**Machine Learning**

**LAB 04**



**DEPARTMENT OF COMPUTER SCIENCE**

**FACULTY OF ENGINEERING & COMPUTER SCIENCES**

**NATIONAL UNIVERSITY OF MODERN LANGUAGES**

**H-9, ISLAMABAD**

**PAKISTAN**

**LAB 4: DATA PREPROCESSING - II**

**Objectives:**

* To understand and implement the data preprocessing techniques

**Time Required** : 3 hrs

**Programming Language** : Python

**Software Required** : Anaconda

**Introduction**

In our daily life, we deal with lots of data, but this data is in raw form. To provide the data as the input of machine learning algorithms, we need to convert it into a meaningful data. That is where data preprocessing comes into picture. In other simple words, we can say that before providing the data to the machine learning algorithms, we need to preprocess the data. using a properly processed dataset while training will not only make life easier for you but also **increase the efficiency and accuracy** of your model.

Steps in Data Preprocessing:

* Importing the libraries
* Importing the dataset
* Taking care of missing data
* Encoding categorical data
* Normalizing the data
* Splitting the data into test and train

**Data preprocessing steps**

Follow these steps to preprocess the data in Python:

**Step 1: Import useful libraries**

we’ll import three basic libraries which are very common in machine learning and will be used every time you train a model

* NumPy: it is a library that allows us to work with arrays and as most machine learning models work on arrays NumPy makes it easier
* matplotlib: this library helps in plotting graphs and charts, which are very useful while showing the result of your model
* Pandas: pandas allow us to import our dataset and also creates a matrix of features containing the dependent and independent variable.
* Sklearn: It contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering, and dimensionality reduction.This package provides many common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for machine learning algorithms.

**Task 1:** Import all the necessary libraries

**Step 2: Import dataset**

The second step includes importing the dataset. In this lab, we are using a very simple dataset that contains information about customers who have purchased a particular product from a company. It contains various information about the customers like their age, salary, country, etc. It also shows whether a particular customer has purchased the product or not. It also contains a null value in the fifth row.

**Task 2:** Import the dataset that is provided to you and display. As the given data is in CSV format, use the *read\_csv*function from the pandaslibrary.

Note: You must remember the data imported using the *read\_csv* function is in a Data Frame format, we’ll later convert it into *NumPy arrays*to perform other operations and training.

**Step 3: Handling the missing values**

In our dataset, we have two missing values one in the *Salary*column in the 5th Row and another in the *Age*column of the 7th row. Now there are multiple ways to handle missing values, one of them is to ignore them and delete the entire entry/row, this is commonly done in datasets containing a very large number of entries, where the missing values only constitute 0.1% of the total data. Thus, they affect the model negligibly and can be removed. But in our case, the dataset is very small, and we cannot just ignore those rows. So, we use another method, in which we take the mean of the entire column containing the missing values (in our case the *age*or *salary* column) and replace the missing values with that mean.

**Task 3:** Import SimpleImputer classfrom the ScikitLearn library by using the following code.

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing\_values=np.nan, strategy='mean')

# 'np.nan' signifies that we are targeting missing values

# and the strategy we are choosing is replacing it with 'mean'

imputer.fit(data.iloc[:, 1:3])

data.iloc[:, 1:3] = imputer.transform(data.iloc[:, 1:3])

**Step 4: Encoding categorical data**

1. **OneHot Encoding:**

In our case, we have two categorical columns, the *country* column, and the *purchased* column. In the *country* column, we have three different categories: France, Germany, Spain. We can simply label France as 0, Germany as 1, and Spain as 2 but doing this might lead our machine learning model to interpret that there is some correlation between these numbers and the outcome. So, to avoid this, we apply OneHot Encoding.

OneHot Encoding consists of turning the country column into three separate columns, each column consists of 0s and 1s. Therefore, each country will have a unique vector/code and no correlation between the vectors and outcome can be formed.

**Task 4:** Implement OneHotEncoder using ColumnTransformerclass from the same ScikitLearnlibrary by using the following code:

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder

ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder='passthrough')

# [0] signifies the index of the column we are applying the encoding on

data = pd.DataFrame(ct.fit\_transform(data))

The ColumnTransformer class allows us to select the column to apply encoding on and leave the other columns untouched.

Note: The new columns created will be added in the front of the data frame and the original column will be deleted.

1. **Label Encoder:**

In the last column, i.e. the purchased column, the data is in binary form meaning that there are only two outcomes either Yes or No. Therefore, here we need to perform Label Encoding.

**Task 5:** Use LabelEncoder class from the same ScikitLearn library to transform purchased column using the following code:

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

data.iloc[:,-1] = le.fit\_transform(data.iloc[:,-1])

# 'data.iloc[:,-1]' is used to select the column that we need to be encoded

**Step 5: Normalizing the dataset**

Feature scaling is bringing all of the features on the dataset to the same scale, this is necessary while training a machine learning model because in some cases the dominant features become so dominant that the other ordinary features are not even considered by the model.

When we normalize the dataset, it brings the value of all the features between 0 and 1 so that all the columns are in the same range, and thus there is no dominant feature.

**Task 6:** Normalize the dataset using MinMaxScaler class from the same ScikitLearn library using the following code:

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

data = pd.DataFrame(scaler.fit\_transform(data))

**Step 6: Splitting the dataset**

In machine learning, a **larger part**of the dataset is used to train the model, and a small part is used to test the trained model for finding out the accuracy and the efficiency of the model. Before we begin splitting the dataset, we need to separate the dependent and independent variables.

The independent variable is the columns that we are going to use to predict the dependent variable, or in other words, the independent variable affects the dependent variable. In our dataset, the *country*, *age*, and *salar*y column are the independent variable and will be used to predict the *purchased* column which is the dependent variable.

Another important part we need to remember is that while training the model accepts data as arrays, so it is necessary that we convert the data to arrays. We do that while separating the dependent and independent variables by adding *.values*while storing data.

**Task 7:** Split the dependent and independent variables and store the dependent variable in ‘y’ and the independent variables in ‘X’.

After splitting our data into dependent and independent variables, we need to split it into testing and training data. Deciding the ratio between testing data and training data is up to us and depends on what we are trying to achieve with our model. In our case, we are going to go with an 80-20% split between the train-test data. So, 80% training and 20% testing data.

**Task 8:** Split the dataset into testing data and training data using the **train\_test\_split**class from the same **ScikitLearn**library. The code below is for help.

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2)

Here the *test\_size = 0.2*signifies that we have selected 20% of data as testing data, you can change that according to your choice. After this, the *X\_train* and *X\_test* variables will have their respective data and our data is ready for training.

**Final Lab Task:**

**Your task is to perform data preprocessing on the given dataset.**

1. Import the dataset ‘titanic.csv’ (you can directly import via seaborn library as well).
2. Delete columns 'alive', 'alone', 'embark\_town', 'who', 'adult\_male', 'deck', 'embarked', and 'class' because they are either irrelevant or have less influence on the prediction of survival or contain repeated values with just different attribute name.

Note: You should not delete the columns one by one. Create a list of all those features that you want to delete and delete this list.

1. Convert the categorical column ‘sex’ using Label Encoder
2. Check for missing values and replace them by ‘mean’ for numerical attributes whereas ‘mode’ for categorical attributes.
3. Perform feature scaling by using MinMax Scaler or Standard Scaler
4. Print the dataset after implementing all of the above data preprocessing
5. Split dataset into dependent and independent variable. ‘Survive’ is your dependent variable here.
6. Split dataset into testing and training sets.