# Практическая работа №2. Подготовка данных. Регрессия.

Предварительная обработка данных
В нашей повседневной жизни мы имеем дело с большим количеством данных, но эти данные в необработанном виде. Чтобы предоставить данные в качестве входных данных алгоритмов машинного обучения, нам необходимо преобразовать их в значимые данные. Вот где начинается предварительная обработка данных. Другими словами, мы можем сказать, что перед предоставлением данных в алгоритмы машинного обучения нам необходимо предварительно обработать данные.

Этапы предварительной обработки данных
Выполните следующие шаги для предварительной обработки данных в Python.

***Шаг 1 — Импорт полезных пакетов.*** Если мы используем Python, то это будет первый шаг для преобразования данных в определенный формат, т. Е. Предварительная обработка. Это можно сделать следующим образом

```
import numpy as np
import sklearn.preprocessing
```

Здесь мы использовали следующие два пакета —

- **NumPy** — в основном NumPy — это универсальный пакет для обработки массивов, предназначенный для эффективной работы с большими многомерными массивами произвольных записей без потери слишком большой скорости для небольших многомерных массивов.
- **Sklearn.preprocessing** — этот пакет предоставляет множество общих служебных функций и классов-преобразователей для преобразования необработанных векторов объектов в представление, которое больше подходит для алгоритмов машинного обучения.

***Шаг 2. Определение примеров данных.*** После импорта пакетов нам нужно определить некоторые примеры данных, чтобы мы могли применить методы предварительной обработки к этим данным. Теперь мы определим следующий пример данных

```
Input_data = np.array([2.1, -1.9, 5.5],
                      [-1.5, 2.4, 3.5],
                      [0.5, -7.9, 5.6],
                      [5.9, 2.3, -5.8])
```

***Шаг 3. Применение метода предварительной обработки.***На этом этапе нам необходимо применить любой из методов предварительной обработки.
В следующем разделе описываются методы предварительной обработки данных.

Методы предварительной обработки данных
***Бинаризация***
Это метод предварительной обработки, который используется, когда нам нужно преобразовать наши числовые значения в логические значения. Мы можем использовать встроенный метод для бинаризации входных данных, скажем, используя 0,5 в качестве порогового значения следующим образом:

```
data_binarized = preprocessing.Binarizer(threshold = 0.5).transform(input_data)
print("\nBinarized data:\n", data_binarized)
```

Теперь, после выполнения вышеуказанного кода, мы получим следующий вывод: все значения выше 0,5 (пороговое значение) будут преобразованы в 1, а все значения ниже 0,5 будут преобразованы в 0.

```
[[ 1. 0. 1.]
[ 0. 1. 1.]
[ 0. 0. 1.]
[ 1. 1. 0.]]
```

***Среднее удаление***

Это еще один очень распространенный метод предварительной обработки, который используется в машинном обучении. В основном это используется, чтобы исключить среднее значение из вектора объекта, так что каждый объект центрируется на нуле. Мы также можем удалить отклонение от объектов в векторе объектов. Для применения метода предварительной обработки среднего удаления к образцу данных мы можем написать код Python, показанный ниже. Код отобразит среднее и стандартное отклонение входных данных

```
print("Mean = ", input_data.mean(axis = 0))
print("Std deviation = ", input_data.std(axis = 0))
```

Мы получим следующий вывод после выполнения вышеуказанных строк кода:

```
         Mean = [ 1.75       -1.275       2.2]
Std deviation = [ 2.71431391  4.20022321  4.69414529]
```

Теперь код ниже удалит среднее и стандартное отклонение входных данных —

```
data_scaled = preprocessing.scale(input_data)
print("Mean =", data_scaled.mean(axis=0))
print("Std deviation =", data_scaled.std(axis = 0))
```

Мы получим следующий вывод после выполнения вышеуказанных строк кода:

```
Mean = [ 1.11022302e-16 0.00000000e+00 0.00000000e+00]
Std deviation = [ 1.             1.             1.]
```

***Пересчет***

Это еще один метод предварительной обработки данных, который используется для масштабирования векторов признаков. Масштабирование векторов объектов необходимо, поскольку значения каждого объекта могут варьироваться между многими случайными значениями. Другими словами, мы можем сказать, что масштабирование важно, потому что мы не хотим, чтобы какая-либо функция была синтетически большой или маленькой. С помощью следующего кода на Python мы можем выполнить масштабирование наших входных данных, то есть вектор признаков

```
data_scaler_minmax = preprocessing.MinMaxScaler(feature_range=(0,1))
data_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
print ("\nMin max scaled data:\n", data_scaled_minmax)
```

Мы получим следующий вывод после выполнения вышеуказанных строк кода:

```
[ [ 0.48648649  0.58252427   0.99122807]
[   0.          1.           0.81578947]
[   0.27027027  0.           1.        ]
[   1.          0. 99029126  0.        ]]
```

***Нормализация***
Это еще один метод предварительной обработки данных, который используется для изменения векторов признаков. Такая модификация необходима для измерения векторов признаков в общем масштабе. Ниже приведены два типа нормализации, которые можно использовать в машинном обучении.

***L1 нормализация***
Это также упоминается как **Наименее Абсолютные Отклонения** . Этот тип нормализации изменяет значения так, что сумма абсолютных значений всегда равна 1 в каждой строке. Это может быть реализовано на входных данных с помощью следующего кода Python

```
# Normalize data
data_normalized_l1 = preprocessing.normalize(input_data, norm = 'l1')
print("\nL1 normalized data:\n", data_normalized_l1)
```

Приведенная выше строка кода генерирует следующие выходные данные

```
L1 normalized data:
[[ 0.22105263  -0.2          0.57894737]
[ -0.2027027    0.32432432   0.47297297]
[  0.03571429  -0.56428571   0.4       ]
[  0.42142857   0.16428571  -0.41428571]]
```

***L2 нормализация***
Это также упоминается как **наименьших квадратов** . Этот тип нормализации изменяет значения так, что сумма квадратов всегда равна 1 в каждой строке. Это может быть реализовано на входных данных с помощью следующего кода Python

```
# Normalize data
data_normalized_l2 = preprocessing.normalize(input_data, norm = 'l2')
print("\nL2 normalized data:\n", data_normalized_l2)
```

Выше строка кода будет генерировать следующий вывод —

```
L2 normalized data:
[[ 0.33946114  -0.30713151   0.88906489]
[ -0.33325106   0.53320169   0.7775858 ]
[  0.05156558  -0.81473612   0.57753446]
[  0.68706914   0.26784051  -0.6754239 ]]
```

***Маркировка данных***
Мы уже знаем, что данные в определенном формате необходимы для алгоритмов машинного обучения. Другим важным требованием является то, что данные должны быть правильно помечены перед отправкой в качестве входных данных алгоритмов машинного обучения. Например, если мы говорим о классификации, на данных много меток. Эти метки представлены в виде слов, чисел и т. Д. Функции, связанные с машинным обучением в **sklearn,** предполагают, что данные должны иметь **числовые** метки. Следовательно, если данные представлены в другой форме, они должны быть преобразованы в числа. Этот процесс преобразования меток слов в числовую форму называется кодированием меток.

***Шаги кодирования меток***
Выполните следующие шаги для кодирования меток данных в Python.

***Шаг 1 — Импорт полезных пакетов***

Если мы используем Python, то это будет первый шаг для преобразования данных в определенный формат, т. Е. Предварительная
обработка. Этоможносделатьследующимобразом

```
importnumpyasnp
from sklearn import preprocessing
```

***Шаг 2 — Определение образцов меток***
После импорта пакетов нам нужно определить некоторые образцы меток, чтобы мы могли создавать и обучать кодировщик меток. Теперь мы определим следующие образцы меток

```
# Sample input labels
input_labels = ['red','black','red','green','black','yellow','white']
```

***Шаг 3 — Создание и обучение объекта кодировщика меток***
На этом этапе нам нужно создать кодировщик меток и обучить его. Следующий код Python поможет в этом

```
# Creating the label encoder
encoder = preprocessing.LabelEncoder()
encoder.fit(input_labels)
```

Следующим будет вывод после запуска приведенного выше кода Python

```
LabelEncoder()
```

***Шаг 4 — Проверка производительности путем кодирования случайного упорядоченного списка***
Этот шаг можно использовать для проверки производительности путем кодирования случайного упорядоченного списка. Следующий код Python может быть написан, чтобы сделать то же самое

```
# encoding a set of labels
test_labels = ['green','red','black']
encoded_values = encoder.transform(test_labels)
print("\nLabels =", test_labels)
```

Этикетки будут напечатаны следующим образом

```
Labels = ['green', 'red', 'black']
```

Теперь мы можем получить список закодированных значений, т. е. метки слов, преобразованные в числа следующим образом:

```
print("Encoded values =", list(encoded_values))
```

Закодированные значения будут напечатаны следующим образом:

```
Encoded values = [1, 2, 0]
```

***Шаг 5 — Проверка производительности путем декодирования случайного набора чисел***
Этот шаг можно использовать для проверки производительности путем декодирования случайного набора чисел. Следующий код Python может быть написан, чтобы сделать то же самое

```
# decoding a set of values
```

```
encoded_values = [3,0,4,1]
decoded_list = encoder.inverse_transform(encoded_values)
print("\nEncoded values =", encoded_values)
```

Теперь закодированные значения будут напечатаны следующим образом:

```
Encoded values = [3, 0, 4, 1]
print("\nDecoded labels =", list(decoded_list))
```

Теперь декодированные значения будут напечатаны следующим образом:

```
Decoded labels = ['white', 'black', 'yellow', 'green']
```

**Линейный регрессор / регрессор с одной переменной**

Давайте отметим несколько необходимых пакетов

```
importnumpyasnp
from sklearn import linear_model
import sklearn.metrics as sm
importmatplotlib.pyplotasplt
```

Теперь нам нужно предоставить входные данные, и мы сохранили наши данные в файле с именем linear.txt.

```
input = 'D:/ProgramData/linear.txt'
```

Нам нужно загрузить эти данные с помощью функции **np.loadtxt** .

```
input_data = np.loadtxt(input, delimiter=',')
X, y = input_data[:, :-1], input_data[:, -1]
```

Следующим шагом будет обучение модели. Давайте дадим учебные и испытательные образцы.

```
training_samples = int(0.6 * len(X))
testing_samples = len(X) - num_training

X_train, y_train = X[:training_samples], y[:training_samples]

X_test, y_test = X[training_samples:], y[training_samples:]
```

Теперь нам нужно создать объект линейного регрессора.

```
reg_linear = linear_model.LinearRegression()
```

Тренируйте объект с обучающими образцами.

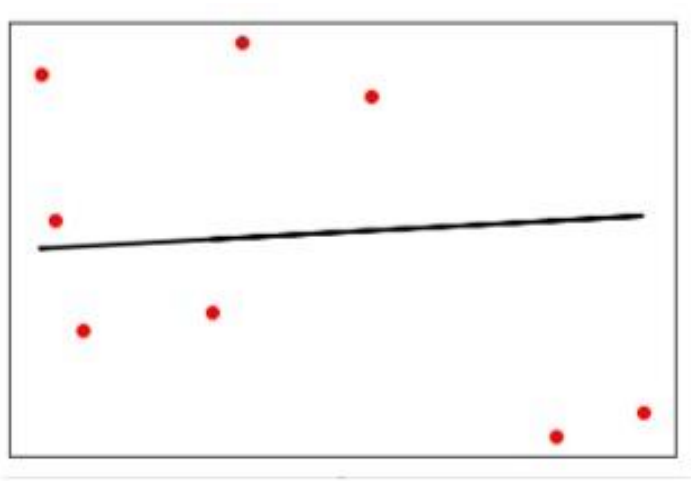
```
reg_linear.fit(X_train, y_train)
```

Нам нужно сделать прогноз с данными тестирования.

```
y_test_pred = reg_linear.predict(X_test)
```

Теперь постройте и визуализируйте данные.

```
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_test, y_test_pred, color = 'black', linewidth = 2)
plt.xticks(())
plt.yticks(())
plt.show()
```



Теперь мы можем вычислить производительность нашей линейной регрессии следующим образом:

```
print("Performance of Linear regressor:")
print("Mean absolute error =", round(sm.mean_absolute_error(y_test, y_test_pred), 2))
print("Mean squared error =", round(sm.mean_squared_error(y_test, y_test_pred), 2))
print("Median abs error =", round(sm.median_absolute_error(y_test, y_test_pred), 2))
print("Explain var scr =", round(sm.explained_variance_score(y_test, y_test_pred),2))
print("R2 score =", round(sm.r2_score(y_test, y_test_pred), 2))
```

**Многовариантный регрессор**

Во-первых, давайте импортируем несколько необходимых пакетов

```
import numpy as np
from sklearn import linear_model
import sklearn.metrics as sm
import matplotlib.pyplot as plt
fromsklearn.preprocessingimportPolynomialFeatures
```

Теперь нам нужно предоставить входные данные, и мы сохранили наши данные в файле с именем linear.txt.

```
input = 'D:/ProgramData/Mul_linear.txt'
```

Мы загрузим эти данные с помощью функции np.loadtxt .

```
input_data = np.loadtxt(input, delimiter=',')
X, y = input_data[:, :-1], input_data[:, -1]
```

Следующим шагом будет обучение модели; Мы дадим образцы для обучения и тестирования.

```
training_samples = int(0.6 * len(X))
testing_samples = len(X) - num_training
X_train, y_train = X[:training_samples], y[:training_samples]
X_test, y_test = X[training_samples:], y[training_samples:]
```

Теперь нам нужно создать объект линейного регрессора.

```
reg_linear_mul = linear_model.LinearRegression()
```

Тренируйте объект с обучающими образцами.

```
reg_linear_mul.fit(X_train, y_train)
```

Теперь, наконец, нам нужно сделать прогноз с данными тестирования.

```
y_test_pred = reg_linear_mul.predict(X_test)
print("Performance of Linear regressor:")
print("Mean absolute error =", round(sm.mean_absolute_error(y_test, y_test_pred), 2))
print("Mean squared error =", round(sm.mean_squared_error(y_test, y_test_pred), 2))
print("Median abs error =", round(sm.median_absolute_error(y_test, y_test_pred), 2))
print("Explain var scr=", round(sm.explained_variance_score(y_test, y_test_pred), 2))
print("R2 score =", round(sm.r2_score(y_test, y_test_pred), 2))
```

Теперь мы создадим полином 10 степени и обучим регрессора.

```
polynomial = PolynomialFeatures(degree = 10)
X_train_transformed = polynomial.fit_transform(X_train)
datapoint = [[2.23, 1.35, 1.12]]
poly_datapoint = polynomial.fit_transform(datapoint)
poly_linear_model = linear_model.LinearRegression()
poly_linear_model.fit(X_train_transformed, y_train)
print("\nLinear regression:\n", reg_linear_mul.predict(datapoint))
print("\nPolynomial regression:\n", poly_linear_model.predict(poly_datapoint))
```