K-Means

Dear Professor : Mr.Manthouri

Produced By : Ghasemi,morteza

https://github.com/Morteza-Ghasemi

January 2021

Introducing k-Means

- The *k*-means algorithm searches for a pre-determined number of clusters within an unlabeled multidimensional dataset. It accomplishes this using a simple conception of what the optimal clustering looks like:
  - The "cluster center" is the arithmetic mean of all the points belonging to the cluster.
  - Each point is closer to its own cluster center than to other cluster centers.

k-Means Algorithm: Expectation–Maximization

- Expectation–maximization (E–M) is a powerful algorithm that comes up in a variety of contexts within data science.
- he expectation–maximization approach here consists of the following procedure:
  - Guess some cluster centers
  - Repeat until converged
    - *E-Step*: assign points to the nearest cluster center
    - *M-Step*: set the cluster centers to the mean

k-Means Algorithm: Expectation–Maximization

- Expectation–maximization (E–M) is a powerful algorithm that comes up in a variety of contexts within data science.
- he expectation–maximization approach here consists of the following procedure:
  - Guess some cluster centers
  - Repeat until converged
    - *E-Step*: assign points to the nearest cluster center
    - *M-Step*: set the cluster centers to the mean

K-Means On Digits

- we will attempt to use *k*-means to try to identify similar digits *without using the original label information*;
- this might be similar to a first step in extracting meaning from a new dataset about which you don't have any *a priori* label information.
- We will start by loading the digits and then finding the Kmeans Clusters.

K-Means On Digits

- Recall that the digits consist of 1,797 samples with 64 features, where each of the 64 features is the brightness of one pixel in an 8×8 image:
  - from sklearn.datasets import load_digits
  - digits = load_digits()
  - digits.data.shape

K-Means On Digits

- The clustering can be performed as before:
  - kmeans = KMeans(n_clusters=10, random_state=0)
  - clusters = kmeans.fit_predict(digits.data)
  - kmeans.cluster_centers_.shape
- The result is 10 clusters in 64 dimensions. Notice that the cluster centers themselves are 64-dimensional points, and can themselves be interpreted as the "typical" digit within the cluster.

K-Means On Digits

- We see that *even without the labels*, kmeans is able to find clusters whose centers are recognizable digits, with perhaps the exception of 1 and 8.
- Because *k*-means knows nothing about the identity of the cluster, the 0–9 labels may be permuted.
  - fig, ax = plt.subplots(2, 5, figsize=(8, 3))
  - centers = kmeans.cluster_centers_.reshape(10, 8, 8)
  - for axi, center in zip(ax.flat, centers):

    axi.set(xticks=[], yticks=[])

    axi.imshow(center, interpolation='nearest', cmap=plt.cm.binary)

K-Means On Digits

- We can fix this by matching each learned cluster label with the true labels found in them:
  - from scipy.stats import mode
  - labels = np.zeros_like(clusters)
  - for i in range(10):

    mask = (clusters == i)

    labels[mask] = mode(digits.target[mask])[0]

K-Means On Digits

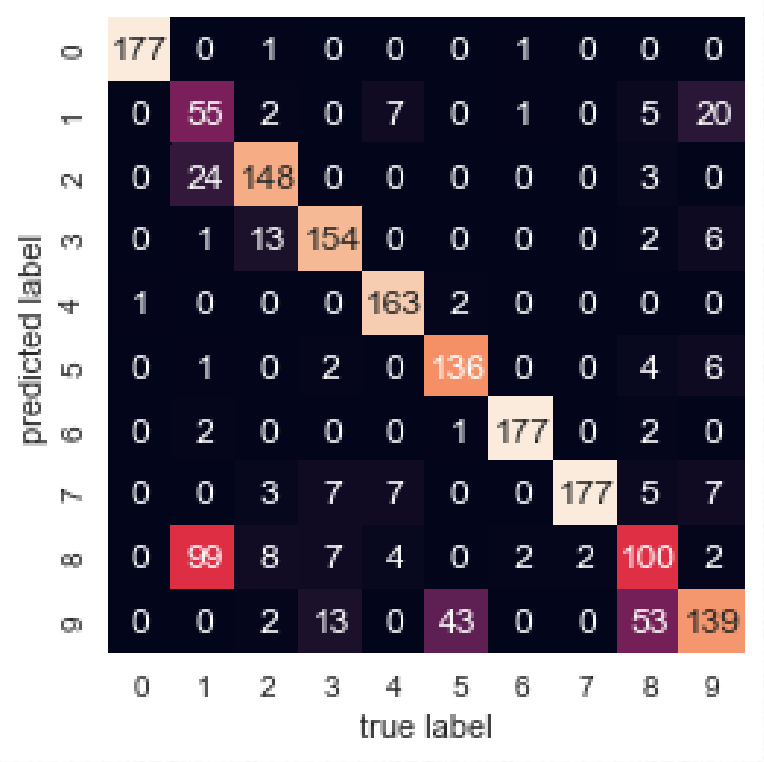
- Now we can check how accurate our unsupervised clustering was in finding similar digits within the data :
  - from sklearn.metrics import accuracy_score
  - accuracy_score(digits.target, labels)
- With just a simple *k*-means algorithm, we discovered the correct grouping for 80% of the input digits!

K-Means On Digits

- the confusion matrix for this:
  - from sklearn.metrics import confusion_matrix
  - mat = confusion_matrix(digits.target, labels)
  - sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
  - xticklabels=digits.target_names,
  - yticklabels=digits.target_names)
  - plt.xlabel('true label')
  - plt.ylabel('predicted label');

K-Means On Digits

- the confusion matrix for this:



K-Means On Digits

- the confusion matrix for this:
  - As we might expect from the cluster centers we visualized before, the main point of confusion is between the eights and ones.
  - But this still shows that using *k*-means, we can essentially build a digit classifier *without reference to any known labels*!

K-Means On Digits

- We can use the t-distributed stochastic neighbor embedding (t-SNE) algorithm to pre-process the data before performing *k*-means.
- t-SNE is a nonlinear embedding algorithm that is particularly adept at preserving points within clusters.

K-Means On Digits

- see how it does:
  - from sklearn.manifold import TSNE
  - # Project the data: this step will take several seconds
  - tsne = TSNE(n_components=2, init='random', random_state=0)
  - digits_proj = tsne.fit_transform(digits.data)
  - # Compute the clusters
  - kmeans = KMeans(n_clusters=10, random_state=0)
  - clusters = kmeans.fit_predict(digits_proj)

K-Means On Digits

- see how it does:
  - # Permute the labels
  - labels = np.zeros_like(clusters)
  - for i in range(10):

    mask = (clusters == i)

    labels[mask] = mode(digits.target[mask])[0]
  - # Compute the accuracy
  - accuracy_score(digits.target, labels)

K-Means On Digits

- Compute the accuracy :
  - 0.93489148580968284

- That's nearly 93% classification accuracy *without using the labels*.
- This is the power of unsupervised learning when used carefully: it can extract information from the dataset that it might be difficult to do by hand or by eye.

The End