Hands-On Experiment 1-2: Data Preprocessing

Instructor: Dr. Peng Kang

Introduction

This hands-on experiment aims to give you a practical experience in data preprocessing techniques such as handling missing values, data cleaning, and normalization.

1 Setup

Make sure you have access to Google Colab or a Python environment with libraries like Pandas, NumPy, and Matplotlib installed.

1.1 Import Libraries

Start by importing the necessary libraries.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

1.2 Generate Data (0.2 points)

For this exercise, we'll generate a toy dataset containing random values. Execute the code below to generate it.

```
# Generate a DataFrame with random values
np.random.seed(0)
df = pd.DataFrame({
    'A': np.random.randn(100),
    'B': np.random.randint(1, 100, 100),
    'C': [np.nan if np.random.rand() < 0.2 else x for x in np.
                                    random.randn(100)]
})
df
```

2 Data Preprocessing

2.1 Missing Values (0.2 points)

After generating the DataFrame, let's first check for missing values.

```
# Check for missing values
df.isnull().sum()
```

Observation: You will notice that column 'C' has some missing values. These could be due to various reasons such as data collection issues, or they might be intentionally left blank.

2.2 Handle Missing Values (0.25 points)

Let's replace the missing values in column 'C' with the mean of the column. Note that this is only one of the ways of accounting for missing values. There are several additional ways of data imputation and handling missing values that we will investigate in future lectures.

```
# Fill missing values with mean
df['C'].fillna(df['C'].mean(), inplace=True)
df
```

Observation: The NaN values in column 'C' have now been replaced by the mean of the column.

2.3 Noisy Data (0.2 points)

Just to make things a bit more interesting, let's manually add some noise to the data. Please note that the real data may already contain noise. Here, we are synthetically adding some noise to the data to simulate real data.

Generate some noisy data and add it to column ‘A’.

```
# Add noise
noise = np.random.normal(0, 0.1, df.shape[0])
df['A'] = df['A'] + noise
```

Observation: The column ‘A‘ is now altered with some random noise.

2.4 Binning (0.25 points)

Binning can be a useful technique for noise reduction in data. When you bin a continuous variable, you group a set of values into a category, thereby smoothing the dataset. It’s also a form of discretization.

Let’s perform binning on column ‘A’ to categorize the data into three bins: Low, Medium, and High.

```
# Binning
labels = ['Low', 'Medium', 'High']
df['A_bins'] = pd.cut(df['A'], bins=3, labels=labels)
df
```

Observation: The ‘A_bins‘ column is created with labels specifying the bin each value in column ‘A’ belongs to.

2.5 Mean Smoothing (0.25 points)

One way to smooth the data after binning is by taking the mean of all the values in each bin and replacing the original values with this mean. This approach can help reduce the noise in the dataset:

```
df['A_binned_mean'] = df.groupby('A_bins')['A'].transform('mean'
                                )
df
```

2.6 Gaussian Smoothing (0.25 points)

Another advanced technique for noise removal involves using Gaussian smoothing or filtering. Here, each data point is replaced by a weighted average of its neighbors, with weights given by a Gaussian function:

```
from scipy.ndimage.filters import gaussian_filter

sigma = 2  # Standard deviation for Gaussian kernel
df['A_gaussian_smoothed'] = gaussian_filter(df['A'], sigma)
df
```

2.7 Data Reduction: Sampling (0.25 points)

Take a random sample of 20% of the DataFrame.

```
# Sampling
df_sample = df.sample(frac=0.2)
df_sample
```

Observation: A new DataFrame 'df_sample' is created with a random 20% of the original data.

2.8 Normalization

Data normalization is a required data preparation step for many Data Mining/Machine Learning algorithms. These algorithms are sensitive to the relative values of the feature attributes. Data normalization is the process of bringing all the attribute values within some desired range. Unless the data is normalized, these algorithms don't behave correctly.

To provide some context, we will also discuss how different supervised learning algorithms are negatively impacted by lack of normalization.

2.8.1 Why Normalize?

Some Data Mining/Machine Learning algorithms are sensitive to the relative magnitudes of the feature attributes. Normalization alleviates this problem.

- The K Nearest Neighbor Algorithm (KNN) is based on the distance between records. Unless data is normalized, distance will be incorrectly calculated because different attributes will not contribute to the distance in a uniform way. Attributes with a larger value range will have an unduly larger influence on the distance because they will make a greater contribution to the distance.

- In Artificial Neural Network (ANN), linear algebra operations are performed between the input and weight vectors. With ANN, normalization is not strictly necessary, as the weights can accommodate various input feature attributes. However, training can be more efficient, and convergence can be reached faster when the data is normalized.
- In Support Vector Machine (SVM), the algorithm finds the hyperplane separating the data points belonging to the different classed by optimization techniques, and distance calculation enter the picture. Hence, normalization becomes a necessity. However, if kernel functions are used instead of calculating distance directly, the function may be able to handle differences in scales between attributes directly, and normalization may be skipped.

As a counterexample, let's consider Decision Tree and Random Forest. In the Decision Tree, the feature space is subdivided into different regions, keeping data homogeneity in each region as the criteria. The algorithm operates on each attribute independently, and relative values of different attributes are irrelevant. So, normalization is not necessary.

2.8.2 Normalization Techniques

There are various normalization techniques. The appropriate technique to be used depends on the data mining/machine learning algorithm to be used on the normalized data. The most popular techniques are minmax and z-score.

- The minmax technique is based on the min and max values of the attribute as follows. Normalize values will be between 0 and 1 typically.
- The max technique only uses the max value for normalization. The normalized values will between -1 and 1.
- The z-score technique is based on mean and standard deviation. Most of the normalized data will be between -1 and 1. Since the normalized data will follow a standard distribution, this technique is known as standardization. Standard distribution N(0,1) is a normal distribution with a mean of 0 and standard deviation 1.
- The center technique is based on mean only as below. The normalized data is not constrained by any range limit.

- The decimal technique, the value is scaled by a quantity which is a power of 10 and greater than the max value. Normalized values will be within the limits -1 and 1

- The unitSum technique is based on the sum of the values as below. The normalized data is not constrained by any range limit.

All the techniques described are prone to outliers. The z-score technique provides the option of purging outlier data while normalizing. Since outliers have high z-score, we could remove any record with a z-score above some threshold.

2.8.3 MinMax Scaling (0.5 points)

Manually Perform Min-Max normalization on column 'A'.

```
# Min-Max normalization - manually
df['A_norm_man'] = (df['A'] - df['A'].min()) / (df['A'].max() -
                                        df['A'].min())
df
```

Observation: A new column 'A_norm_man' is created where values are manually normalized to lie between 0 and 1.

MinMaxScaler from 'scikit-learn' transforms a dataset of Vector rows, rescaling each feature to a specific range (often [0, 1]). It takes parameters:

- min: 0.0 by default. Lower bound after transformation, shared by all features

- max: 1.0 by default. Upper bound after transformation, shared by all features

The MinMaxScaler computes summary statistics on the data set (also known as fitting) and produces a Scaler. The scaler can then transform each feature individually such that it is in the given range. These two steps of the process are done by invoking one command, called *fit_transform*.

MinMaxScaler Perform Min-Max normalization on column 'A' using scikit-learn's MinMaxScaler.

```
# Min-Max normalization using sklearn
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the data
df['A_norm_sk'] = scaler.fit_transform(df['A'].values.reshape(-1
                                , 1))
df
```

Observation: A new column 'A_norm_sk' is created where values are normalized to lie between 0 and 1, using the sklearn library.

2.9 Z-Score Normalization (0.5 points)

Z-score normalization, or standardization, is a technique used to transform a dataset such that it has a mean of zero and a standard deviation of one.

Given a dataset $X = \{x_1, x_2, \ldots, x_n\}$, the z-score for an individual data point x_i is calculated as:

$$z_i = \frac{x_i - \mu}{\sigma} \tag{1}$$

Where:

- x_i is an individual data point.
- μ is the mean of the dataset X, defined as $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$.
- σ is the standard deviation of the dataset X, defined as $\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$.

After performing the z-score normalization on X, the new dataset will have a mean of $\mu' = 0$ and a standard deviation $\sigma' = 1$.

Similar to MinMax scaling, we can do this normalization *manually*, or we can use sklearn's built-in functions.

Manually Perform Z-Score normalization on column 'B'.

```
# Z-Score normalization - Manually
df['B_zscore_man'] = (df['B'] - df['B'].mean()) / df['B'].std()
df
```

Observation: A new column 'B_zscore_man' is created with manually calculated Z-Score normalized values.

Using scikit-learn The StandardScaler from scikit-learn is a quick and efficient way to perform Z-score normalization.

```
# Z-Score normalization using sklearn
from sklearn.preprocessing import StandardScaler

# Create a scaler object and fit-transform the data
scaler = StandardScaler()
df['B_zscore_sk'] = scaler.fit_transform(df['B'].values.reshape(
                                -1,1))
df
```

Observation: A new column 'B_zscore_sk' is created with Z-Score normalized values, calculated using sklearn.

2.10 Data Aggregation (0.2 points)

Let's create an aggregated column that takes the average of columns 'A' and 'B'.

```
# Data aggregation
df['Aggregated'] = (df['A'] + df['B']) / 2
df
```

Observation: The new column 'Aggregated' is the average of columns 'A' and 'B'.

2.11 Visualizing Data (0.25 points)

Use Matplotlib to visualize the distribution of a column.

```
# Plotting
plt.hist(df['A'], bins=20)
plt.title('Distribution of Column A')
plt.show()
```

Observation: The histogram helps in visualizing the distribution of values in column 'A'.

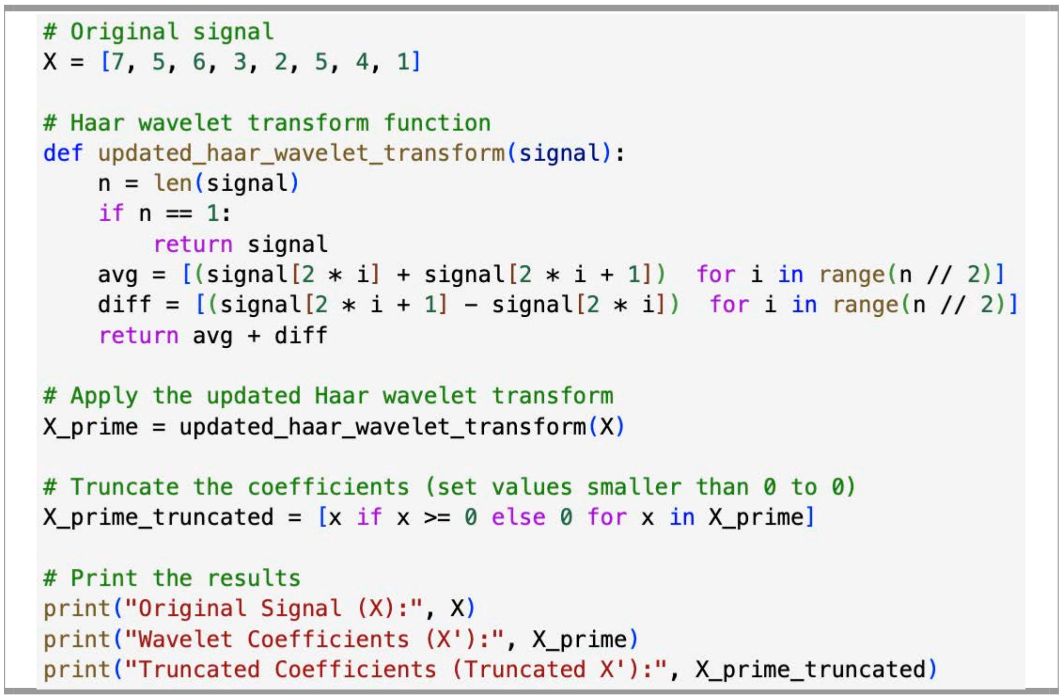
2.12 Correlation Matrix (0.25 points)

Generate a correlation matrix to check the relationships between the columns.

```
# Correlation matrix
correlation_matrix = df[['A', 'B', 'C']].corr()
correlation_matrix
```

Observation: The correlation matrix shows how strong each pair of attributes are related

2.13 Wavelet Transform - Data Reduction (0.45 points)



```
# Original signal
X = [7, 5, 6, 3, 2, 5, 4, 1]

# Haar wavelet transform function
def updated_haar_wavelet_transform(signal):
    n = len(signal)
    if n == 1:
        return signal
    avg = [(signal[2 * i] + signal[2 * i + 1])  for i in range(n // 2)]
    diff = [(signal[2 * i + 1] - signal[2 * i])  for i in range(n // 2)]
    return avg + diff

# Apply the updated Haar wavelet transform
X_prime = updated_haar_wavelet_transform(X)

# Truncate the coefficients (set values smaller than 0 to 0)
X_prime_truncated = [x if x >= 0 else 0 for x in X_prime]

# Print the results
print("Original Signal (X):", X)
print("Wavelet Coefficients (X'):", X_prime)
print("Truncated Coefficients (Truncated X'):", X_prime_truncated)
```

3 Summary

Congratulations, you've successfully preprocessed a dataset, filling in missing values, handling noisy data, performing data transformations, normalization, and much more. Understanding these techniques is fundamental in preparing your data for Machine Learning algorithms.