocessing shows lower results for the homogeneity and v measure scores for both the KMeans and GMM implementations.

Meanwhile, training KMeans and GMM on the sobel filtered data yields scores of 0. It indicates either that the filtering erased some of the differentiable features between each items in the set, or that something is wrong with the sobel implementation.

1.6 5. Exploring the parameters of GMM

To explore the impact of different hyperparameters on the GMM algorithm, we will implement our own version of a gridsearch function to iterate over the following hyperparameters:

Hyperparameter	Variables
n_components	10 to 40 components by steps of 2 (i.e. 10, 12, 14, ..., 40)
covariance types*	full, tied, diagonal and spherical
Initialization	kmeans, random

* '**full**' indicates that each component has its own general covariance matrix, '**tied**' that all components share the same general covariance matrix, '**diag**' that each component has its own diagonal covariance matrix, and '**spherical**' that each component has its own single variance.

Such a gridsearch will yield **128 different models** that we will be able to compare using the metrics we have introduced and used in the previous parts: homogeneity, v-measure, and silhouette.

[34]:
```
class GridSearch():
    def __init__(self, hyperparameters):
        """
        Initializes our gridsearch object.
        """
        self.hyperparameters = hyperparameters
        self.training_results = {}
```

```
    def fit(self, x_train, y_train):
        """
        Fits a GMM algorithm given the set of hyperparameters used during
        initializations.
        """
        model_number = 0
        for component in hyperparameters["n_components"]:
            for covariance in hyperparameters["covariance_type"]:
                for param in hyperparameters["init_params"]:
                    model_number += 1
                    model = GaussianMixture(n_components=component,
                                            covariance_type=covariance,
                                            init_params=param,
                                            random_state=0)
                    model.fit(x_train)
                    h_score, v_score, sil_score = self.scoring(model,
                                                               x_train,
                                                               y_train)
                    print(f"GMM #{model_number} trained with {component} " +
                          f"components, cov. type {covariance}, " +
                          f"and a {param} parameter initialization, which " +
                          "yielded homogeneity, v_measure and silhouette " +
                          f"scores of {h_score}, {v_score} and " +
                          f"{sil_score} resp.\n")
                    self.training_results[model_number] = {
                        "n_components":component,
                        "covariance_type":covariance,
                        "model": model,
                        "homogeneity_score": h_score,
                        "v_measure_score": v_score}

        return self.training_results

    def scoring(self, model, X, y):
        """
        Scoring function that returns the homogeneity, v-measure and
        silhouette score
        of a given model
        """
        y_pred = model.predict(X)
        v_score = v_measure_score(y, y_pred)
        h_score = homogeneity_score(y, y_pred)
        sil_score = silhouette_score(X, y_pred, metric='euclidean')
        return h_score, v_score, sil_score

    def predict(self, model_number, x_test, y_test):
```

```
        """
        Predicts the results on a dataset (here test set) given a model number.
        """
        model = self.training_results[model_number]["model"]
        h_score, v_score, sil_score = self.scoring(model, x_test, y_test)
        print(f"GMM #{model_number} yielded homogeneity, " +
              "v_measure and silhouette scores "+
              f"of {h_score}, {v_score} and {sil_score} resp. on the test set")
        return model.predict(x_test)
```

[35]:
```
hyperparameters = {"n_components":range(10, 41, 2),
                   "covariance_type":["full", "tied", "diag", "spherical"],
                   "init_params":["kmeans", "random"]}

grid_search = GridSearch(hyperparameters)
```

Please find the summary of the best models resulting from this gridsearch at the end of this document.

[36]:
```
results = grid_search.fit(x_train, y_train)
```

```
GMM #1 trained with 10 components, cov. type full, and a kmeans parameter
initialization, which yielded homogeneity, v_measure and silhouette scores of
0.5511592078344985, 0.5584791747996576 and 0.0815661328826588 resp.

GMM #2 trained with 10 components, cov. type full, and a random parameter
initialization, which yielded homogeneity, v_measure and silhouette scores of
0.508523389964617, 0.5338744662137322 and 0.09199440509517363 resp.

GMM #3 trained with 10 components, cov. type tied, and a kmeans parameter
initialization, which yielded homogeneity, v_measure and silhouette scores of
0.5758313952528333, 0.5902039215949794 and 0.0930896303712398 resp.

GMM #4 trained with 10 components, cov. type tied, and a random parameter
initialization, which yielded homogeneity, v_measure and silhouette scores of
0.051513090460491066, 0.051757989351417316 and -0.03797984175168568 resp.

GMM #5 trained with 10 components, cov. type diag, and a kmeans parameter
initialization, which yielded homogeneity, v_measure and silhouette scores of
0.4213694087615115, 0.43551659399550924 and 0.03393882372839429 resp.

GMM #6 trained with 10 components, cov. type diag, and a random parameter
initialization, which yielded homogeneity, v_measure and silhouette scores of
0.41303993288019947, 0.42615289227735254 and 0.020101545059343955 resp.

GMM #7 trained with 10 components, cov. type spherical, and a kmeans parameter
initialization, which yielded homogeneity, v_measure and silhouette scores of
0.48136451998642066, 0.4858943289426888 and 0.12135738304840013 resp.
```

GMM #8 trained with 10 components, cov. type spherical, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.478890509938528, 0.48314854023274045 and 0.10620416490574947 resp.

GMM #9 trained with 12 components, cov. type full, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5598478169065954, 0.5506227893164184 and 0.06436799908401719 resp.

GMM #10 trained with 12 components, cov. type full, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5137931469621577, 0.5180325680453681 and 0.07718537773189406 resp.

GMM #11 trained with 12 components, cov. type tied, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6238188089374215, 0.6118611577286936 and 0.10172971270028791 resp.

GMM #12 trained with 12 components, cov. type tied, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.05796291925362067, 0.056403296651715146 and -0.04374872189168907 resp.

GMM #13 trained with 12 components, cov. type diag, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.4617083145666439, 0.45622505363055343 and 0.027126293399575268 resp.

GMM #14 trained with 12 components, cov. type diag, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.4337502212248077, 0.4275391153628751 and -0.001333047574986653 resp.

GMM #15 trained with 12 components, cov. type spherical, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5231689279269707, 0.5078025987916949 and 0.11612892597614222 resp.

GMM #16 trained with 12 components, cov. type spherical, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5369935328313277, 0.5199714544495825 and 0.1071662561267613 resp.

GMM #17 trained with 14 components, cov. type full, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5554245729511919, 0.526753868025694 and 0.05572198427191316 resp.

GMM #18 trained with 14 components, cov. type full, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5257863850966122, 0.5213323919914687 and 0.0805291218971453 resp.

GMM #19 trained with 14 components, cov. type tied, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6337920678036757, 0.6082724394510765 and 0.10095282529515394 resp.

GMM #20 trained with 14 components, cov. type tied, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.041865740520193255, 0.0392748827273594 and -0.05021875279704544 resp.

GMM #21 trained with 14 components, cov. type diag, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.49562733904592815, 0.47032173818389844 and 0.023619137831341657 resp.

GMM #22 trained with 14 components, cov. type diag, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.4763069925421629, 0.4547188524358954 and 0.00942131063663208 resp.

GMM #23 trained with 14 components, cov. type spherical, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5541076066703363, 0.521365600607185 and 0.11890792928401955 resp.

GMM #24 trained with 14 components, cov. type spherical, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5611104192595245, 0.5271920674578544 and 0.11724512705565131 resp.

GMM #25 trained with 16 components, cov. type full, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5664702196714098, 0.5278397477211413 and 0.05288143281715352 resp.

GMM #26 trained with 16 components, cov. type full, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5591624574070863, 0.5302617055164749 and 0.0682044896434874 resp.

GMM #27 trained with 16 components, cov. type tied, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6366207008021804, 0.5988740904698301 and 0.10740203071379664 resp.

GMM #28 trained with 16 components, cov. type tied, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.07472272879080241, 0.06840827700555532 and -0.050544553048197 resp.

GMM #29 trained with 16 components, cov. type diag, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5000265255871127, 0.4608059451480909 and 0.022943280846533683 resp.

GMM #30 trained with 16 components, cov. type diag, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.49075427739652916, 0.4590269713522486 and 0.004073110773516253 resp.

GMM #31 trained with 16 components, cov. type spherical, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5618599662656887, 0.5187228102902145 and 0.11631831610229047 resp.

GMM #32 trained with 16 components, cov. type spherical, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5771839225564764, 0.5270748963748376 and 0.10148761201241978 resp.

GMM #33 trained with 18 components, cov. type full, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5892548285648327, 0.5391130902980763 and 0.05115474264134474 resp.

GMM #34 trained with 18 components, cov. type full, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5459052223766789, 0.5314999803385276 and 0.0674344220785279 resp.

GMM #35 trained with 18 components, cov. type tied, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6368209806583783, 0.5885126050141969 and 0.09290388003867105 resp.

GMM #36 trained with 18 components, cov. type tied, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.0767213681185098, 0.06837979916969525 and -0.07252056520309504 resp.

GMM #37 trained with 18 components, cov. type diag, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5278972412351576, 0.47811236024086007 and 0.03116439120253383 resp.

GMM #38 trained with 18 components, cov. type diag, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.466838361505079, 0.42617628832486365 and -0.027702707367792196 resp.

GMM #39 trained with 18 components, cov. type spherical, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5940584697782718, 0.5330566534933096 and 0.10828825285894891 resp.

GMM #40 trained with 18 components, cov. type spherical, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5851476739980825, 0.5228970625057223 and 0.10027891010757325 resp.

GMM #41 trained with 20 components, cov. type full, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5904178344426593, 0.5312789590369091 and 0.03294134578626384 resp.

GMM #42 trained with 20 components, cov. type full, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5691849522781137, 0.5381565629410758 and 0.05549079935371116 resp.

GMM #43 trained with 20 components, cov. type tied, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6385730931155998, 0.5847829355947112 and 0.08586260720662879 resp.

GMM #44 trained with 20 components, cov. type tied, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.07610277622552529, 0.06675696213806105 and -0.059639935265385365 resp.

GMM #45 trained with 20 components, cov. type diag, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5276926551691309, 0.46953484537446366 and -0.000292218653392915 resp.

GMM #46 trained with 20 components, cov. type diag, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5205925288234193, 0.4639241233658952 and 0.0016305804268343517 resp.

GMM #47 trained with 20 components, cov. type spherical, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5974025049904225, 0.5222078918757881 and 0.09101937183385013 resp.

GMM #48 trained with 20 components, cov. type spherical, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5956675078412026, 0.5245674766415919 and 0.1047068604028664 resp.

GMM #49 trained with 22 components, cov. type full, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6118334672381527, 0.5398327536263086 and 0.02399766866200203 resp.

GMM #50 trained with 22 components, cov. type full, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5848551811081396, 0.5395751874323149 and 0.044255652285948986 resp.

GMM #51 trained with 22 components, cov. type tied, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6273175830935661, 0.5548174765993673 and 0.07704857636482493 resp.

GMM #52 trained with 22 components, cov. type tied, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.07049274809072457, 0.060747255553047656 and -0.05756851246429023 resp.

GMM #53 trained with 22 components, cov. type diag, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5577267369163565, 0.4860539065486018 and 0.001744670428826735 resp.

GMM #54 trained with 22 components, cov. type diag, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5235962385175945, 0.4580962173120006 and -0.004562701010438045 resp.

GMM #55 trained with 22 components, cov. type spherical, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6057050645779888, 0.5208610856765662 and 0.09072668787766638 resp.

GMM #56 trained with 22 components, cov. type spherical, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6028976549055255, 0.5189862326430035 and 0.08629150304184173 resp.

GMM #57 trained with 24 components, cov. type full, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6157528317248764, 0.5358049384358518 and 0.026220682911336845 resp.

GMM #58 trained with 24 components, cov. type full, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5829936729034619, 0.5366512157657013 and 0.0588177245582944 resp.

GMM #59 trained with 24 components, cov. type tied, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6412752705331654, 0.5622227365308801 and 0.07848970029414071 resp.

GMM #60 trained with 24 components, cov. type tied, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.08790143982889982, 0.07443195598673912 and -0.07621144147829091 resp.

GMM #61 trained with 24 components, cov. type diag, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5716231107916109, 0.49131619337097926 and 0.009778533662292154 resp.

GMM #62 trained with 24 components, cov. type diag, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.569469456442973, 0.48825443819012854 and 0.0033005834288409503 resp.

GMM #63 trained with 24 components, cov. type spherical, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6102378688593838, 0.5176928201393303 and 0.0931571652683062 resp.

GMM #64 trained with 24 components, cov. type spherical, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6076918545559725, 0.5155535654967621 and 0.08548981946058337 resp.

GMM #65 trained with 26 components, cov. type full, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6219812346464375, 0.534571329936454 and 0.016862561752960956 resp.

GMM #66 trained with 26 components, cov. type full, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5756564713922728, 0.523623738412341 and 0.03797730919768536 resp.

GMM #67 trained with 26 components, cov. type tied, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6453697198223844, 0.5564801885829749 and 0.06590269652935883 resp.

GMM #68 trained with 26 components, cov. type tied, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.07869854362181289, 0.06599765279994037 and -0.09237339746427718 resp.

GMM #69 trained with 26 components, cov. type diag, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5815446219613071, 0.4917411173610142 and 0.0021794125122238514 resp.

GMM #70 trained with 26 components, cov. type diag, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5504676787998324, 0.4675338817880206 and -0.008783507413694871 resp.

GMM #71 trained with 26 components, cov. type spherical, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6164129195472114, 0.5156386382203226 and 0.08463370415939113 resp.

GMM #72 trained with 26 components, cov. type spherical, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6185742728458935, 0.5176401063882825 and 0.082253909979846 resp.

GMM #73 trained with 28 components, cov. type full, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6285307056197539, 0.5328276691393291 and 0.013819703188571184 resp.

GMM #74 trained with 28 components, cov. type full, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5850833750319282, 0.5335366092582466 and 0.04995163499433112 resp.

GMM #75 trained with 28 components, cov. type tied, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.638347240776981, 0.547321075331773 and 0.07070281945073202 resp.

GMM #76 trained with 28 components, cov. type tied, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.09894255795218472, 0.08173707705315729 and -0.06524970441158875 resp.

GMM #77 trained with 28 components, cov. type diag, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5826772968456823, 0.4857922697561558 and 0.0024134141399392317 resp.

GMM #78 trained with 28 components, cov. type diag, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5701768132682518, 0.4768620686082618 and 0.003735806325893379 resp.

GMM #79 trained with 28 components, cov. type spherical, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6211174507498037, 0.5120306266375902 and 0.07812546483093392 resp.

GMM #80 trained with 28 components, cov. type spherical, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.625509712105183, 0.5162666602751257 and 0.08370365306431757 resp.

GMM #81 trained with 30 components, cov. type full, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6356969905331503, 0.5322908756110916 and 0.016274334213934167 resp.

GMM #82 trained with 30 components, cov. type full, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6113502842566055, 0.5466321752284938 and 0.04375350940597203 resp.

GMM #83 trained with 30 components, cov. type tied, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6373881011265745, 0.5422618240730109 and 0.06897430179928837 resp.

GMM #84 trained with 30 components, cov. type tied, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.10769395746255303, 0.08791057042992256 and -0.08242960838829867 resp.

GMM #85 trained with 30 components, cov. type diag, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5954412377143761, 0.49122922657462736 and 0.0031382932586419957 resp.

GMM #86 trained with 30 components, cov. type diag, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5848427634970648, 0.48013216066996744 and -0.003025714290702317 resp.

GMM #87 trained with 30 components, cov. type spherical, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6279943478843549, 0.5111398111262072 and 0.07612519543473795 resp.

GMM #88 trained with 30 components, cov. type spherical, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6304679980661473, 0.5125340766324475 and 0.07780151448485162 resp.

GMM #89 trained with 32 components, cov. type full, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6447879798681181, 0.5353702194413904 and 0.01590763095286477 resp.

GMM #90 trained with 32 components, cov. type full, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6061571974065333, 0.5425416640834348 and 0.04956043615272582 resp.

GMM #91 trained with 32 components, cov. type tied, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6438923642088882, 0.541378990067031 and 0.06882814055770531 resp.

GMM #92 trained with 32 components, cov. type tied, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.10260184690750473, 0.08253897799427365 and -0.08755638270161785 resp.

GMM #93 trained with 32 components, cov. type diag, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6036365955004808, 0.4912536044374901 and -0.0022181960302024924 resp.

GMM #94 trained with 32 components, cov. type diag, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5852016100052166, 0.4794125754748545 and -0.008436501820797505 resp.

GMM #95 trained with 32 components, cov. type spherical, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6333740960164178, 0.5103616954834367 and 0.07434934599981138 resp.

GMM #96 trained with 32 components, cov. type spherical, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6312447821355572, 0.5073495684814024 and 0.07795703303479413 resp.

GMM #97 trained with 34 components, cov. type full, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6469137273860927, 0.5300863852218907 and 0.013163282722900798 resp.

GMM #98 trained with 34 components, cov. type full, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5519360369276791, 0.5047171308976462 and 0.048446544748734124 resp.

GMM #99 trained with 34 components, cov. type tied, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6408098402806431, 0.5322431655047817 and 0.06604082591718548 resp.

GMM #100 trained with 34 components, cov. type tied, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.09919468035980761, 0.07900530260519674 and -0.07943626851989846 resp.

GMM #101 trained with 34 components, cov. type diag, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6005213212900432, 0.4840761721996331 and 0.002080866338576563 resp.

GMM #102 trained with 34 components, cov. type diag, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6055275356244055, 0.48856249506681015 and -0.0007606854027688692 resp.

GMM #103 trained with 34 components, cov. type spherical, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6443530882989187, 0.5111920470485384 and 0.07358532843834488 resp.

GMM #104 trained with 34 components, cov. type spherical, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6488464594843347, 0.5177527650441761 and 0.07944652999981622 resp.

GMM #105 trained with 36 components, cov. type full, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6561050923912535, 0.5330845294939696 and 0.00871787654114858 resp.

GMM #106 trained with 36 components, cov. type full, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5949777741121877, 0.5241585655960902 and 0.0365670195446916 resp.

GMM #107 trained with 36 components, cov. type tied, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6401716787442303, 0.5279036065279169 and 0.07333843304465923 resp.

GMM #108 trained with 36 components, cov. type tied, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.10952871250207208, 0.08660358626111823 and -0.09376487687045444 resp.

GMM #109 trained with 36 components, cov. type diag, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6146340068735646, 0.4904045496394524 and 0.0020939626603080222 resp.

GMM #110 trained with 36 components, cov. type diag, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5990652934108179, 0.4804013209884637 and -0.00690728956503555 resp.

GMM #111 trained with 36 components, cov. type spherical, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.645125447424523, 0.5082997004706775 and 0.07173145159756025 resp.

GMM #112 trained with 36 components, cov. type spherical, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6396633426308849, 0.5043550197153982 and 0.0764653587834188 resp.

GMM #113 trained with 38 components, cov. type full, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6526586574149544, 0.5273375845531014 and 0.001828516530424049 resp.

GMM #114 trained with 38 components, cov. type full, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5927073937731158, 0.5212844803189792 and 0.03764787556562273 resp.

GMM #115 trained with 38 components, cov. type tied, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.64484644735645, 0.5262162042580933 and 0.07000907665555685 resp.

GMM #116 trained with 38 components, cov. type tied, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.10678113419592955, 0.08401429882567489 and -0.08599125466470015 resp.

GMM #117 trained with 38 components, cov. type diag, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6187608598478858, 0.4889705949138141 and 0.004518739188253759 resp.

GMM #118 trained with 38 components, cov. type diag, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6099435002030874, 0.4838189529688526 and -0.006007908763835657 resp.

GMM #119 trained with 38 components, cov. type spherical, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.649093697934968, 0.5059536393081042 and 0.07355616229886916 resp.

GMM #120 trained with 38 components, cov. type spherical, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6480980207774807, 0.5062681884364958 and 0.07422653488058151 resp.

GMM #121 trained with 40 components, cov. type full, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.670583711966034, 0.5372405935265697 and 0.01005202834266495 resp.

GMM #122 trained with 40 components, cov. type full, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.5862346254503419, 0.5111723046725593 and 0.03338220120822476 resp.

GMM #123 trained with 40 components, cov. type tied, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6556166237687404, 0.5307621186136762 and 0.06594676848149288 resp.

GMM #124 trained with 40 components, cov. type tied, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.10691548111892622, 0.08300541979463538 and -0.08245505598656584 resp.

GMM #125 trained with 40 components, cov. type diag, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6248986917974485, 0.4933888246835814 and 0.0058015101763056395 resp.

GMM #126 trained with 40 components, cov. type diag, and a random parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6180914929273682, 0.4884024239070429 and -0.0006485187916137647 resp.

GMM #127 trained with 40 components, cov. type spherical, and a kmeans parameter initialization, which yielded homogeneity, v_measure and silhouette scores of 0.6539658836638477, 0.50553068625482 and 0.0758706518064718 resp.

```
GMM #128 trained with 40 components, cov. type spherical, and a random parameter
initialization, which yielded homogeneity, v_measure and silhouette scores of
0.6530918702900061, 0.5041000790322858 and 0.0688983961817038 resp.
```

[37]:
```
for i in range(1, 129):
    y_pred = grid_search.predict(i, x_test, y_test)
```

```
GMM #1 yielded homogeneity, v_measure and silhouette scores of
0.5492297436734518, 0.5563060612807291 and 0.08098997682317997 resp. on the test
set
GMM #2 yielded homogeneity, v_measure and silhouette scores of
0.5060642316896842, 0.5322830070420804 and 0.0895828290812694 resp. on the test
set
GMM #3 yielded homogeneity, v_measure and silhouette scores of
0.5728075097533654, 0.5872633169349237 and 0.09267037477070339 resp. on the test
set
GMM #4 yielded homogeneity, v_measure and silhouette scores of
0.054115982523730455, 0.054385031772762424 and -0.03737284065814241 resp. on the
test set
GMM #5 yielded homogeneity, v_measure and silhouette scores of
0.4221928611689869, 0.4363229458671653 and 0.03456878587517899 resp. on the test
set
GMM #6 yielded homogeneity, v_measure and silhouette scores of
0.41140280811971053, 0.42476908404433816 and 0.020868564071252853 resp. on the
test set
GMM #7 yielded homogeneity, v_measure and silhouette scores of
0.4829051828580381, 0.48734775589756274 and 0.11909541462416337 resp. on the
test set
GMM #8 yielded homogeneity, v_measure and silhouette scores of
0.4746634460581876, 0.4786690497021991 and 0.10547570350299629 resp. on the test
set
GMM #9 yielded homogeneity, v_measure and silhouette scores of
0.5577541932947284, 0.548084759787255 and 0.06432097856722659 resp. on the test
set
GMM #10 yielded homogeneity, v_measure and silhouette scores of
0.5099913885840314, 0.5143020266714203 and 0.07604695820458196 resp. on the test
set
GMM #11 yielded homogeneity, v_measure and silhouette scores of
0.6182757294672525, 0.606287374544866 and 0.09893370796371523 resp. on the test
set
GMM #12 yielded homogeneity, v_measure and silhouette scores of
0.05997798777861918, 0.058359667498691016 and -0.041445991235889657 resp. on the
test set
GMM #13 yielded homogeneity, v_measure and silhouette scores of
0.4602939913121982, 0.45449844616753304 and 0.027186890669517516 resp. on the
test set
GMM #14 yielded homogeneity, v_measure and silhouette scores of
```

0.43221348837516677, 0.42632290537563255 and -0.003439354339102841 resp. on the test set
GMM #15 yielded homogeneity, v_measure and silhouette scores of 0.5222597658088481, 0.5066350419405754 and 0.11463343251406609 resp. on the test set
GMM #16 yielded homogeneity, v_measure and silhouette scores of 0.532279761523049, 0.5149448998786567 and 0.1059384518774572 resp. on the test set
GMM #17 yielded homogeneity, v_measure and silhouette scores of 0.5546619865091159, 0.5251862047365673 and 0.05862339831080041 resp. on the test set
GMM #18 yielded homogeneity, v_measure and silhouette scores of 0.5249377969668607, 0.5225011149963239 and 0.08010234324594372 resp. on the test set
GMM #19 yielded homogeneity, v_measure and silhouette scores of 0.632582785892556, 0.6075207678718232 and 0.09832733919218299 resp. on the test set
GMM #20 yielded homogeneity, v_measure and silhouette scores of 0.043784743165218014, 0.04107908485778861 and -0.04926940204317424 resp. on the test set
GMM #21 yielded homogeneity, v_measure and silhouette scores of 0.49525991352244425, 0.47019575928046203 and 0.021225332168280316 resp. on the test set
GMM #22 yielded homogeneity, v_measure and silhouette scores of 0.4777184020802149, 0.45592897834824164 and 0.009670955539160916 resp. on the test set
GMM #23 yielded homogeneity, v_measure and silhouette scores of 0.550063228629128, 0.5174882725755856 and 0.1151148642286942 resp. on the test set
GMM #24 yielded homogeneity, v_measure and silhouette scores of 0.5594356441418148, 0.5251701405263116 and 0.11458586568369729 resp. on the test set
GMM #25 yielded homogeneity, v_measure and silhouette scores of 0.565126456665959, 0.5255238598816955 and 0.05411908532457854 resp. on the test set
GMM #26 yielded homogeneity, v_measure and silhouette scores of 0.5574896996717797, 0.5287558518854961 and 0.06752053211291116 resp. on the test set
GMM #27 yielded homogeneity, v_measure and silhouette scores of 0.632300566740357, 0.5955286240603316 and 0.10485043621448353 resp. on the test set
GMM #28 yielded homogeneity, v_measure and silhouette scores of 0.07644249305918527, 0.06999229011857595 and -0.0486249684357642 resp. on the test set
GMM #29 yielded homogeneity, v_measure and silhouette scores of 0.4965080118594711, 0.4579286873111916 and 0.01909795498673785 resp. on the test set
GMM #30 yielded homogeneity, v_measure and silhouette scores of

0.49157938502395504, 0.45972305362253374 and 0.0025100376892440725 resp. on the test set
GMM #31 yielded homogeneity, v_measure and silhouette scores of 0.5608032192900976, 0.5176152874503557 and 0.11273261735925189 resp. on the test set
GMM #32 yielded homogeneity, v_measure and silhouette scores of 0.5712921839676047, 0.5216744910343434 and 0.09872517613416 resp. on the test set
GMM #33 yielded homogeneity, v_measure and silhouette scores of 0.5904483238734498, 0.5395253462005211 and 0.05187586545544616 resp. on the test set
GMM #34 yielded homogeneity, v_measure and silhouette scores of 0.542859898143428, 0.5332920158545255 and 0.06083352695390386 resp. on the test set
GMM #35 yielded homogeneity, v_measure and silhouette scores of 0.6370413248981837, 0.5888576605234688 and 0.09100518178732635 resp. on the test set
GMM #36 yielded homogeneity, v_measure and silhouette scores of 0.07765262555321492, 0.06924529173362151 and -0.0765799956465718 resp. on the test set
GMM #37 yielded homogeneity, v_measure and silhouette scores of 0.5292619712588189, 0.47957310134953696 and 0.028697262487633584 resp. on the test set
GMM #38 yielded homogeneity, v_measure and silhouette scores of 0.4683125142761911, 0.42785175346392507 and -0.028967872352952076 resp. on the test set
GMM #39 yielded homogeneity, v_measure and silhouette scores of 0.5920017617168565, 0.531072035441043 and 0.10448239277485075 resp. on the test set
GMM #40 yielded homogeneity, v_measure and silhouette scores of 0.5841039161406292, 0.5217716268407302 and 0.0969570599238824 resp. on the test set
GMM #41 yielded homogeneity, v_measure and silhouette scores of 0.5926076214983949, 0.5331845026642604 and 0.03422790148513623 resp. on the test set
GMM #42 yielded homogeneity, v_measure and silhouette scores of 0.5687180424810963, 0.5404708409199765 and 0.05637667309939955 resp. on the test set
GMM #43 yielded homogeneity, v_measure and silhouette scores of 0.6354442861491408, 0.582302451603886 and 0.08345232696148294 resp. on the test set
GMM #44 yielded homogeneity, v_measure and silhouette scores of 0.07664315484485114, 0.06732713967942831 and -0.06153654211654342 resp. on the test set
GMM #45 yielded homogeneity, v_measure and silhouette scores of 0.5246264743572164, 0.4673876042115289 and -0.002927057145478752 resp. on the test set
GMM #46 yielded homogeneity, v_measure and silhouette scores of

0.5207391480037575, 0.46473624349443715 and -0.002208335954356113 resp. on the test set
GMM #47 yielded homogeneity, v_measure and silhouette scores of 0.5932320172009288, 0.5183902078373643 and 0.08849534594583526 resp. on the test set
GMM #48 yielded homogeneity, v_measure and silhouette scores of 0.591614824530954, 0.5209069155702588 and 0.10023408787834114 resp. on the test set
GMM #49 yielded homogeneity, v_measure and silhouette scores of 0.6085469230645741, 0.5376730254509643 and 0.03204475335549253 resp. on the test set
GMM #50 yielded homogeneity, v_measure and silhouette scores of 0.5840970044589819, 0.5407759065279217 and 0.045669803685534784 resp. on the test set
GMM #51 yielded homogeneity, v_measure and silhouette scores of 0.628252855394869, 0.5558200836553692 and 0.07582750048022847 resp. on the test set
GMM #52 yielded homogeneity, v_measure and silhouette scores of 0.07498351334209276, 0.06460586372654406 and -0.05981119411918794 resp. on the test set
GMM #53 yielded homogeneity, v_measure and silhouette scores of 0.5573222758936549, 0.48579091145975595 and -0.0009314958535951472 resp. on the test set
GMM #54 yielded homogeneity, v_measure and silhouette scores of 0.5229423067293238, 0.4582945221417903 and -0.008170494633269892 resp. on the test set
GMM #55 yielded homogeneity, v_measure and silhouette scores of 0.6050114539234628, 0.5200430185312269 and 0.08773435546308993 resp. on the test set
GMM #56 yielded homogeneity, v_measure and silhouette scores of 0.6007459038012954, 0.5167395457771872 and 0.08520958156559774 resp. on the test set
GMM #57 yielded homogeneity, v_measure and silhouette scores of 0.6145865438511855, 0.5360038255057448 and 0.030112409215559473 resp. on the test set
GMM #58 yielded homogeneity, v_measure and silhouette scores of 0.5847166812960344, 0.5407299449614407 and 0.05861808510574932 resp. on the test set
GMM #59 yielded homogeneity, v_measure and silhouette scores of 0.6417470600130685, 0.562844999518346 and 0.0764870334283554 resp. on the test set
GMM #60 yielded homogeneity, v_measure and silhouette scores of 0.09158524370260444, 0.07760150384669773 and -0.07481179529345788 resp. on the test set
GMM #61 yielded homogeneity, v_measure and silhouette scores of 0.5708228212101388, 0.4915607345066861 and 0.006629525100266338 resp. on the test set
GMM #62 yielded homogeneity, v_measure and silhouette scores of

0.5708057698169106, 0.48951964445709856 and 0.0013143005279558121 resp. on the test set
GMM #63 yielded homogeneity, v_measure and silhouette scores of 0.6079446954901471, 0.5156267477614893 and 0.08897363801840766 resp. on the test set
GMM #64 yielded homogeneity, v_measure and silhouette scores of 0.6052903840154027, 0.5132485861151722 and 0.0833489409340663 resp. on the test set
GMM #65 yielded homogeneity, v_measure and silhouette scores of 0.6207447473623444, 0.534394579842044 and 0.02058810844437629 resp. on the test set
GMM #66 yielded homogeneity, v_measure and silhouette scores of 0.5750998438627278, 0.5255819835560529 and 0.03830652301569563 resp. on the test set
GMM #67 yielded homogeneity, v_measure and silhouette scores of 0.6451708288148085, 0.5563582686007437 and 0.06398862607695839 resp. on the test set
GMM #68 yielded homogeneity, v_measure and silhouette scores of 0.08438819656151751, 0.07079144763091823 and -0.0932835547062239 resp. on the test set
GMM #69 yielded homogeneity, v_measure and silhouette scores of 0.5796309948971494, 0.4909473675921015 and -0.0018115399475804332 resp. on the test set
GMM #70 yielded homogeneity, v_measure and silhouette scores of 0.5514261512641148, 0.4692480113805209 and -0.012076315629434082 resp. on the test set
GMM #71 yielded homogeneity, v_measure and silhouette scores of 0.616063080902579, 0.5150116090396865 and 0.08278768034916684 resp. on the test set
GMM #72 yielded homogeneity, v_measure and silhouette scores of 0.6163501084387061, 0.515958315377266 and 0.08013958164327711 resp. on the test set
GMM #73 yielded homogeneity, v_measure and silhouette scores of 0.6266239160067993, 0.5316756929481818 and 0.015544654342503312 resp. on the test set
GMM #74 yielded homogeneity, v_measure and silhouette scores of 0.5889858485301285, 0.5405188352692055 and 0.051807044921085024 resp. on the test set
GMM #75 yielded homogeneity, v_measure and silhouette scores of 0.6374172781124182, 0.5467648612361959 and 0.0685628408546022 resp. on the test set
GMM #76 yielded homogeneity, v_measure and silhouette scores of 0.10052639910714675, 0.08306060004966935 and -0.07421382450594688 resp. on the test set
GMM #77 yielded homogeneity, v_measure and silhouette scores of 0.5840075608164408, 0.48749816269987123 and 0.0010938202271712559 resp. on the test set
GMM #78 yielded homogeneity, v_measure and silhouette scores of

0.569093104707644, 0.4766201719369689 and -0.00026271773944045655 resp. on the test set
GMM #79 yielded homogeneity, v_measure and silhouette scores of 0.6182132401374867, 0.5095241990433907 and 0.0759031780107229 resp. on the test set
GMM #80 yielded homogeneity, v_measure and silhouette scores of 0.6237795773507082, 0.5146263657919716 and 0.0819109344086993 resp. on the test set
GMM #81 yielded homogeneity, v_measure and silhouette scores of 0.6350582088712838, 0.5321031276666173 and 0.01627710518792394 resp. on the test set
GMM #82 yielded homogeneity, v_measure and silhouette scores of 0.6154674984999282, 0.5542018649591176 and 0.022652048917963626 resp. on the test set
GMM #83 yielded homogeneity, v_measure and silhouette scores of 0.6358895688674996, 0.5410578027108178 and 0.06576330064172337 resp. on the test set
GMM #84 yielded homogeneity, v_measure and silhouette scores of 0.11234770411454562, 0.09172339595864935 and -0.08758603699383612 resp. on the test set
GMM #85 yielded homogeneity, v_measure and silhouette scores of 0.5955584546331582, 0.4918131812273247 and 0.0008992887605974124 resp. on the test set
GMM #86 yielded homogeneity, v_measure and silhouette scores of 0.5840261976854604, 0.47989981019721994 and -0.007422629624990959 resp. on the test set
GMM #87 yielded homogeneity, v_measure and silhouette scores of 0.6281443128929451, 0.5112310534188135 and 0.07433128961035129 resp. on the test set
GMM #88 yielded homogeneity, v_measure and silhouette scores of 0.6286610085875144, 0.5109544867820468 and 0.07701768385006767 resp. on the test set
GMM #89 yielded homogeneity, v_measure and silhouette scores of 0.6443200575413612, 0.5360942197926221 and 0.015809356913760557 resp. on the test set
GMM #90 yielded homogeneity, v_measure and silhouette scores of 0.6080139308376759, 0.549683858040066 and 0.052143039721204175 resp. on the test set
GMM #91 yielded homogeneity, v_measure and silhouette scores of 0.6424860995626958, 0.5405959514852792 and 0.06581825532082783 resp. on the test set
GMM #92 yielded homogeneity, v_measure and silhouette scores of 0.10535288213054166, 0.08485297755309906 and -0.08968683290626923 resp. on the test set
GMM #93 yielded homogeneity, v_measure and silhouette scores of 0.6030159981342, 0.49155928985426484 and -0.00639328756299314 resp. on the test set
GMM #94 yielded homogeneity, v_measure and silhouette scores of 0.582134732195507, 0.47833381249929197 and -0.013482518356510877 resp. on the

test set
GMM #95 yielded homogeneity, v_measure and silhouette scores of
0.631847568489133, 0.5092333872684234 and 0.0743702982681751 resp. on the test
set
GMM #96 yielded homogeneity, v_measure and silhouette scores of
0.6303507567300694, 0.5067584440376047 and 0.07584151398050358 resp. on the test
set
GMM #97 yielded homogeneity, v_measure and silhouette scores of
0.6459248646979752, 0.5310223607493146 and 0.01378747084384716 resp. on the test
set
GMM #98 yielded homogeneity, v_measure and silhouette scores of
0.5495525087238635, 0.5120941134669319 and 0.046840391722968576 resp. on the
test set
GMM #99 yielded homogeneity, v_measure and silhouette scores of
0.6448739122109067, 0.5357199388551221 and 0.06300476371598109 resp. on the test
set
GMM #100 yielded homogeneity, v_measure and silhouette scores of
0.09715861180716337, 0.07744648207328969 and -0.08140208647594406 resp. on the
test set
GMM #101 yielded homogeneity, v_measure and silhouette scores of
0.6008612424396097, 0.4851228883112997 and -0.0009803212765848363 resp. on the
test set
GMM #102 yielded homogeneity, v_measure and silhouette scores of
0.6066726772603597, 0.4896641145729233 and -0.002812383259228808 resp. on the
test set
GMM #103 yielded homogeneity, v_measure and silhouette scores of
0.644044313774318, 0.5110444497175128 and 0.07029138400793054 resp. on the test
set
GMM #104 yielded homogeneity, v_measure and silhouette scores of
0.6489853940943698, 0.5179694818073408 and 0.07644797734903908 resp. on the test
set
GMM #105 yielded homogeneity, v_measure and silhouette scores of
0.652703520029303, 0.5318925404163822 and 0.008556752840734977 resp. on the test
set
GMM #106 yielded homogeneity, v_measure and silhouette scores of
0.5964243949554762, 0.5310780160841452 and 0.040525516384861175 resp. on the
test set
GMM #107 yielded homogeneity, v_measure and silhouette scores of
0.6420867300441507, 0.5295983651446601 and 0.06990453940557505 resp. on the test
set
GMM #108 yielded homogeneity, v_measure and silhouette scores of
0.11484064946382673, 0.09085554250652672 and -0.09669141758481141 resp. on the
test set
GMM #109 yielded homogeneity, v_measure and silhouette scores of
0.6140548794001198, 0.49028625233779516 and -0.00023371377856054956 resp. on the
test set
GMM #110 yielded homogeneity, v_measure and silhouette scores of
0.596509480383798, 0.47960871120033693 and -0.011670270476432744 resp. on the

test set
GMM #111 yielded homogeneity, v_measure and silhouette scores of
0.6425632062550398, 0.5064668649126424 and 0.07035656931062909 resp. on the test
set
GMM #112 yielded homogeneity, v_measure and silhouette scores of
0.6408963321152251, 0.5052708323758979 and 0.0742571065132989 resp. on the test
set
GMM #113 yielded homogeneity, v_measure and silhouette scores of
0.6495528145475682, 0.5274040078974358 and 0.007269445398636043 resp. on the
test set
GMM #114 yielded homogeneity, v_measure and silhouette scores of
0.5966356366328227, 0.532036309586 and 0.039455995825520826 resp. on the test
set
GMM #115 yielded homogeneity, v_measure and silhouette scores of
0.646158951848421, 0.5274626810079904 and 0.0662786463909319 resp. on the test
set
GMM #116 yielded homogeneity, v_measure and silhouette scores of
0.11029044674503394, 0.08678746229721276 and -0.08598871110388136 resp. on the
test set
GMM #117 yielded homogeneity, v_measure and silhouette scores of
0.6157435068327551, 0.4873635508537411 and 0.0018685759767998011 resp. on the
test set
GMM #118 yielded homogeneity, v_measure and silhouette scores of
0.6061980727590546, 0.48198565078648115 and -0.00986681283235269 resp. on the
test set
GMM #119 yielded homogeneity, v_measure and silhouette scores of
0.6466671134467514, 0.5043006383656269 and 0.07174180010030833 resp. on the test
set
GMM #120 yielded homogeneity, v_measure and silhouette scores of
0.6462651215057374, 0.5050257228895632 and 0.0720910427701889 resp. on the test
set
GMM #121 yielded homogeneity, v_measure and silhouette scores of
0.6678109956418901, 0.5371267930985015 and 0.009435596846425146 resp. on the
test set
GMM #122 yielded homogeneity, v_measure and silhouette scores of
0.5894001468085891, 0.5224480668320534 and 0.03148402313802997 resp. on the test
set
GMM #123 yielded homogeneity, v_measure and silhouette scores of
0.6528062556509402, 0.5285616090919159 and 0.061788398664010626 resp. on the
test set
GMM #124 yielded homogeneity, v_measure and silhouette scores of
0.11054874508226999, 0.08584163981824668 and -0.08442576985041735 resp. on the
test set
GMM #125 yielded homogeneity, v_measure and silhouette scores of
0.6210911832101629, 0.4909996572310958 and 0.003585654635734014 resp. on the
test set
GMM #126 yielded homogeneity, v_measure and silhouette scores of
0.6159702587959945, 0.4876118468058923 and -0.0060130275921986024 resp. on the

```
test set
GMM #127 yielded homogeneity, v_measure and silhouette scores of
0.6537981272895338, 0.5054936739394688 and 0.07298637759270613 resp. on the test
set
GMM #128 yielded homogeneity, v_measure and silhouette scores of
0.6531165268159834, 0.5044738401649339 and 0.06645619874662943 resp. on the test
set
```

After training and testing over 128 GMM models by varying the hyperparameters, we find that the best results are achieved here:

Model number	k components	Covariance parameter	Initialization parameter
11	12	tied	kmeans
19	14	tied	kmeans
35	18	tied	kmeans
59	24	tied	kmeans
91	32	tied	kmeans
121	40	full	kmeans

Model number	Homogeneity score (train set)	V-measure score (train set)	Silhouette score (train set)	Homogeneity score (test set)	V-measure score (test set)	Silhouette score (test set)
11	0.624	0.612	0.102	0.618	0.606	0.099
19	0.634	0.608	0.101	0.633	0.608	0.098
35	0.637	0.589	0.093	0.637	0.589	0.091
59	0.641	0.562	0.078	0.642	0.563	0.076
91	0.644	0.541	0.069	0.642	0.541	0.066
121	0.671	0.537	0.010	0.669	0.537	0.001

In conclusion. It seems that the best results (a homogeneity and v-measure scores above 0.6 and a silhouette score around 0.1) are obtained while using between **12 and 24 components**, with a **kmeans initialization** and a **tied covariance matric** (i.e. all components share the same general covariance matrix).

As we increase the number of component, the silhouette score seems to drop towards 0, indicating that mixtures are increasingly overlapping each other.

1.7 6. References used in this exercise

Dataset information:

- https://www.kaggle.com/c/ttic-31020-hw5-fmnist-gmm/leaderboard

Kmeans and GMM information:

- https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
- https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html

- https://brilliant.org/wiki/gaussian-mixture-model/
- https://towardsdatascience.com/gaussian-mixture-models-implemented-from-scratch-1857e40ea566
- https://ukdevguy.com/k-means-algorithm-for-clustering/

Metrics information:

- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.v_measure_score.html
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.homogeneity_score.html
- https://www.geeksforgeeks.org/ml-v-measure-for-evaluating-clustering-performance/