Tesina di "Fondamenti di scienza dei dati" 2022-2023

Andrea Spinelli, Raffaele Terracino, Marco Valenti

<19/09/2023>

Indice

1. - Traccia

2. - Pipeline Analysis

    2.1 - Data collection

    2.2 - Data preprocessing

    2.3 - Analytical processing

3. - Risultati

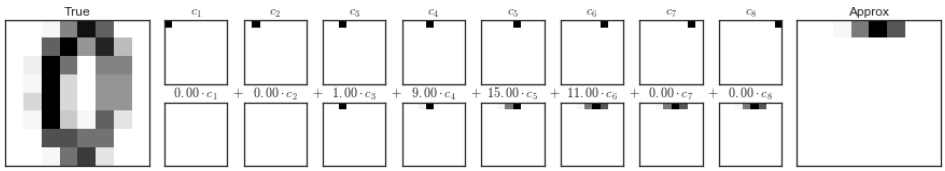
    3.1 - Clustering & Matrix Confusion

4. - Extra

    4.1 - Plotting dei primi n numeri

1 Traccia

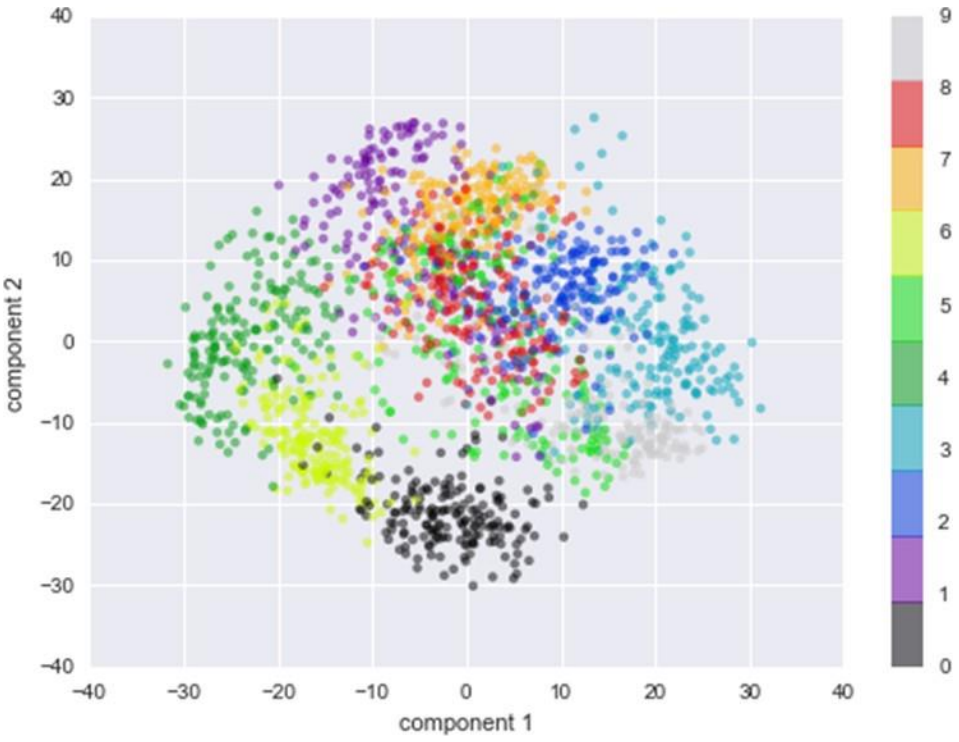
Sia dato il dataset `digits.csv` che contiene le immagini di numeri scritti a mano (pixel), inseriti come righe del file, in cui l'ultima colonna di ogni riga rappresenta il target.



https://www.educative.io/answers/what-is-datasetsloaddigits-in-sklearn

Realizzare un **Progetto di Data Analysis** che:

- Prevede l' `utilizzo della PCA` per proiettare le dimensioni sulle prime due componenti principali, questi punti sono la proiezione di ciascun punto dati lungo le direzioni con la varianza maggiore;
- Prevede la `clusterizzazione dei punti` () tramite tecnica - *Means* (basato sul centroide);
- Ripete la `clusterizzazione sulle proiezioni sulle prime N componenti principali,` con che varia tra e 2 e 6;
- Per ogni clusterizzazione produce una `matrice di confusione` che metta a confronto l'accuracy nella classificazione al variare del numero di componenti principali scelte.



2 Pipeline di data processing

2.1 Data collection

La tesina seguirà la **Pipeline di elaborazione dati** studiata durante il corso, in modo tale da risultare più chiaro il procedimento seguito.

> Il *dataset* di riferimento per il progetto viene fornito sia dalla libreria `scikit-learn` di Python, sia in formato csv in allegato alla traccia, ed è consultabile in `../dataset/digits.csv`. Per importare il dataset mediante `scikit-learn` è sufficiente utilizzare il metodo `load_digits()`, che restituisce un oggetto da cui è possibile ricavare gli attributi `data` e `target`, entrambi di tipo `ndarray`.
> Il primo rappresenta una matrice di dimensione 1797×64, tale che `data[i,j]` è il livello di grigio del `j`-esimo pixel dell'immagine `i`; `target` è invece un array di 1797 elementi taleche `target[i]` è la cifra raffigurata nell'immagine `i`.
>
> Il file csv usa come separatore la virgola e contiene, per ciascun record, 65 campi, di cui i primi 64 corrispondono esattamente a `data`, mentre l'ultimo corrisponde a `target`.

Formalmente, si sta parlando di dati multidimensionali, in cui ogni record ha la seguente forma:

$$D = (x_1, \dots, x_{64}, y), \qquad x_j \in \{0, \dots, 255\}, \qquad y \in \{0, \dots, 9\}$$

Vista la facilità di accesso fornita dalla libreria utilizzata, la prima modalità di lettura del dataset è stata scelta per il progetto.

In [ ]:
```
import numpy as np
  from scipy.stats import mode
  from sklearn.decomposition import PCA
  from sklearn.datasets import load_digits
  from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
  import matplotlib.pyplot as plt

  digits = load_digits()

  X, Y = digits.data, digits.target
```

2.2 Data preprocessing

Lo step successivo è la **pre-elaborazione dei dati**, a sua volta suddivisa in più fasi. L' **Estrazione delle features rilevanti**, in generale necessaria affinché si riduca la quantità di dati, qui non deve essere eseguita in quanto servono tutte le 64 features per lo svolgimento del progetto.

Caricando il dataset con `sklearn` e analizzandolo, si è notato che non è necessaria alcuna forma di **Pulizia dei dati**, in quanto il dataset risulta pulito e pronto all'uso. Un eventuale valore mancante potrebbe essere colmato facendo la media o la mediana dei livelli di grigio dei pixel vicini, oppure con tecniche più fini come il curve fitting.

Se si volesse fare **Data Integration**, basterebbe appendere al file csv una sequenza di 64 valori rappresentanti l'immagine, seguite da una cifra per indicare il numero rappresentato. L'identificazione degli outlier non è risultata necessaria, tuttavia si osserva che nel caso delle immagini un outlier può essere identificato mediante istogrammi, contando le occorrenze di ogni livello di grigio.
Per esempio, un singolo pixel bianco in un'immagina scura rappresenta un outlier.

Con lo scopo di voler visualizzare i dati in modo tale da poter fare delle osservazioni, i dati possono essere *normalizzati* mediante la tecnica di *Scalatura decimale*, tipica della **Data Transformation.** Non sono risultate necessarie operazioni di - **Data Reduction**. In un ipotetico caso di lavorare con un miliardo di record, facendo un calcolo si scopre che si tratterebbe di lavorare con un dataset di circa $7.5Gb$. Nonostante non sia eccessivamente oneroso per una macchina prestante, viste le analisi che si devono condurre, tecniche di data reduction possono essere di aiuto, come l'aggregazione o il campionamento.

> La seguente istruzione esegue la Scalatura decimale per il dataset, riducendo il range dei campi nell'intervallo

In [ ]:
```
X = X/1000
```

2.3 Analytical processing

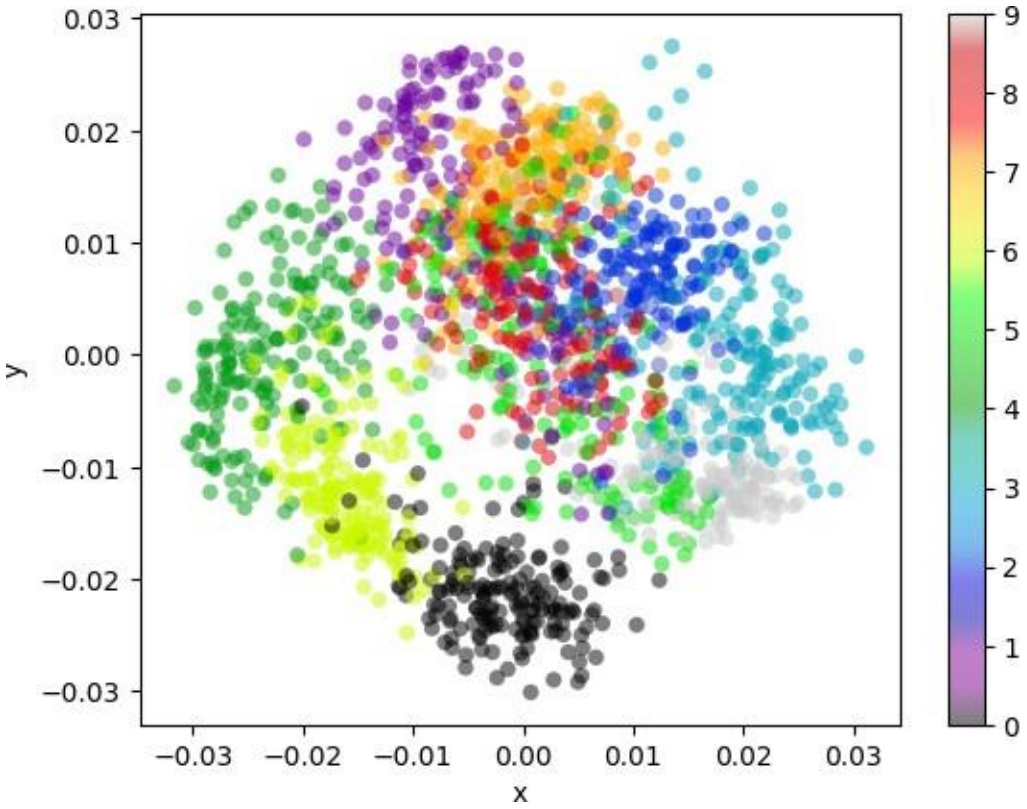
L'ultima fase della Pipeline è l'**elaborazione dei dati**. L'obiettivo del progetto è la *rappresentanza*, pertanto si farà riferimento all'**apprendimento non supervisionato**, sfruttando sia la **riduzione della dimensionalità** che il **clustering**.

Si fa uso di entrambe le tecniche perché si vuole visualizzare se il modello sia capace di raggruppare in modo corretto, o meno, le immagini delle cifre che gli si danno in input.
In questo viene d'aiuto il clustering che realizza tale compito; tuttavia, affinché si possa visualizzare, confrontare e analizzare, è necessario ridurre la dimensionata dei dati in modo tali da rappresentarli, in questo ci viene d'aiuto la riduzione dei dati.

Si parte da un dataset a dimensioni, che devono essere ridotte a due. L'idea della **PCA** è quella di trasformare i record in punti e proiettarli, nel piano cartesiano, lungo le loro direzioni con la varianza maggiore.
La **PCA** rivela le strutture nascoste a livello multidimensionale riducendole ad un numero basso per poterle rappresentare.

> Nel codice a seguire si sceglie il numero di componenti principali, dopodiché si fa uso dei metodi, forniti dalla libreria `sklearn.decomposition`. `PCA()`, che prepara il numero delle componenti principali, e `fit_transform()`, che effettua la trasformazione dei dati nelle componenti principali scelte.
> A seguire vi sono i metodi per effettuare il plot del risultato.

In [ ]:
```
n_comp = 2

  X = - PCA(n_comp).fit_transform(X)

  plt.scatter(X[:,0], X[:,1],
              c=Y, edgecolor='none', alpha=0.5,
              cmap=plt.colormaps["nipy_spectral"])

  plt.colorbar()
  plt.xlabel('x')
  plt.ylabel('y')
  plt.show()
```



Per effettuare la clusterizzazione si è scelto l'algoritmo di clustering K-means,implementato da zero. Tale scelta è dovuta sia per motivi didattici sia per particolari esigenze sull'output, difatti la seguente versione di -Means restituisce, oltre ai cluster, anche i centroidi di ogni cluster.
Seguendo una strategia bottom-up, il primo passo è l'inizializzazione dei centroidi in modo casuale. Ciò viene fatto mediante la funzione `init_rand_centroids(K, X)`,dove `K` è il numero di cluster e `X` i dati da clusterizzare.

In [ ]:
```
# Inizializzazione di K centroidi casuali d'esempio dalla matrice X
  def  init_rand_centroids(K,X):

      m,n = np.shape(X)
      centroids = np.empty((K,n))

      for i in range(K):
          centroids[i] =   X[np.random.choice(range(m))]

      return centroids
```

La metrica scelta per calcolare la distanza dei punti dai centroidi è la distanza euclidea, implementata nella funzione che segue

In [ ]
```
# Calcolo della distanza euclidea
  def euclidean_distance(x1, x2):
      return np.linalg.norm(x1 - x2)
```

La seguente funzione, necessaria per la costruzione dei cluster, trova ilcentroide più vicino a un dato vettore `X`

In [ ]:
```
# Trova e restituisce l'indice del centroide più vicino di un vettore d
  def closest_centroid(x, centroids, K):

      distances = np.empty(K)

      for i in range(K):
          distances[i] = euclidean_distance(centroids[i], x)

      return np.argmin(distances)
```

Per la creazione dei cluster basta eseguire la funzione precedente `m` volte, dove `m` è il numero di righe del dataset.

In [ ]:
```
# Restituisce una matrice di indici di cluster per tutti i campioni di
  def create_clusters(centroids, K, X):

      m, _ = np.shape(X)
      cluster_idx = np.empty(m)

      for i in range(m):
          cluster_idx[i] = closest_centroid(X[i], centroids, K)

      return cluster_idx
```

La funzione seguente serve per ricalcolare i centroidi ad ogni iterazione

In [ ]:
```
# Calcolo della media dei nuovi centroidi dei cluster
 def means(cluster_idx, K, X):

     _, n = np.shape(X)
     centroids = np.empty((K, n))

     for i in range(K):
         cluster_i_points = X[cluster_idx == i]
         centroids[i] = np.mean(cluster_i_points, axis=0)

     return centroids
```

L'algoritmo finale è incapsulato nella funzione `K_means`, che prende a parametro il numero di cluster, il dataset di cui effettuare la clusterizzazione e il massimo numero di iterazioni. Dopo aver inizializzato i `K` centroidi, in ogni iterazione si creano i cluster e si controlla se vi sia o meno convergenza, nel qual caso si esce dalla funzione prima del numero massimo di iterazioni fissate. In ogni caso vengono restituiti sia i cluster che i rispettivi centroidi.

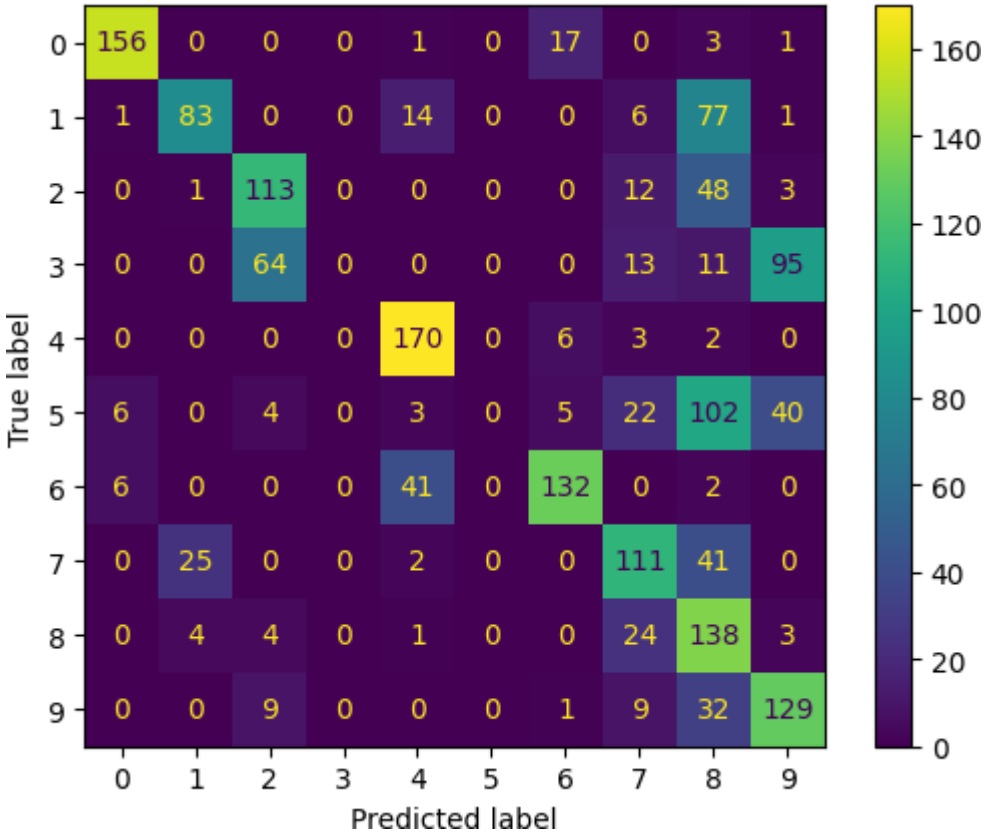
In [ ]:
```
# Calcolo del K-means e dei cluster finali
 def K_means(K, X, max_iterations = 500):

     centroids = init_rand_centroids(K, X)
     # print(f"initial centroids: {centroids}")

     # Loop fino a max_iterations o convergenza
     for _ in range(max_iterations):

         clusters = create_clusters(centroids, K, X)
         prev_centroids = centroids

         centroids = means(clusters, K, X)

         diff = prev_centroids - centroids

         if not diff.any():
             return clusters, centroids

     return clusters, centroids
```

3 Risultati

3.1 Clustering & Matrix Confusion

Il primo passo è la clusterizzazione senza uso della PCA. Viene anche prodotta una matrice di confusione, che dice quante volte il numero `i` è stato riconosciuto come il numero `j`.

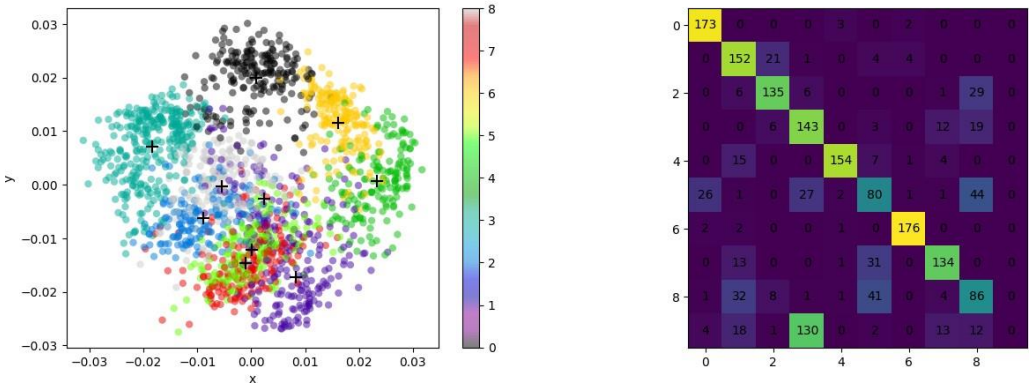
In [ ]:
```
K = 10

  # Applicazione del K-Means
  clusters, centroids = K_means(K, X, 500)

  #Per produrre la matrice di confusione è necessario riodinare le labe
  labels = np.zeros_like(clusters)
  for i in range(K):
      mask = (clusters == i)
      labels[mask] = mode(Y[mask])[0]

  # Confusion-Matrix
  ConfusionMatrixDisplay(confusion_matrix(Y,  labels)).plot()
```



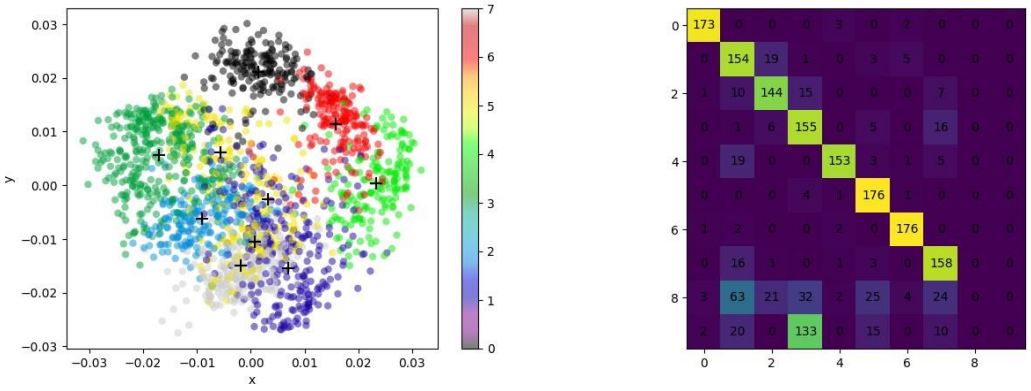
A questo punto si ripete il procedimento facendo uso della PCA, variando le componenti principali da 2 a 6.
Per visualizzare le variazioni dei cluster al variare del numero di componenti principali, si plottano le prime due dimensioni.
Nonostante non sia stato richiesto, risulta visivamente utile per capire la pipeline di elaborazione.

In [ ]:
```
K = 10

  for j in range (2,7):

      X, Y = digits.data, digits.target

      X = X/1000

      # PCA applying
      X = PCA(j).fit_transform(X)


      plt.subplots(figsize=(15,  5))
      plt.suptitle("PCA  =  "+str(j))
      plt.subplot(1, 2, 1)

      # K-Means & Cluster applying
      clusters, centroids = K_means(K, X, 500)

      labels  =  np.zeros_like(clusters)
      for i in range(K):
          mask = (clusters == i)
          labels[mask]  =  mode(Y[mask])[0]

      # K-Means & Cluster plotting
      plt.scatter(X[:,0],  X[:,1],
                  c=labels.astype(int),  edgecolor='none',  alpha=0.5,
                  cmap=plt.colormaps["nipy_spectral"])
      plt.colorbar()
      plt.xlabel('x')
      plt.ylabel('y')

      plt.scatter(centroids[:,0],  centroids[:,1],  marker="+",  s=100,  co


      # using subplot function and creating plot two
      plt.subplot(1, 2, 2)

      # Confusion-Matrix
      #ConfusionMatrixDisplay(confusion_matrix(Y,  labels))
      test = confusion_matrix(Y, labels)
      plt.imshow(test)

      for i in range(K):
          for j in range(K):
              c = test[j,i]
              plt.text(i, j, str(c), va='center', ha='center')

      plt.show()
```

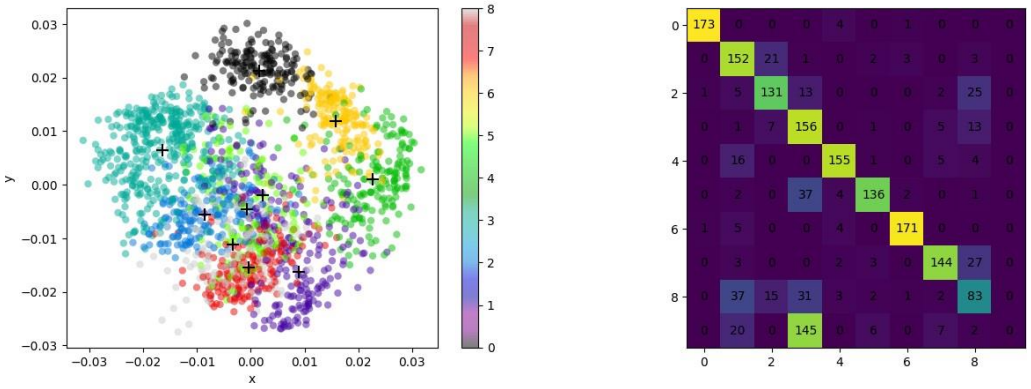
PCA = 4



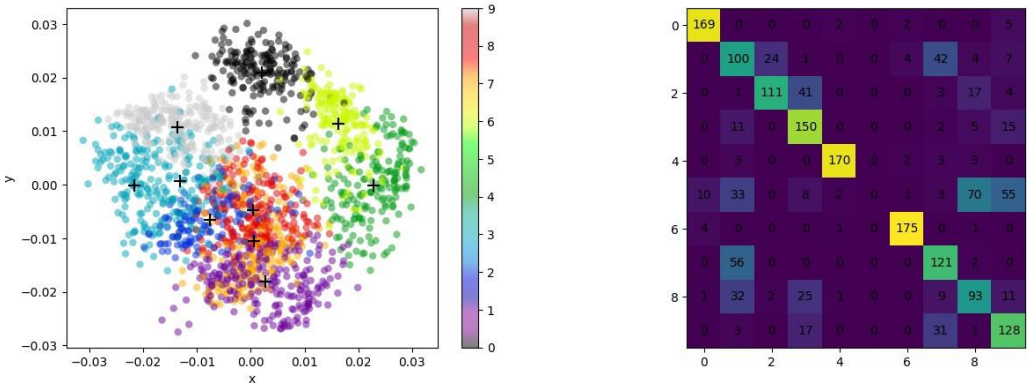
PCA = 5



PCA = 2
PCA = 6



PCA = 3



Si osservi che all'aumentare del numero di componenti principale, incrementala precisione con cui la macchina riesce a classificare i numeri; tuttavia, sfruttando dei metodi NON supervisionati, si avranno risultato diversi all'addestramento dato in input.

4 Extra

4.1 Plotting dei primi n numeri

In [ ]
```
A = digits.data

n = 50

for i in range(0,n,10):
    figure,axis  =  plt.subplots(1,10,figsize=(15,  5))
    B = A[i:i+10]

    for j in range(10):
        axis[j].imshow(B[j].reshape(8,8))

    plt.show()
```