Data Preprocessing

Introduction

Data is a crucial component in the field of Machine Learning. Data is typically divided into two types:

1. Labeled data
2. Unlabeled data

Labeled data includes a label or target variable that the model is trying to predict, whereas unlabeled data does not include a label or target variable. The data used in machine learning is typically numerical or categorical. Numerical data includes values that can be ordered and measured, such as age or income. Categorical data includes values that represent categories, such as gender or type of fruit.

How do we split data in Machine Learning?

- **Training Data:** The part of data we use to train our model. This is the data that your model actually sees(both input and output) and learns from.
- **Validation Data:** The part of data that is used to do a frequent evaluation of the model, fit on the training dataset along with improving involved hyperparameters (initially set parameters before the model begins learning). This data plays its part when the model is actually training.
- **Testing Data:** Once our model is completely trained, testing data provides an unbiased evaluation. When we feed in the inputs of Testing data, our model will predict some values(without seeing actual output). After prediction, we evaluate our model by comparing it with the actual output present in the testing data. This is how we evaluate and see how much our model has learned from the experiences feed in as training data, set at the time of training.

```
# Example1: Basic Model from scikit learn
from sklearn.linear_model import LogisticRegression
X = [[1, 2], [2, 3], [3, 4], [4, 5], [5, 6]]
y = [0, 0, 1, 1, 1]

# Train a model
model = LogisticRegression()
model.fit(X, y)

# Make a prediction
prediction = model.predict([[6, 7]])[0]
print(prediction)
```

Step 1: The first step is to create the model object. We did this by importing the `LogisticRegression` class and then instantiating it

Step 2: Next, we need to train the model on our data. We did this by calling the `fit()` method on the model object, passing in the features (`X`) and labels (`y`) of our training data:

Step 3: Once the model is trained, we can make predictions on new data by calling the `predict()` method on the model object. The `predict()` method takes a NumPy array of features as input and returns a NumPy array of predicted labels as output

The `[0]` at the end of the `predict()` method call is necessary because the `predict()` method returns a NumPy array, even if the input is a single data point. The `[0]` indexes into the NumPy array to extract the first element, which is the predicted label for the input data point.

The main steps involved in data processing typically include

1.Data collection: This is the process of gathering data from various sources, such as sensors, databases, or other systems. The data may be structured, semi structured or unstructured, and may come in various formats such as text, images, or audio.
2.Data preprocessing: This step involves cleaning, filtering, and transforming the data to make it suitable for further analysis. This may include removing

missing values, scaling or normalizing the data, or converting it to a different format.
3.Data analysis: In this step, the data is analyzed using various techniques such as statistical analysis, machine learning algorithms, or data visualization. The goal of this step is to derive insights or knowledge from the data.
4.Data interpretation: This step involves interpreting the results of the data analysis and drawing conclusions based on the insights gained. It may also involve presenting the findings in a clear and concise manner, such as through reports, dashboards, or other visualizations.
5.Data storage and management: Once the data has been processed and analyzed, it must be stored and managed in a way that is secure and easily accessible. This may involve storing the data in a database, cloud storage, or other systems, and implementing backup and recovery strategies to protect against data loss.
6.Data visualization and reporting: Finally, the results of the data analysis are presented to stakeholders in a format that is easily understandable and actionable. This may involve creating visualizations, reports, or dashboards that highlight key findings and trends in the data.

Python | Create Test DataSets using Sklearn

```
#Example2: Creating a sample test dataset using sklearn.datasets.make_blobs
from sklearn.datasets import make_blobs
from matplotlib import pyplot as plt
from matplotlib import style

style.use("fivethirtyeight")

X, y = make_blobs(n_samples = 100, centers = 3, cluster_std = 1, n_features = 2)

plt.scatter(X[:, 0], X[:, 1], s = 40, color = 'g')
plt.xlabel("X")
plt.ylabel("Y")

plt.show()
plt.clf()
```

In the code above, we are setting the size of the data points to 40 and the color of the data points to green. We are also setting the x-label to "X" and the y-label to "Y".When the code runs, it generated 100 sample data points.

```
from sklearn.datasets import make_moons
```

```
from sklearn.datasets import make_circles
```

Generate Test Datasets for Machine learning

While there are many datasets that you can find on websites such as Kaggle, sometimes it is useful to extract data on your own and generate your own dataset. Generating your own dataset gives you more control over the data and allows you to train your machine-learning model.

#Example 3 Test data
Using make_multilabel_classification() - A random multi-label classification method
# Import necessary libraries
from sklearn.datasets import make_multilabel_classification
import pandas as pd
import matplotlib.pyplot as plt

Generate 2d classification dataset
X, y = make_multilabel_classification(n_samples=500, n_features=2, n_classes=2, n_labels=2, allow_unlabeled=True, random_state=23)

# create pandas dataframe from generated dataset
df = pd.concat([pd.DataFrame(X, columns=['X1', 'X2']), pd.DataFrame(y, columns=['Label1', 'Label2'])], axis=1)
display(df.head())

# Plot the generated datasets
plt.scatter(df['X1'], df['X2'], c=df['Label1'])
plt.show()

The code generates a synthetic multilabel classification dataset, structures it into a Pandas DataFrame, and then visualizes the dataset by plotting a scatter plot where the points are colored based on the 'Label1'.

```
#Example 4 Multilabel feature using make_friedman2()
# Import necessary libraries
from sklearn.datasets import make_friedman2
import matplotlib.pyplot as plt

# Generate 1d Regression dataset
X, y = make_friedman2(n_samples = 100, random_state=23)

# Plot the generated datasets
plt.figure(figsize=(12,10))
for i in range(4):
        plt.subplot(2,2, i+1)
        plt.scatter(X[:,i], y)
        plt.xlabel('X'+str(i+1))
        plt.ylabel('Y')
plt.show()
```

Exploratory Data Analysis (EDA)

- Import the necessary libraries
- Load the dataset
- Check the data information using df.info()

```
import pandas as pd
import numpy as np

# Load the dataset
df = pd.read_csv('train.csv')
df.head()
```

1. Data inspection and exploration:

This step involves understanding the data by inspecting its structure and identifying missing values, outliers, and inconsistencies.

- Check the duplicate rows: df.duplicated()
- Check the data information using df.info()
- descriptive structure of the data using df.describe()
- Check the categorical and numerical columns

```
# Categorical columns
cat_col = [col for col in df.columns if df[col].dtype == 'object']
print('Categorical columns :',cat_col)
# Numerical columns
num_col = [col for col in df.columns if df[col].dtype != 'object']
print('Numerical columns :',num_col)
```

- Check the total number of unique values in the Categorical columns. This includes deleting duplicate/ redundant or irrelevant values from your dataset.

```
df1 = df.drop(columns=['XXX','YYY'])
```

```
df1.shape


df2 = df1.drop(columns='XXX')
df2.dropna(subset=['Embarked'], axis=0, inplace=True)
df2.shape
```

Mean and Median imputations

```
# Mean imputation
df3 = df2.fillna(df2.Age.mean())
# Let's check the null values again
df3.isnull().sum()
```

4. Handling outliers:

Outliers are extreme values that deviate significantly from the majority of the data. They can negatively impact the analysis and model performance. Techniques such as clustering, interpolation, or transformation can be used to handle outliers.

To check the outliers, We generally use a box plot. A box plot, also referred to as a box-and-whisker plot, is a graphical representation of a dataset's distribution. It shows a variable's median, quartiles, and potential outliers. The line inside the box denotes the median, while the box itself denotes the interquartile range (IQR). The whiskers extend to the most extreme non-outlier values within 1.5 times the IQR. Individual points beyond the whiskers are considered potential outliers. A box plot offers an easy-to-understand overview of the range of the data and makes it possible to identify outliers or skewness in the distribution.
Let's plot the box plot for Age[replace with actual column] column data.

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

# Reading the CSV file into a DataFrame
df3 = pd.read_csv('sample.csv')

# Converting 'Value' column to numeric, handling non-convertible values as NaN
df3['Value'] = pd.to_numeric(df3['Value'], errors='coerce')

# Creating a box plot for the 'Value' column after dropping NaN values
plt.boxplot(df3['Value'].dropna(), vert=False)
plt.ylabel('Variable')
plt.xlabel('Value')
plt.title('Box Plot')
plt.show()
```

One Hot Encoding in Machine Learning

For categorical data to fit() into a ML model it needs to be converted into numerical data. For example, suppose a dataset has a *Gender* column with categorical elements like *Male and Female*. These labels have no specific order of preference and also since the data is string labels, machine learning models misinterpreted that there is some sort of hierarchy in them.

One approach to solve this problem can be label encoding where we will assign a numerical value to these labels for example *Male* and *Female* mapped to *0* and *1*. But this can add bias in our model as it will start giving higher preference to the *Female* parameter as 1>0 but ideally, both labels are equally important in the dataset. To deal with this issue we will use the One Hot Encoding technique.

One Hot Encoding
One hot encoding is a technique that we use to represent categorical variables as numerical values in a machine learning model.
The advantages of using one hot encoding include:

1. It allows the use of categorical variables in models that require numerical input.
2. It can improve model performance by providing more information to the model about the categorical variable.
3. It can help to avoid the problem of ordinality, which can occur when a categorical variable has a natural ordering (e.g. “small”, “medium”, “large”).

The disadvantages of using one hot encoding include:

1. It can lead to increased dimensionality, as a separate column is created for each category in the variable. This can make the model more complex and slow to train.

2. It can lead to sparse data, as most observations will have a value of 0 in most of the one-hot encoded columns.

3. It can lead to overfitting, especially if there are many categories in the variable and the sample size is relatively small.

One Hot Encoding Examples
In **One Hot Encoding**, the categorical parameters will prepare separate columns for both Male and Female labels. So, wherever there is a Male, the value will be 1 in the Male column and 0 in the Female column, and vice-versa. Let’s understand with an example: Consider the data where fruits, their corresponding categorical values, and prices are given.

Fruit	Categorical value of fruit	Price
apple	1	5
mango	2	10
apple	1	15
orange	3	20

The output after applying one-hot encoding on the data is given as follows,

apple	mango	orange	price
1	0	0	5
0	1	0	10
1	0	0	15
0	0	1	20

```
# importing libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder

# Retrieving data
data = pd.read_csv('Employee_data.csv')

# Converting type of columns to category
data['Gender'] = data['Gender'].astype('category')
data['Remarks'] = data['Remarks'].astype('category')


# Assigning numerical values and storing it in another columns
data['Gen_new'] = data['Gender'].cat.codes
data['Rem_new'] = data['Remarks'].cat.codes


# Create an instance of One-hot-encoder
enc = OneHotEncoder()
```

```
# Passing encoded columns

enc_data = pd.DataFrame(enc.fit_transform(
        data[['Gen_new', 'Rem_new']]).toarray())

# Merge with main
New_df = data.join(enc_data)

print(New_df)
```