Machine learning

TutorialsPoint

Real-World Application - ML

- Emotion analysis
- Sentiment analysis
- Error detection and prevention
- Weather forecasting and prediction
- Stock market analysis and forecasting
- Speech synthesis
- Speech recognition
- Customer segmentation
- Object recognition
- Fraud detection
- Fraud prevention
- Recommendation of products to customer in online shopping

Different Methods For ML

The following are various ML methods based on some broad categories −

- Based on human supervision
- Unsupervised Learning
- Semi-supervised Learning
- Reinforcement Learning

Understanding Data with Statistics

1- Looking at Raw Data

It is important to look at raw data because the insight we will get after looking at raw data will boost our chances to better pre-processing as well as handling of data for ML projects.

```
from pandas import read_csv
path = r"C:\pima-indians-diabetes.csv"
headernames = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
data = read_csv(path, names=headernames)
print(data.head(50))

>>>
preg plas  pres skin test   mass pedi   age     class
0     6    148  72   35     0    33.6   0.627   50     1
1     1     85  66   29     0    26.6   0.351   31     0
2     8    183  64    0     0    23.3   0.672   32     1
```

2- Checking Dimensions of Data

It is always a good practice to know how much data, in terms of rows and columns, we are having for our ML project. The reasons behind are −

- Suppose if we have too many rows and columns then it would take a long time to run the algorithm and train the model.
- Suppose if we have too few rows and columns then it we would not have enough data to train the model.

```
print(data.shape)

>>>
(150, 4)
```

3- Getting Each Attribute’s Data Type

It is another good practice to know the data type of each attribute. The reason behind is that, as per the requirement, sometimes we may need to convert one data type to another.

```
print(data.dtypes)

>>>
sepal_length   float64
sepal_width    float64
petal_length   float64
petal_width    float64
dtype: object
```

4- Statistical Summary of Data

We have discussed the shape() function of data but many times we need to review the summaries out of that shape of data. It can be done with the help of *describe()* function of Pandas DataFrame that further provide the following 8 statistical properties of each & every data attribute −

- Count
- Mean
- Standard Deviation
- Minimum Value
- Maximum value
- 25%
- Median i.e. 50%
- 75%

```
print(data.shape)
print(data.describe())

>>>
(768, 9)
       preg     plas     pres    skin     test     mass     pedi     age      class
count  768.00   768.00   768.00  768.00   768.00   768.00   768.00   768.00   768.00
```

```
mean      3.85    120.89    69.11   20.54    79.80    31.99     0.47    33.24     0.35
std       3.37     31.97    19.36   15.95   115.24     7.88     0.33    11.76     0.48
min       0.00      0.00     0.00    0.00     0.00     0.00     0.08    21.00     0.00
25%       1.00     99.00    62.00    0.00     0.00    27.30     0.24    24.00     0.00
50%       3.00    117.00    72.00   23.00    30.50    32.00     0.37    29.00     0.00
75%       6.00    140.25    80.00   32.00   127.25    36.60     0.63    41.00     1.00
max      17.00    199.00   122.00   99.00   846.00    67.10     2.42    81.00     1.00
```

5- Reviewing Class Distribution

Class distribution statistics is useful in classification problems where we need to know the balance of class values.

```
count_class = data.groupby('class').size()
print(count_class)

>>>
Class
0 500
1 268
dtype: int64
```

6- Reviewing Correlation between Attributes

The relationship between two variables is called **correlation**. In statistics, the most common method for calculating correlation is **Pearson's Correlation Coefficient**. It can have three values as follows −

- Coefficient value = **1** − It represents full positive correlation between variables.
- Coefficient value = **-1** − It represents full negative correlation between variables.
- Coefficient value = **0** − It represents no correlation at all between variables.

```
correlations = data.corr(method='pearson')
print(correlations)

>>>
       preg   plas   pres   skin   test    mass    pedi    age    class
preg   1.00   0.13   0.14  -0.08   -0.07   0.02   -0.03   0.54    0.22
plas   0.13   1.00   0.15   0.06    0.33   0.22    0.14   0.26    0.47
pres   0.14   0.15   1.00   0.21    0.09   0.28    0.04   0.24    0.07
skin  -0.08   0.06   0.21   1.00    0.44   0.39    0.18  -0.11    0.07
test  -0.07   0.33   0.09   0.44    1.00   0.20    0.19  -0.04    0.13
mass   0.02   0.22   0.28   0.39    0.20   1.00    0.14   0.04    0.29
```

```
pedi  -0.03   0.14   0.04   0.18    0.19    0.14    1.00   0.03    0.17
age    0.54   0.26   0.24  -0.11   -0.04    0.04    0.03   1.00    0.24
```

7- Reviewing Skew of Attribute Distribution

Skewness may be defined as the distribution that is assumed to be Gaussian but appears distorted or shifted in one direction or another. Reviewing the skewness of attributes is one of the important tasks due to following reasons −

- Presence of skewness in data requires the correction at data preparation stage so that we can get more accuracy from our model.
- Most of the ML algorithms assumes that data has a Gaussian distribution i.e. either normal of bell curved data.

```
print(data.skew())

>>>
preg   0.90
plas   0.17
pres  -1.84
skin   0.11
test   2.27
mass  -0.43
pedi   1.92
age    1.13
class  0.64
dtype: float64
```

From the above output, positive or negative skew can be observed. If the value is closer to zero, then it shows less skew.

Understanding Data with Visualization

With the help of data visualization, we can see how the data looks like and what kind of correlation is held by the attributes of data.

Data Visualization Techniques:

1- Univariate Plots: Understanding Attributes Independently

The simplest type of visualization is **single-variable or "univariate"** visualization. With the help of univariate visualization, we can understand each attribute of our dataset independently.

	Univariate Plots & Description
1	Histograms Histograms group the data in bins and is the fastest way to get an idea about the distribution of each attribute in dataset.
2	Density Plots Another quick and easy technique for getting each attributes distribution is Density plots.
3	Box and Whisker Plots Box and Whisker plots, also called boxplots in short, is another useful technique to review the distribution of each attribute's distribution.

2- Multivariate Plots: Interaction Among Multiple Variables

Another type of visualization is **multivariable or "multivariate"** visualization. With the help of multivariate visualization, we can understand interaction between multiple attributes of our dataset.

	Multivariate Plots & Description
1	Correlation Matrix Plot Correlation is an indication about the changes between two variables.
2	Scatter Matrix Plot Scatter plots shows how much one variable is affected by another or the relationship between them with the help of dots in two dimensions.

Preparing Data

Data Pre-processing Techniques

1- Scaling

Most probably our dataset comprises of the attributes with varying scale, but we cannot provide such data to ML algorithm hence it requires rescaling. Data rescaling makes sure that attributes are at the same scale. Generally, attributes are rescaled into the range of 0 and 1. ML algorithms like gradient descent and k-Nearest Neighbors requires scaled data. We can rescale the data with the help of *MinMaxScaler* class of *scikit-learn* Python library.

```
from pandas import read_csv
from numpy import set_printoptions
from sklearn import preprocessing

dataframe = read_csv(pima-indians-diabetes.csv)
array = dataframe.values
data_scaler = preprocessing.MinMaxScaler(feature_range=(0,1))
data_rescaled = data_scaler.fit_transform(array)
print (data_rescaled[0:10])
```

```
>>>
[[0.4 0.7 0.6 0.4 0.  0.5 0.2 0.5 1. ]
[0.1  0.4 0.5 0.3 0.  0.4 0.1 0.2 0. ]
[0.5  0.9 0.5 0.  0.  0.3 0.3 0.2 1. ]
[0.1  0.4 0.5 0.2 0.1 0.4 0.  0.  0. ]
[0.   0.7 0.3 0.4 0.2 0.6 0.9 0.2 1. ]
[0.3  0.6 0.6 0.  0.  0.4 0.1 0.2 0. ]
[0.2  0.4 0.4 0.3 0.1 0.5 0.1 0.1 1. ]
[0.6  0.6 0.  0.  0.  0.5 0.  0.1 0. ]
[0.1  1.  0.6 0.5 0.6 0.5 0.  0.5 1. ]
[0.5  0.6 0.8 0.  0.  0.  0.1 0.6 1. ]]
```

2- Normalization

Another useful data preprocessing technique is Normalization. This is used to rescale each row of data to have a length of 1. It is mainly useful in Sparse dataset where we have lots of zeros.

Types of Normalization

In machine learning, there are two types of normalization preprocessing techniques:

- L1 Normalization
- L2 Normalization

It's a technique that modifies the dataset values in a way that in each row the sum of the absolute values will always be up to 1. It is also called Least Absolute Deviations.

```
from sklearn.preprocessing import Normalizer

array = dataframe.values
Data_normalizer = Normalizer(norm='l1').fit(array)
Data_normalized = Data_normalizer.transform(array)
print (Data_normalized [0:3])

>>>
[[0.02 0.43 0.21 0.1  0. 0.1  0. 0.14 0. ]
[0.   0.36 0.28 0.12 0. 0.11 0. 0.13 0. ]
[0.03  0.59 0.21 0.   0. 0.07 0. 0.1  0. ]]
```

3- Binarization

This is the technique with the help of which we can make our data binary. We can use a binary threshold for making our data binary. The values above that threshold value will be converted to 1 and below that threshold will be converted to 0.

This technique is useful when we have probabilities in our dataset and want to convert them into crisp values.

```
from sklearn.preprocessing import Binarizer

array = dataframe.values
binarizer = Binarizer(threshold=0.5).fit(array)
Data_binarized = binarizer.transform(array)
print (Data_binarized [0:5])
[[1. 1. 1. 1. 0. 1. 1. 1. 1.]
[1.  1. 1. 1. 0. 1. 0. 1. 0.]
[1.  1. 1. 0. 0. 1. 1. 1. 1.]
[1.  1. 1. 1. 1. 1. 0. 1. 0.]
[0.  1. 1. 1. 1. 1. 1. 1. 1.]]
```

4- Standardization

It’s basically used to transform the data attributes with a **Gaussian distribution**. It differs the mean and SD (Standard Deviation) to a standard Gaussian distribution with a mean of 0 and an SD of 1. This technique is useful in ML algorithms like **linear regression, logistic regression** that assumes a Gaussian distribution in input dataset and produce better results with rescaled data. We can standardize the data (**mean = 0 and SD =1**) with the help of *StandardScaler* class of *scikit-learn* Python library.

```
from sklearn.preprocessing import StandardScaler
array = dataframe.values
data_scaler = StandardScaler().fit(array)
data_rescaled = data_scaler.transform(array)
print (data_rescaled [0:5])

>>>
[[ 0.64 0.85  0.15  0.91 -0.69  0.2   0.47  1.43  1.37]
[-0.84 -1.12 -0.16  0.53 -0.69 -0.68 -0.37 -0.19 -0.73]
[ 1.23  1.94 -0.26 -1.29 -0.69 -1.1   0.6  -0.11  1.37]
[-0.84 -1.   -0.16  0.15  0.12 -0.49 -0.92 -1.04 -0.73]
[-1.14  0.5  -1.5   0.91  0.77  1.41  5.48 -0.02  1.37]]
```

5- Data Labeling

One more aspect in this regard is data labeling. It is also very important to send the data to ML algorithms having proper labeling. For example, in the case of classification problems, lot of labels in the form of words, numbers etc. are there on the data.

Label Encoding

Most of the sklearn functions expect that the data with number labels rather than word labels. Hence, we need to convert such labels into number labels. This process is called label encoding.

```
import numpy as np
from sklearn import preprocessing

input_labels = ['red','black','red','green','black','yellow','white']

encoder = preprocessing.LabelEncoder()
encoder.fit(input_labels)

test_labels = ['green','red','black']
encoded_values = encoder.transform(test_labels)
print("\nLabels =", test_labels)
print("Encoded values =", list(encoded_values))
encoded_values = [3,0,4,1]
decoded_list = encoder.inverse_transform(encoded_values)
 # We can get the list of encoded values
print("\nEncoded values =", encoded_values)
print("\nDecoded labels =", list(decoded_list))

>>>
Labels = ['green', 'red', 'black']
Encoded values = [1, 2, 0]
Encoded values = [3, 0, 4, 1]
Decoded labels = ['white', 'black', 'yellow', 'green']
```

Data Feature Selection

The performance of ML model will be affected negatively if the data features provided to it are irrelevant. what is automatic feature selection? It may be defined as the process with the help of which we select those features in our data that are most relevant to the output or prediction variable in which we are interested. It is also called attribute selection.

Some of the benefits of automatic feature selection:

- Performing feature selection before data modeling will reduce the overfitting.
- It will increase the accuracy of ML model.
- It will reduce the training time.

Feature Selection Techniques

A - Univariate Selection

Useful in selecting those features, with the help of statistical testing, having the strongest relationship with the prediction variables.

```
 from pandas import read_csv
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

names=['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv('pima-indians-diabetes.csv', names=names)
array = dataframe.values

#Next, we will separate array into input and output components
X = array[:,0:8]
Y = array[:,8]

 # select the best features from dataset
test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X,Y)

 # We can also summarize the data for output as per our choice. Here, we are
 setting the precision to 2 and showing the 4 data attributes with best
 features along with best score of each attribute
```

```
print(fit.scores_)
featured_data = fit.transform(X)
print (featured_data[0:4])

>>>
[ 111.52 1411.89 17.61 53.11 2175.57 127.67 5.39 181.3 ]
Featured data:
[[148.  0. 33.6 50. ]
[  85.  0. 26.6 31. ]
[ 183.  0. 23.3 32. ]
[  89. 94. 28.1 21. ]]
```

B - Recursive Feature Elimination

RFE feature selection technique removes the attributes recursively and builds the model with remaining attributes. In this example, we will use RFE with logistic regression algorithm to select the best 3 attributes having the best features.

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

# The following lines of code will select the best features from a dataset
model = LogisticRegression()
rfe = RFE(model, 3)
fit = rfe.fit(X, Y)
print("Number of Features: %d")
print("Selected Features: %s")
print("Feature Ranking: %s")

>>>
Number of Features: 3
Selected Features: [ True False False False False True True False]
Feature Ranking: [1 2 3 5 6 1 1 4]
```

We can see in the above output, RFE choose preg, mass and pedi as the first 3 best features. They are marked as 1 in the output.

C - Principal Component Analysis (PCA)

PCA, generally called data reduction technique, is very useful feature selection technique as it uses linear algebra to transform the dataset into a compressed form.

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 3)
fit = pca.fit(X)
print("Explained Variance: %s") % fit.explained_variance_ratio_
print(fit.components_)

>>>
Explained Variance: [ 0.88854663 0.06159078 0.02579012]
[[ -2.02176587e-03 9.78115765e-02 1.60930503e-02 6.07566861e-02
9.93110844e-01 1.40108085e-02 5.37167919e-04 -3.56474430e-03]
[ 2.26488861e-02 9.72210040e-01 1.41909330e-01 -5.78614699e-02
-9.46266913e-02 4.69729766e-02 8.16804621e-04 1.40168181e-01]
[ -2.24649003e-02 1.43428710e-01 -9.22467192e-01 -3.07013055e-01
2.09773019e-02 -1.32444542e-01 -6.39983017e-04 -1.25454310e-01]]
```

D - Feature Importance

It basically uses a trained supervised classifier to select features.

```
from sklearn.ensemble import ExtraTreesClassifier

model = ExtraTreesClassifier()
model.fit(X, Y)
print(model.feature_importances_)

>>>
[ 0.11070069 0.2213717 0.08824115 0.08068703 0.07281761 0.14548537
0.12654214 0.15415431]
```