BU.330.775 Machine Learning: Design and Deployment

**Lab 5. T-SNE for visualization on MNIST dataset**

Learning Goal: practice PCA and t-SNE for various visualization tasks on the MNIST database

Background: This example is curated from Geron (2022). Please refer to Lab 3 instructions for information about the MNIST dataset.

1. We begin by importing Matplotlib, and set the global style of the Matplotlib chart.

import matplotlib.pyplot as plt

plt.rc('font', size=14)

plt.rc('axes', labelsize=14, titlesize=14)

plt.rc('legend', fontsize=14)

plt.rc('xtick', labelsize=10)

plt.rc('ytick', labelsize=10)

1. First, we perform dimensionality reduction on the MNIST dataset using principal component analysis (PCA), with the goal of finding the minimum number of principal components that explains 95% of the variance in the data. Load the MNIST data, split it into a training set and a test set, then fit the training data through PCA, calculate the cumulative explained variance ratio, and find the minimum principal component number required to explain 95% of the variance, thereby reducing the data dimension, retain the main information.

from sklearn.decomposition import PCA

import numpy as np

from sklearn.datasets import fetch\_openml

mnist = fetch\_openml('mnist\_784', as\_frame=False, parser="auto")

X\_train, y\_train = mnist.data[:60\_000], mnist.target[:60\_000]

X\_test, y\_test = mnist.data[60\_000:], mnist.target[60\_000:]

pca = PCA(n\_components=0.95)

X\_reduced = pca.fit\_transform(X\_train)

pca.n\_components\_

1. We plot the cumulative explained variance as the number of dimensions increases. The horizontal axis represents the number of principal components (number of dimensions), and the vertical axis represents the cumulative explained variance.

pca = PCA()

pca.fit(X\_train)

cumsum = np.cumsum(pca.explained\_variance\_ratio\_)

plt.figure(figsize=(6, 4))

plt.plot(cumsum, linewidth=3)

plt.axis([0, 400, 0, 1])

plt.xlabel("Dimensions")

plt.ylabel("Explained Variance")

plt.grid(True)

plt.show()

1. Now we use PCA to compress the data to preserve 80% of the variance, and then approximately restore the dimensionally reduced data to the original space to demonstrate the compression and recovery effects of PCA. Here, inverse\_transform() function transforms data back to its original space. The original image is shown on the left and the compressed and restored image is shown on the right. In this way, the quality changes of the compressed image can be visually observed, verifying the effect of PCA compression in reducing the data dimension while retaining the main features.

pca = PCA(0.80)

X\_reduced = pca.fit\_transform(X\_train, y\_train)

X\_recovered = pca.inverse\_transform(X\_reduced)

plt.figure(figsize=(7, 4))

for idx, X in enumerate((X\_train[::2100], X\_recovered[::2100])):

plt.subplot(1, 2, idx + 1)

plt.title(["Original", "Compressed"][idx])

for row in range(5):

for col in range(5):

plt.imshow(X[row \* 5 + col].reshape(28, 28), cmap="binary",

vmin=0, vmax=255, extent=(row, row + 1, col, col + 1))

plt.axis([0, 5, 0, 5])

plt.axis("off")

1. Now, let’s load a smaller digits dataset for visualization. Using PCA with the first two principal components, we generate a scatterplot. Each data point is represented by the corresponding digit, and categories are distinguished using different colors. The horizontal and vertical axes represent the first and second principal components, respectively. This visualization method helps observe the distribution of each digit category in the reduced-dimensional space and reveals data clustering and the relationships between categories.

from sklearn.datasets import load\_digits

digits = load\_digits()

# build a PCA model

pca = PCA(n\_components=2)

pca.fit(digits.data)

# transform the digits data onto the first two principal components

digits\_pca = pca.transform(digits.data)

colors = ["#476A2A", "#7851B8", "#BD3430", "#4A2D4E", "#875525",

"#A83683", "#4E655E", "#853541", "#3A3120", "#535D8E"]

plt.figure(figsize=(10, 10))

plt.xlim(digits\_pca[:, 0].min(), digits\_pca[:, 0].max())

plt.ylim(digits\_pca[:, 1].min(), digits\_pca[:, 1].max())

for i in range(len(digits.data)):

# actually plot the digits as text instead of using scatter

plt.text(digits\_pca[i, 0], digits\_pca[i, 1], str(digits.target[i]),

color = colors[digits.target[i]],

fontdict={'weight': 'bold', 'size': 9})

plt.xlabel("First principal component")

plt.ylabel("Second principal component")

1. Lastly, let’s try t-SNE.

from sklearn.manifold import TSNE

tsne = TSNE(random\_state=42)

# use fit\_transform instead of fit, as TSNE has no transform method

digits\_tsne = tsne.fit\_transform(digits.data)

plt.figure(figsize=(10, 10))

plt.xlim(digits\_tsne[:, 0].min(), digits\_tsne[:, 0].max() + 1)

plt.ylim(digits\_tsne[:, 1].min(), digits\_tsne[:, 1].max() + 1)

for i in range(len(digits.data)):

# actually plot the digits as text instead of using scatter

plt.text(digits\_tsne[i, 0], digits\_tsne[i, 1], str(digits.target[i]),

color = colors[digits.target[i]],

fontdict={'weight': 'bold', 'size': 9})

plt.xlabel("t-SNE feature 0")

plt.ylabel("t-SNE feature 1")

**Submission:** Complete and submit on Canvas by the beginning of Class 6. Use homework5mnist\_yourname.ipynb as the file name.