Machine Learning

Assignment #2 (Unsupervised Learning)

Prof. Eunwoo Kim

Submit all codes written in Python to run the following algorithms and a pdf file that describes your experimental setup and results.

Algorithms to be implemented:

- ✓ Dimension reduction: PCA
- ✓ Clustering: K-means, DBSCAN

Dataset

- ✓ Fashion MNIST (10 classes, 60,000 training images and 10,000 test images)
  - 28×28 grayscale images.
  - Use the “test images” only for this assignment.

What you have to submit (through e-class)

- ✓ A zip file containing
  - A folder including implementation codes written in Python for the algorithms.
  - A report (pdf file) describing experimental setup and results on the test set.

■ Data and Evaluation Metrics

① First, import the packages needed for learning.

- About pytorch : torch · PyPI
- About numpy : numpy · PyPI
- About pandas : pandas · PyPI
- About sklearn : 2. Unsupervised learning — scikit-learn 1.2.2 documentation

```
import numpy as np
from torch.utils.data import DataLoader
import pandas as pd
from sklearn import metrics
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt
import seaborn as sns
from torchvision import datasets, transforms
```

② The next step involves loading the data will be used for learning.

- Fashion MNIST dataset : https://www.kaggle.com/datasets/zalando-research/fashionmnist
- It extracts the input images and their labels from the dataloaders and concatenates them into numpy arrays, FM_test_images, and FM_test_labels. The images are flattened into vectors and normalized to have values between 0 and 1. These numpy arrays can be used for training and testing machine learning models. (Note that we only use test dataset)

```
Fashion_mnist_test_transform = transforms.Compose([transforms.ToTensor()])

testset_Fashion_mnist = datasets.FashionMNIST(root='./data', train=False, download=True, transform=Fashion_mnist_test_transform)

FM_test = DataLoader(testset_Fashion_mnist, batch_size=32, shuffle=False, num_workers=2)

FM_test_images = []
FM_test_labels = []

for batch in FM_test:
    images, labels = batch
    images_flat = images.view(images.shape[0], -1)
    FM_test_images.append(images_flat.numpy())
    FM_test_labels.append(labels.numpy())
FM_test_images = np.vstack(FM_test_images)
FM_test_labels = np.concatenate(FM_test_labels)
```

- Once the data has been loaded, it can be converted into pandas DataFrame and Series for use in learning algorithms. While unsupervised learning algorithms do not require labels, they can still be use for visualization purposes.

```
X_ = pd.DataFrame(data=FM_test_images) # test data
y_ = pd.Series(data=FM_test_labels) # test label
```

■ PCA for dimension reduction

The following code implements PCA on the Fashion MNIST test dataset and visualizes it.

✓ Initially, a PCA object is created with set "n_components" as 50 components and applied to the data stored in the DataFrame X using the “fit_transform()” function. The resulting transformed data is stored in a new DataFrame called “test_PCA”.

✓ Subsequently, a new DataFrame named “testDF” is created, containing the two principal components represented by the "x-axis" and "y-axis" of the test_PCA DataFrame, along with their corresponding labels denoted as "Label".

✓ Finally, the Seaborn library's “lmplot” function is created a scatter plot of the first two principal components. The hue parameter is set to the dataset label to visualize the separation of classes using only these two components. The plot is displayed with a grid.

```
pca = PCA(n_components= 50)
test_PCA = pca.fit_transform(X_)
test_PCA = pd.DataFrame(data = test_PCA)

testDF = pd.DataFrame(data=test_PCA.loc[:,0:1], index=test_PCA.index)
testDF = pd.concat((testDF,y_), axis=1, join="inner")
testDF.columns = ["x-axis", "y-axis", "Label"]
sns.lmplot(x="x-axis", y="y-axis", hue="Label", data=testDF, fit_reg=False, height=8)
plt.grid()
```

■ t-SNE for visualization

The following code uses t-SNE to visualize the test dataset after applying PCA.

✓ First, a t-SNE object is created with hyperparameters and stored in the variable "tSNE". The t-SNE object is then applied to the "test_PCA" dataset. A new DataFrame called "testDF" is created, which includes the two principal components denoted "x-axis" and "y-axis" of the test_PCA along with the corresponding labels denoted "Label".

✓ Next, the k-means algorithm is applied to the "X_test_tSNE" input, and a new DataFrame is created with the resulting cluster labels denoted as "labels".

✓ Finally, Seaborn's "lmplot" function is used to visualize the data, with the color of each point representing its cluster. The "hue" parameter is set to "Label" to automatically color each point based on its label.

```
n_components = 2
learning_rate = 300
perplexity = 30
early_exaggeration = 12
init = 'random'

tSNE = TSNE(n_components=n_components, learning_rate=learning_rate,
            perplexity=perplexity, early_exaggeration=early_exaggeration, init=init)
```

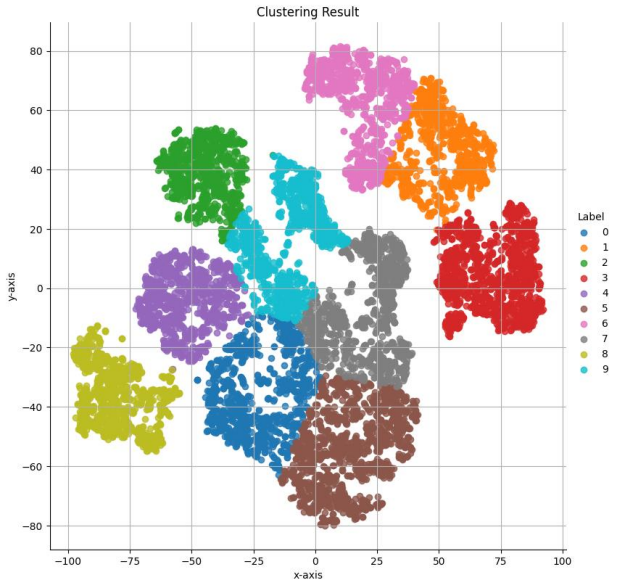
```
X_test_tSNE = tSNE.fit_transform(test_PCA.loc[:,:])
X_test_tSNE = pd.DataFrame(data=X_test_tSNE)
testDF = pd.DataFrame(data=X_test_tSNE.loc[:,:], index=test_PCA.index)
testDF = pd.concat((testDF,y_), axis=1, join="inner")
testDF.columns = ["x-axis", "y-axis", "Label"]

################### This part should include K-means algorithm ###################

##################################################################################
testDF["Label"] = labels
sns.lmplot(x="x-axis", y="y-axis", hue="Label", data=testDF, fit_reg=False, height=8)
plt.title("Clustering Result")
plt.grid()
```

What you have to do:

- Implement **k-means** (you should manually set hyperparameters involved).
  - You may use KMeans in the library or you can write your own code from scratch.
  - Perform PCA to reduce the dimension before running the k-means algorithm (set dimension to 100, 50, and 10, respectively).
    - Also run k-means for the original images (dimension of 784).
  - Set k (the number of clusters) to 10 (total number of classes in the dataset).
  - After running k-means with reduced dimensions, plot the 100 random chosen samples from the dataset using t-SNE (example below) – see above how to use t-SNE.
    - If you specify the "hue" parameter as "Label" when using the "Implot" function, the function will automatically assign a color to each point based on its corresponding label.
    - Show visualization results after k-means with four dimensions (784, 100, 50, 10).



  - Compute the clustering results using ARI (Adjusted Rand Index) for the four different dimensions (784, 100, 50, 10). Show results on test set (not train set) in a table.
    - The ARI function is calculated by utilizing the actual labels and clustering labels using Sklearn's metrics package.
- Implement **DBSCAN** in a similar way to k-means.
  - You may use DBSCAN in the library or you can write your own code from scratch.
  - Set the hyperparameters (eps, minNeighbors).
  - Perform DBSCAN for four different dimensions (784, 100, 50, 10) and show clustering results on test set in a table and visualization results.
- Please note that using the Colab environment can make it convenient to implement and experiment with algorithms.

- The experimental setup refers to the experimental environment, which includes the dataset used, the algorithms employed, and the hyper-parameters utilized.

Inquiry (via email)

- Class-01: Jungkyoo Shin (neo293@cau.ac.kr)
- Class-02: Jiho Lee (j2hoooo@cau.ac.kr)