CO544 – Machine Learning and Data Mining

Lab 1: Python Data Science Toolbox

E/20/212 Kumarasinghe R.M.S.H.

Exercise 1: NumPy Advanced Operations

This exercise uses NumPy to illustrate a number of basic operations. First, np.random.randint was used to create a one-dimensional array of 20 randomly selected integers between 0 and 100. This step demonstrates how effectively NumPy can generate test data. Then, values greater than or equal to 50 were filtered using boolean indexing, demonstrating how NumPy enables efficient and clear data filtering without the need for explicit loops.

The array was reshaped into a 4x5 matrix and added to a one-dimensional array of shape (5,) to demonstrate broadcasting. By handling element-wise operations between arrays of various shapes, NumPy eliminates the need for manual looping constructs, as demonstrated by this operation. Lastly, np.dot was used to compute the dot product of two arrays of length 10, demonstrating NumPy's support for high-performance linear algebra operations. In data science workflows, these methods are crucial for effectively managing and converting data.

Exercise 2: Matplotlib Subplots

The goal of this exercise is to use Matplotlib to generate and visualize trigonometric functions. The input domain (x) for the sine and cosine functions was np.linspace, which produced a set of 200 uniformly spaced values between 0 and 2π. Two distinct arrays for plotting were then created by computing these functions using np.sin and np.cos, respectively.

plt.subplots was used to create a figure with two subplots arranged horizontally in order to visualize the data. For uniform scaling, the sine and cosine waves were plotted on the same x-axis in the first and second subplots, respectively. To improve readability, each subplot was given a title and axis labels, and fig.suptitle was used to add a common figure title. Ultimately, plt.show() was used to display the plot after it was saved as an image file (trig_functions.png). This exercise shows how Matplotlib can produce neat, well-structured multi-panel visualizations, which is especially helpful when comparing related datasets side by side.

Exercise 3: Pandas Cleaning & Preprocessing

In this exercise, the Pandas library was used to apply fundamental data cleaning and preprocessing steps to the Titanic dataset. After using pd.read_csv to load the dataset straight from an online source, df.info() was used to perform a preliminary inspection, which showed that the Age and Embarked columns had missing values. The most frequent (mode) value was used to fill the Embarked column, and the median value of the Age column was used to handle these missing entries. These techniques guarantee that missing data is filled in without eliminating significant amounts of the dataset or adding bias from extreme values.

To maintain data integrity and prevent skewed analysis results, drop_duplicates() was used to eliminate duplicates from the dataset. The Fare column was converted to an integer type and rounded to the closest integer in order to further process the data. The result was saved in a new column named Fare_int. The Interquartile Range (IQR) method was used to identify outliers in this column, where values outside the range [Q1 − 1.5 × IQR , Q3 + 1.5 × IQR] were found. This step demonstrates how anomalies that could affect model performance or downstream analysis can be found using statistical techniques.

Exercise 4: Pandas Essentials

A number of essential Pandas operations that are essential for efficient data analysis are introduced in this exercise. The .shape and. index attributes, which offer structural metadata about the Series, were first used to create and inspect a simple Series object. The versatility of Pandas indexing was further demonstrated by the creation of another Series with unique labels as index values.

After that, two DataFrames were constructed: a smaller DataFrame (df1) with named columns for people and their scores, and a larger DataFrame (df2) with three columns and 100 rows of random numbers. The content, structure, and statistical characteristics of these DataFrames were examined using summary operations like .head(), .tail(), .info(), and .describe().

Strong indexing strategies for accessing data by label or by integer position were demonstrated by the use of .loc and .iloc. Sorting by a specific column and dropping a column ('B') illustrated common data manipulation patterns. Using dropna() and fillna(), two popular methods for handling incomplete data, it was shown how to handle missing data by either removing rows with missing values or filling them with zeroes.

Lastly, pd.read_excel() and .to_excel() were used to execute input/output operations on Excel files. This demonstrated how Pandas can easily work with external file formats, which makes it appropriate for real-world data pipelines involving data exchange based on Excel.

Exercise 5: Loading Open Dataset from UCI Repository

To improve readability and semantic comprehension, the Wine dataset from the UCI Machine Learning Repository was loaded using pd.read_csv and given the proper column names in this exercise. This dataset contains a variety of wine chemical characteristics as well as the class labels that correspond to distinct cultivars.

To aggregate the numerical features by wine class, a group-by operation was applied to the Class column using.groupby('Class').mean(). A statistical summary highlighting the variations in chemical composition between the classes is provided by the output that is produced, which gives the mean value of each attribute for each wine class. In exploratory data analysis, this kind of grouping is useful because it makes it possible to find patterns or feature significance that can subsequently help with classification or clustering tasks. Prior to using machine learning models, the grouped summary is also a fundamental step in comprehending class-wise feature distributions.

Exercise 6: scikit-learn Iris Dataset (Extended)

Using the Iris dataset, this exercise shows a comprehensive workflow for creating and assessing a classification model. A DataFrame was made to arrange the features and target labels in a tabular format for convenience of analysis and preprocessing after the dataset was loaded from sklearn.datasets.

Then, using train_test_split, the dataset was divided into training and testing sets, with 30% going to testing and 70% going to training. To guarantee reproducibility of results, a fixed random seed (random_state=42) was specified. A common method for assessing a model's capacity for generalization on unknown data is the train-test split approach.

In order to guarantee convergence, a logistic regression model was trained using LogisticRegression, with a maximum iteration count of 200. Classification_report from sklearn.metrics was used to assess the model's performance after predictions were made on the test set. The report offers a thorough analysis of the classifier's advantages and disadvantages by including precision, recall, f1-score, and support for every class. All things considered, this exercise highlights the value of model validation using systematic performance metrics and shows how to apply a simple machine learning model.