BU.330.775 Machine Learning: Design and Deployment

**Lab 5. Dimensionality reduction on breast cancer dataset**

Learning Goal: practice dimensionality reduction approaches on the Diagnostic Wisconsin Breast Cancer Database

Background: This example is curated from Muller and Guido (2016). Please refer to Lab 4 instructions for information about the Wisconsin breast cancer dataset.

1. We begin by importing libraries such as Matplotlib, NumPy, and Scikit-learn. And load the breast cancer dataset.

import matplotlib.pyplot as plt

import numpy as np

from sklearn.datasets import load\_breast\_cancer

cancer = load\_breast\_cancer()

1. To refresh your memory about the dataset, let’s check the feature names and the labels.

# Print the feature names

print("Feature names:", cancer.feature\_names)

# Print the target names

print("Target names:", cancer.target\_names)

1. Explore the first two features, mean radius and mean texture, by using the scatter plot and different colors for the two classes

# Extract the mean radius feature and target names

mean\_radius = cancer.data[:, 0]

target\_names = cancer.target\_names

target = cancer.target

# Create the scatter plot

plt.figure(figsize=(8, 6))

for i in range(len(target\_names)):

plt.scatter(np.where(target==i)[0], mean\_radius[target==i], label=target\_names[i])

plt.xlabel("Index")

plt.ylabel("Mean Radius")

plt.title("Mean Radius Plot")

plt.legend()

plt.show()

# Extract the mean texture feature and target names

mean\_texture = cancer.data[:, 1]

# Create the scatter plot

plt.figure(figsize=(8, 6))

for i in range(len(target\_names)):

plt.scatter(np.where(target==i)[0], mean\_texture[target==i], label=target\_names[i])

plt.xlabel("Index")

plt.ylabel("Mean Texture")

plt.title("Mean Texture Plot")

plt.legend()

plt.show()

1. Now we can visualize the distribution of benign and malignant tumors in the breast cancer dataset for each feature by plotting a histogram. It creates a sub-graph layout with 15 rows and 2 columns, with red and green representing malignant and benign samples respectively, so as to observe the difference of each feature under different categories. This visualization helps to analyze which features are more discriminative for distinguishing benign and malignant tumors.

fig, axes = plt.subplots(15, 2, figsize=(10, 20))

malignant = cancer.data[cancer.target == 0]

benign = cancer.data[cancer.target == 1]

ax = axes.ravel()

for i in range(30):

\_, bins = np.histogram(cancer.data[:, i], bins=50)

ax[i].hist(malignant[:, i], bins=bins, color='red', alpha=.5)

ax[i].hist(benign[:, i], bins=bins, color='green', alpha=.5)

ax[i].set\_title(cancer.feature\_names[i])

ax[i].set\_yticks(())

ax[0].set\_xlabel("Feature magnitude")

ax[0].set\_ylabel("Frequency")

ax[0].legend(["malignant", "benign"], loc="best")

fig.tight\_layout()

1. Next, we will examine the correlations between different features by generating a heatmap of all features. To create a better visualization, we will use the pandas and seaborn packages.

import pandas as pd

import seaborn as sns

# Create a Pandas DataFrame from the cancer dataset

df = pd.DataFrame(cancer.data, columns=cancer.feature\_names)

# Calculate the correlation matrix

corr = df.corr()

plt.figure(figsize=(20,20))

sns.heatmap(corr, cmap=sns.color\_palette("ch:s=-.2,r=.6", as\_cmap=True),annot=True)

plt.show()

1. Before we apply PCA, we use `StandardScaler` to standardize the breast cancer dataset. Specifically, it first instantiates `StandardScaler`, then calculates the mean and standard deviation of the data through the `fit` method, and finally uses the `transform` method to standardize the data so that the mean of each feature is 0 and the standard deviation is 1. This step helps to eliminate the influence of different feature scales in subsequent analysis.

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(cancer.data)

X\_scaled = scaler.transform(cancer.data)

1. First, we use PCA to reduce the dimension of the breast cancer dataset to two principal components for easy visualization. It first instantiates PCA and sets it to retain two principal components, and then fits the PCA model with the standardized data. Next, the code transforms the data to these two principal components and prints the shape of the original data (569, 30) and the reduced shape (569, 2), indicating that the data dimension is reduced from 30 dimensions to 2 dimensions.

from sklearn.decomposition import PCA

# keep the first two principal components of the data

pca = PCA(n\_components=2)

# fit PCA model to beast cancer data

pca.fit(X\_scaled)

# transform data onto the first two principal components

X\_pca = pca.transform(X\_scaled)

print("Original shape: {}".format(str(X\_scaled.shape)))

print("Reduced shape: {}".format(str(X\_pca.shape)))

1. Check the explained variance ratios for the two principal components.

pca.explained\_variance\_ratio\_

1. We define a `discrete\_scatter` function to draw scatter plots of the first and second principal components and color the points according to the category. Different markers and colors are used in the function to represent each category, and the data of different categories are separated and plotted through a loop. After drawing, adjust the coordinate axis range to increase the readability of the image. Finally, call the `discrete\_scatter` function to display the PCA-transformed data as a two-dimensional scatter plot by category, use different colors to distinguish malignant and benign tumors, and add legends and labels. This visualization helps to observe the distribution of data in the first two principal component spaces.

import matplotlib as mpl

# plot first vs. second principal component, colored by class

def discrete\_scatter(x1, x2, y=None, markers=None, s=10, ax=None,

labels=None, padding=.2, alpha=1, c=None, markeredgewidth=None):

ax = plt.gca()

unique\_y = np.unique(y)

markers = ['o', '^', 'v', 'D', 's', '\*', 'p', 'h', 'H', '8', '<', '>'] \* 10

labels = unique\_y

lines = []

current\_cycler = mpl.rcParams['axes.prop\_cycle']

for i, (yy, cycle) in enumerate(zip(unique\_y, current\_cycler())):

mask = y == yy

color = cycle['color']

lines.append(ax.plot(x1[mask], x2[mask], markers[i], markersize=s,

label=labels[i], alpha=alpha, c=color))

pad1 = x1.std() \* 0.2

pad2 = x2.std() \* 0.2

xlim = ax.get\_xlim()

ylim = ax.get\_ylim()

ax.set\_xlim(min(x1.min() - pad1, xlim[0]), max(x1.max() + pad1, xlim[1]))

ax.set\_ylim(min(x2.min() - pad2, ylim[0]), max(x2.max() + pad2, ylim[1]))

return lines

plt.figure(figsize=(8, 8))

discrete\_scatter(X\_pca[:, 0], X\_pca[:, 1], cancer.target)

plt.legend(cancer.target\_names, loc="best")

plt.gca().set\_aspect("equal")

plt.xlabel("First principal component")

plt.ylabel("Second principal component")

1. The weights of the two principal components are printed to show the contribution of each feature to the principal component. This information helps us understand how PCA maps the original data to the principal component space and highlights the role of different features in the reduced data.

print("PCA components:\n{}".format(pca.components\_))

1. We then use a heat map to show the contribution of each feature to the first two principal components. It uses the matshow function to plot the PCA component matrix, with columns representing the 30 original features, rows representing the two principal components, and the color depth representing the weight of each feature in the corresponding principal component. The legend (colorbar) shows the range of weight values, with the horizontal axis label being the feature name and the vertical axis being the principal component. This visualization helps us understand which features have a greater impact on the first two principal components, thereby more intuitively explaining the performance of the data after dimensionality reduction.

plt.matshow(pca.components\_, cmap='viridis')

plt.yticks([0, 1], ["First component", "Second component"])

plt.colorbar()

plt.xticks(range(len(cancer.feature\_names)),

cancer.feature\_names, rotation=60, ha='left')

plt.xlabel("Feature")

plt.ylabel("Principal components")

1. Next, we will generate a PCA with 95% variance explained.

pca = PCA(n\_components=0.95) # keep components that explain 95% of variance

pca.fit(X\_scaled)

X\_pca = pca.transform(X\_scaled)

print("Original shape: {}".format(str(X\_scaled.shape)))

print("Reduced shape: {}".format(str(X\_pca.shape)))

print("Total explained variance: {}".format(str(sum(pca.explained\_variance\_ratio\_))))

**Homework Question (7pt):** Compare the performance of an SGD classifier before and after applying PCA.  
First, in a text cell, describe your design or the steps, including any data preprocessing you will perform.  
Next, implement the Python code, focusing only on the accuracy measure.  
Finally, in another text cell, report the accuracies before and after PCA and evaluate which approach performs better.

**Submission:** Complete and submit on Canvas by the beginning of Class 6. Use homework5bc\_yourname.ipynb as the file name.