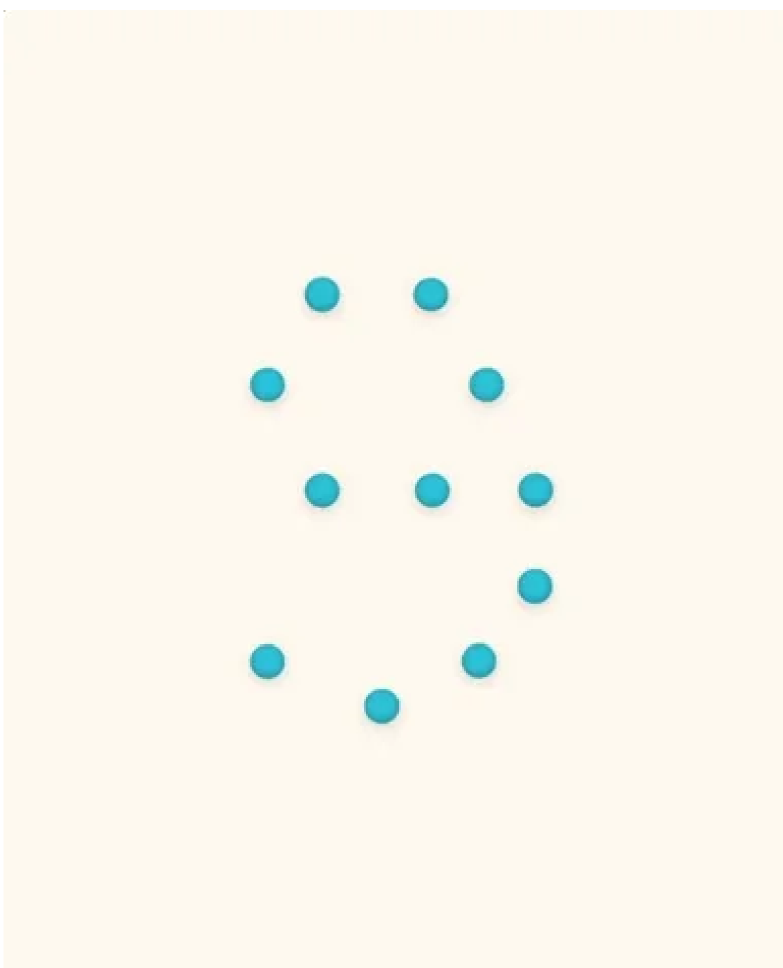
K Means Clustering for Imagery Analysis

Sajjad Salaria Follow
Aug 22, 2019 · 4 min read

Let's learn about K-Means by doing a mini-project.

In this project, we will use a K-means algorithm to perform image classification. Clustering isn't limited to the consumer information and population sciences, it can be used for imagery analysis as well. Leveraging Scikit-learn and the MNIST dataset, we will investigate the use of K-means clustering for computer vision.



https://giphy.com/gifs/l0HlNBe9Z3Xz9x1Yc/html5

In this project, we will learn how to:

- Preprocess images for clustering
- Deploy K-means clustering algorithms
- Use common metrics to evaluate cluster performance
- Visualize high-dimensional cluster centroids

Let's get started by importing a few of the libraries we will use in this project.

```
import sys
import sklearn
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
```

```
%matplotlib inline

print(‘Python: {}’.format(sys.version))
print(‘Sklearn: {}’.format(sklearn.version))
print(‘Matplotlib: {}’.format(matplotlib.version))
print(‘NumPy: {}’.format(np.version))
```

import.py hosted with ♡ by GitHub view raw

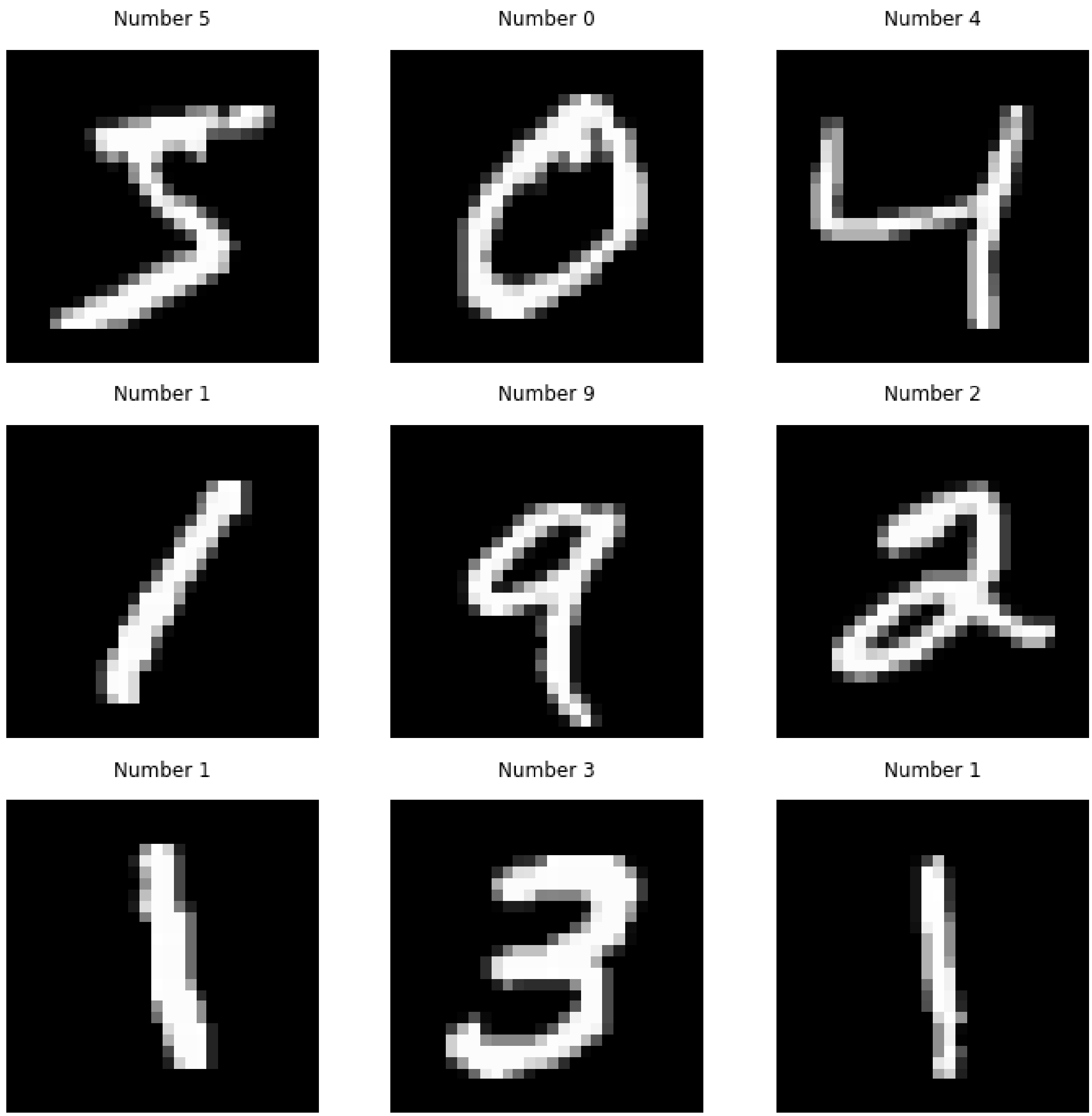
1. Import the MNIST dataset

For this project, we will be using the MNIST dataset. It is available through keras, a deep learning library we have used in previous tutorials. Although we won’t be using other features of keras today, it will save us time to import mnist from this library. It is also available through the tensorflow library or for download at http://yann.lecun.com/exdb/mnist/.

```
from keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()

print(‘Training Data: {}’.format(x_train.shape))
print(‘Training Labels: {}’.format(y_train.shape))

Training Data: (60000L, 28L, 28L)
Training Labels: (60000L,)

print(‘Testing Data: {}’.format(x_test.shape))
print(‘Testing Labels: {}’.format(y_test.shape))

Testing Data: (10000L, 28L, 28L)
Testing Labels: (10000L,)

# EDA

fig, axs = plt.subplots(3, 3, figsize = (12, 12))
plt.gray()

for i, ax in enumerate(axs.flat):
ax.matshow(x_train[i])
ax.axis(‘off’)
ax.set_title(‘Number {}’.format(y_train[i]))

```

```
fig.show()
```

Dataset.py hosted with ♡ by GitHub view raw



2. Preprocessing the MNIST images

Images stored as NumPy arrays are 2-dimensional arrays. However, the K-means clustering algorithm provided by scikit-learn ingests 1-dimensional arrays; as a result, we will need to reshape each image.

Clustering algorithms almost always use 1-dimensional data. For example, if you were clustering a set of X, Y coordinates, each point would be passed to the clustering algorithm as a 1-dimensional array with a length of two (example: [2,4] or [-1, 4]). If you were using 3-dimensional data, the array would have a length of 3 (example: [2, 4, 1] or [-1, 4, 5]).

MNIST contains images that are 28 by 28 pixels; as a result, they will have a length of 784 once we reshape them into a 1-dimensional array.

```
# convert each image to 1 dimensional array

X = x_train.reshape(len(x_train),-1)
Y = y_train

# normalize the data to 0 - 1

X = X.astype(float) / 255.

print(X.shape)
print(X[0].shape)

(60000L, 784L)
(784L,)
```

Pre-Processing.py hosted with ♡ by GitHub view raw

3. K-Means Clustering

Time to start clustering! Due to the size of the MNIST dataset, we will use the mini-batch implementation of k-means clustering provided by scikit-learn. This will dramatically reduce the amount of time it takes to fit the algorithm to the data.

The MNIST dataset contains images of the integers 0 to 9. Because of this, let's start by setting the number of clusters to 10, one for each digit.

```
from sklearn.cluster import MiniBatchKMeans

n_digits = len(np.unique(y_test))
print(n_digits)

# Initialize KMeans model
```

```python

kmeans = MiniBatchKMeans(n_clusters = n_digits)

# Fit the model to the training data

kmeans.fit(X)

kmeans.labels_
```

K-means.py hosted with ♡ by GitHub view raw

4. Assigning Cluster Labels

K-means clustering is an unsupervised machine learning method; consequently, the labels assigned by our KMeans algorithm refer to the cluster each array was assigned to, not the actual target integer. To fix this, let's define a few functions that will predict which integer corresponds to each cluster.

```python
def infer_cluster_labels(kmeans, actual_labels):
  inferred_labels = {}

for i in range(kmeans.n_clusters):

    # find index of points in cluster
    labels = []
    index = np.where(kmeans.labels_ == i)

    # append actual labels for each point in cluster
    labels.append(actual_labels[index])

    # determine most common label
    if len(labels[0]) == 1:
        counts = np.bincount(labels[0])
    else:
        counts = np.bincount(np.squeeze(labels))

    # assign the cluster to a value in the inferred_labels dictionary
    if np.argmax(counts) in inferred_labels:
        # append the new number to the existing array at this slot
        inferred_labels[np.argmax(counts)].append(i)
    else:
        # create a new array in this slot
        inferred_labels[np.argmax(counts)] = [i]
```

```

    #print(labels)
    #print('Cluster: {}, label: {}'.format(i, np.argmax(counts)))

return inferred_labels

def infer_data_labels(X_labels, cluster_labels):
  # empty array of len(X)
predicted_labels = np.zeros(len(X_labels)).astype(np.uint8)

for i, cluster in enumerate(X_labels):
    for key, value in cluster_labels.items():
        if cluster in value:
            predicted_labels[i] = key

return predicted_labels

# test the infer_cluster_labels() and infer_data_labels() functions

cluster_labels = infer_cluster_labels(kmeans, Y)
X_clusters = kmeans.predict(X)
predicted_labels = infer_data_labels(X_clusters, cluster_labels)
print predicted_labels[:20]
print Y[:20]
```

5. Optimizing and Evaluating the Clustering Algorithm

With the functions defined above, we can now determine the accuracy of our algorithms. Since we are using this clustering algorithm for classification, accuracy is ultimately the most important metric; however, there are other metrics out there that can be applied directly to the clusters themselves, regardless of the associated labels. Two of these metrics that we will use are inertia and homogeneity.

```
# Initialize and fit KMeans algorithm
kmeans = MiniBatchKMeans(n_clusters = 36)
kmeans.fit(X)

# record centroid values
centroids = kmeans.cluster_centers_
```

```python

# reshape centroids into images
images = centroids.reshape(36, 28, 28)
images *= 255
images = images.astype(np.uint8)

# determine cluster labels
cluster_labels = infer_cluster_labels(kmeans, Y)

# create figure with subplots using matplotlib.pyplot
fig, axs = plt.subplots(6, 6, figsize = (20, 20))
plt.gray()

# loop through subplots and add centroid images
for i, ax in enumerate(axs.flat):

    # determine inferred label using cluster_labels dictionary
    for key, value in cluster_labels.items():
        if i in value:
            ax.set_title('Inferred Label: {}'.format(key))

    # add image to subplot
    ax.matshow(images[i])
    ax.axis('off')

# display the figure
fig.show()
```

ClusteringAlgorithmVI.py hosted with ♡ by GitHub view raw

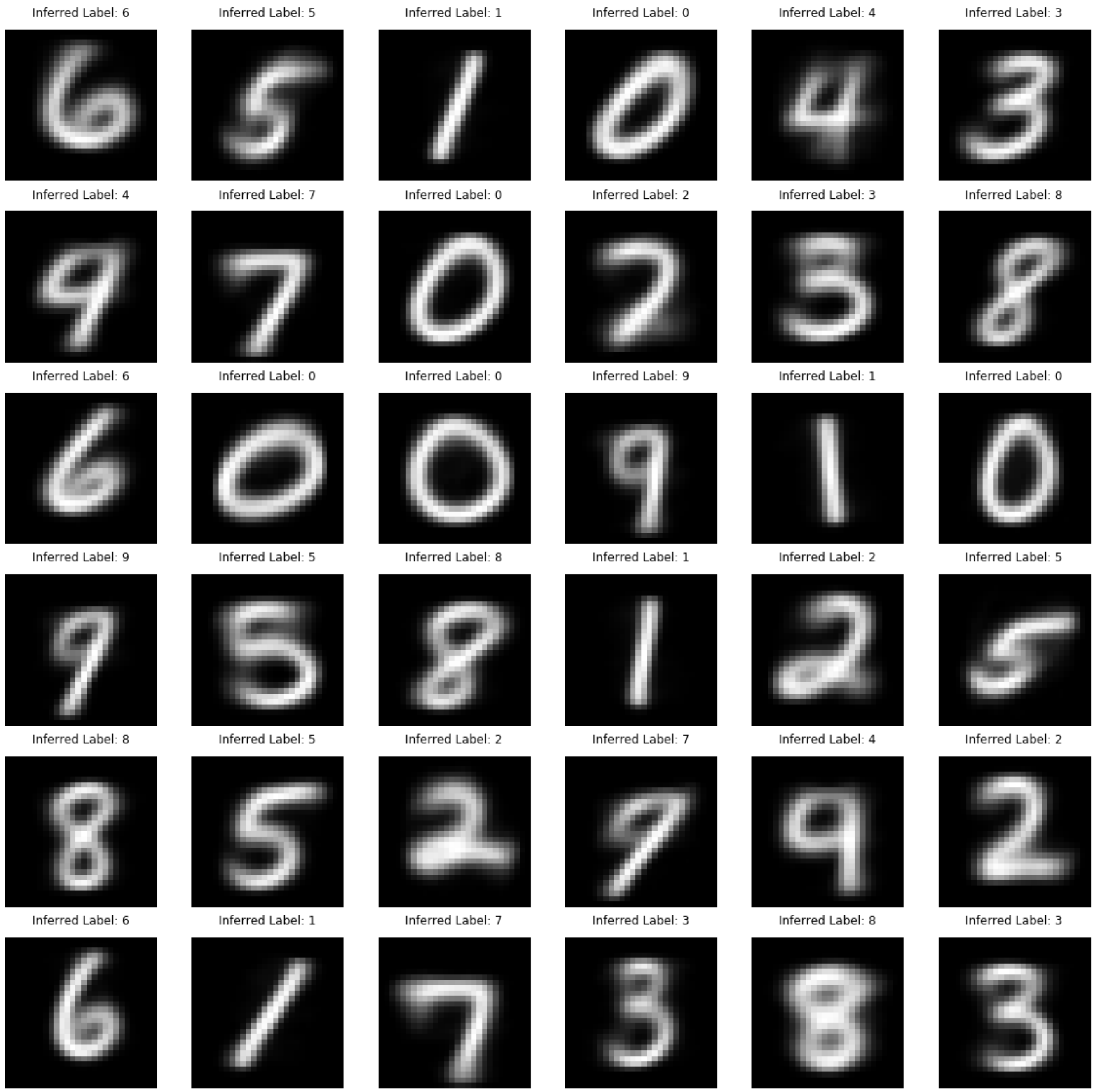
Furthermore, earlier we made the assumption that K = 10 was the appropriate number of clusters; however, this might not be the case. Let's fit the K-means clustering algorithm with several different values of K, than evaluate the performance using our metrics.

6. Visualizing Cluster Centroids

The most representative point within each cluster is called the centroid. If we were dealing with X,Y points, the centroid would simply be a point on the graph. However, since we are using arrays of length 784, our centroid is also going to be an array of length 784. We can reshape this array back into a 28 by 28-pixel image and plot it.

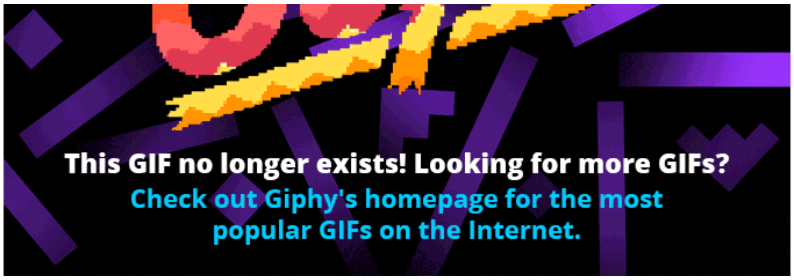
These graphs will display the most representative image for each cluster.

```
print('Accuracy: {}\n'.format(metrics.accuracy_score(y_test, predicted_labels)))
```



Predictions — Accuracy: 0.9023

For Code: refer -> https://github.com/xoraus/K-Means-Clustering-for-Imagery-Analysis



https://giphy.com/gifs/ncK1aJlwmpHNu/html5

References:

[1]: https://en.wikipedia.org/wiki/K-means_clustering

[2]: http://yann.lecun.com/exdb/mnist/

[3]: https://en.wikipedia.org/wiki/MNIST_database

[4]: https://www.kaggle.com/ngbolin/mnist-dataset-digit-recognizer

Machine Learning Computer Science Deeplearing Digits Recognition K Means Clustering