COMP3009/ COMP4139 Machine Learning

Lab 2 – Feature Analysis

Dr Xin Chen, Autumn Semester, 2024

1. Introduction

In this lab, we will learn how to use Python to load a data spreadsheet of a real world example (breast cancer prediction). It is very important to understand, visualise and perform some feature pre-processing before doing machine learning modelling. It is often a good practice to perform feature selection and dimensionality reduction in machine learning applications.

2. Tutorial

You need to install **matplotlib**, **numpy**, **pandas**, **seaborn**, **scipy**, **sklearn**, before starting this tutorial. Check lab 1 if you don't know how to install these APIs.

2.1 Load WDBC dataset and data visualisation

- **Dataset**: The Breast Cancer dataset (WDBC) is available in a machine learning repository maintained by the University of California, Irvine. The dataset contains 569 samples of malignant and benign tumour cells. The first two columns in the dataset store the unique ID numbers of the samples and the corresponding diagnosis (M=malignant, B=benign), respectively. The columns 3-32 contain 30 real-value features that have been computed from digitised images of the cell nuclei, which can be used to build a model to predict whether a tumour is benign or malignant.
  https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic).

  Please download the "WDBC.csv" file from Moodle and save it to the same directory as your Python file. Then go through the code below by running them block by block and see the results. Questions in red are for you to think about.

- **Load WDBC data and display the data structure. Check the dataset document to see if you understand the meaning of each feature.**

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random
import seaborn as sns
import scipy
from scipy.stats import pearsonr

import sklearn
```

```
from sklearn import datasets, linear_model
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

all_df=pd.read_csv('WDBC.csv', index_col=False)
all_df.head()
```

- **Drop the ID column**

```
all_df.drop('ID', axis=1, inplace=True)
all_df.head()
```

- **Obtain a quick description of the data.**

```
all_df.info()
```

- **Calculate some basic statistics of each column. Observe the value range of each feature.**

```
all_df.describe()
```

- **Check the distributions of benign and malignant as labelled in Column "diagnosis". Is the benign/malignant balance? Why is it important to check the distribution?**

```
all_df['Diagnosis'].value_counts()
```

- **Visualise benign vs malignant using a bar chart.**

```
all_df['Diagnosis']= all_df['Diagnosis'].replace('B',0, regex=True)
all_df['Diagnosis']= all_df['Diagnosis'].replace('M',1, regex=True)
sns.countplot(x="Diagnosis", data=all_df)
```

- **Use box plot to check the value range and outliers of each feature. How can we handle outliers?**

```
data_mean = all_df.iloc[:, :]
data_mean.plot(kind='box', subplots=True, layout=(8,4), sharex=False,
sharey=False, fontsize=12, figsize=(15,20));
```

- **Compare the features data ranges. Only for the first 10 features, but try yourself to visualise more features. What do you observe? Any potential problems?**

```
fig,ax=plt.subplots(1,figsize=(20,8))
sns.boxplot(data=all_df.iloc[:, 1:11],ax=ax)
```

- **Use boxplots to see if certain feature can discriminate between benign and malignant. Can you identify some useful features?**

```
fig, axes = plt.subplots(nrows=8, ncols=4, figsize=(15,20))
fig.subplots_adjust(hspace =.2, wspace=.5)
axes = axes.ravel()

for i, col in enumerate(all_df.columns[1:]):
    _= sns.boxplot(y=col, x='Diagnosis', data=all_df, ax=axes[i])
```

- **Compute the correlation matrix to observe the correlations between pairs of features. Check the definition and value range of Pearson correlation.**

```
corrMatt = all_df.corr(numeric_only=True)

# Generate a mask for the upper triangle
mask = np.zeros_like(corrMatt)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
fig, ax = plt.subplots(figsize=(20, 12))
plt.title('Breast Cancer Feature Correlation')

# Generate a custom diverging colormap
cmap = sns.diverging_palette(260, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corrMatt, vmax=1.2, square=False, cmap=cmap, mask=mask,
ax=ax, annot=True, fmt='.2g', linewidths=1);
```

- **What can you observe from the above correlation table?**

  - The area of the tissue nucleus has a strong positive correlation with values of radius and perimeter.

  - Some parameters are moderately positively correlated (r between 0.5-0.75) are concavity and area, concavity and perimeter etc.

- **Scatter plots of the first 10 "mean" features. You may try to plot the other features.**

```
sns.pairplot(all_df[list(all_df.columns[1:11]) + ['Diagnosis']], hue="Diagnosis");
```

- **What do you observe from the above scatter plots?**

  - Mean values of cell radius, perimeter, area, compactness, concavity and concave points can be used to classify the cancer. Larger values of these parameters tend to show a correlation with malignant tumours.

  - Histograms show that mean values of texture, smoothness, symmetry, or fractal dimension do not show a particular preference for one diagnosis over the other.

2.2 Data pre-processing and analysis

- **Data imputation. We don't have missing data in this dataset, but please check how to use dropna(), drop(), fillna() and SimpleImputer() to remove or impute missing data.**

```
# check how to use
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
```

- **Convert categorical texts to ordinal numbers. Some features could be text-based and need to be converted into numbers before building a machine-learning model. Scikit-learn provides some functions to implement the conversion.**

```
# check how to use:
from sklearn.preprocessing import OrdinalEncoder
ordinal_encoder = OrdinalEncoder()
```

- **Ordinal encoding assumes that two nearby values are more similar than two distant values. This is NOT a valid assumption in some cases (e.g. 1-sunny; 2-windy; 3-cloudy; 4-snowy, etc.). In this case, one-hot encoding is a better representation. I.e. convert it to one binary feature per category.**

```
# check how to use:
from sklearn.preprocessing import OneHotEncoder
cat_encoder = OneHotEncoder()
```

- **Feature normalisation. Why do we need to normalise the feature values?**

```
# Assign features to X
X = all_df.drop('Diagnosis', axis=1)

# Normalise the features to use zero mean normalisation
# only for the first 10 features, but try yourself to visualise more features

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
Xs = scaler.fit_transform(X)
fig,ax=plt.subplots(1,figsize=(20,8))
sns.boxplot(data=Xs,ax=ax)
```

- **Apply PCA for dimensionality reduction. This is a Numpy implementation of PCA. It is important that you understand the working principle of PCA.**

```
Xs_centered = Xs - Xs.mean(axis=0)
U, s, Vt = np.linalg.svd(Xs_centered)
c1 = Vt.T[:, 0]  # first mode of PC
c2 = Vt.T[:, 1]  # second mode of PC

W2 = Vt.T[:, :2] # only retain the first two principle components.
X2D = Xs_centered.dot(W2)   # project the data into PCA space

PCA_df = pd.DataFrame()
PCA_df['PCA_1'] = X2D[:,0]
PCA_df['PCA_2'] = X2D[:,1]
```

- **This is Scikit-learn implementation of PCA. Only retain the first two principal components of PCA as the new features. Could you try to retain more principle components?**

```
from sklearn.decomposition import PCA

feature_names = list(X.columns)
pca = PCA(n_components=10)
Xs_pca = pca.fit_transform(Xs)

PCA_df = pd.DataFrame()
PCA_df['PCA_1'] = Xs_pca[:,0]
PCA_df['PCA_2'] = Xs_pca[:,1]
```

- **Visualise the Malignant and Benign using the two PCA features. Could you visualise using three PCA features?**

```
plt.figure(figsize=(6,6))
plt.plot(PCA_df['PCA_1'][all_df['Diagnosis'] ==
1],PCA_df['PCA_2'][all_df['Diagnosis'] == 1],'ro', alpha = 0.7, markeredgecolor
= 'k')
plt.plot(PCA_df['PCA_1'][all_df['Diagnosis']
==0],PCA_df['PCA_2'][all_df['Diagnosis'] == 0],'bo', alpha = 0.7,
markeredgecolor = 'k')

plt.xlabel('PCA_1')
plt.ylabel('PCA_2')
plt.legend(['Malignant','Benign']);
```

3. Additional questions to think about

- Instead of using PCA to perform dimensionality reduction of features. We could also use feature selection methods. Check how to perform feature selection using Scikit-learn: https://scikit-learn.org/stable/modules/feature_selection.html

- How do we choose the right number of retained dimensions using PCA. We could use the % of retained variance (e.g. 95%) as a measure to determine the number of dimensions. Try the following code:

```
pca = PCA()
pca.fit(X_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1
# d is the retained dimension to keep 95%
# of the variance of the training set.
```

However, there is a simpler option: instead of specifying the number of principal components you want to preserve, you can set n_components to be a float between 0.0 and 1.0, indicating the ratio of variance you wish to preserve:

```
pca = PCA(n_components=0.95)
X_reduced = pca.fit_transform(X_train)
```