logistic-regression

1 Basic Example of using Logistic Regression

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import seaborn
%matplotlib inline
```

[2]:
```
df = pd.read_csv('marital status.csv')
```

[3]:
```
df
```

[3]:

	age	marital_status
0	25	Single
1	30	Married
2	NaN	Single
3	45	Divorced
4	35	Married
5	28	Single
6	40	NaN
7	55	Divorced
8	,	Married
9	27	Single

2 Checking Null/Nan Values

[4]:
```
df.isnull().sum()
```

[4]:
```
age              1
marital_status   1
dtype: int64
```

[5]:
```
df = pd.read_csv('marital status_Log_Reg.csv')
```

[6]:
```
df
```

[6]:

	age	marital_status
0	25	0.0
1	30	1.0
2	60	0.0
3	45	1.0
4	35	1.0
5	28	0.0
6	40	1.0
7	55	1.0
8	37	1.0
9	27	0.0
10	26	0.0
11	32	1.0
12	35	0.0
13	47	1.0
14	37	1.0
15	29	0.0
16	41	NaN
17	53	1.0
18	45	1.0
19	25	0.0

[7]:
```
df.isnull().sum()
```

[7]:
```
age              0
marital_status    1
dtype: int64
```

3 Filling null values of age column

[8]:
```
updated_df = df
updated_df['age']=updated_df['age'].fillna(updated_df['age'].mean())
updated_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 2 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             20 non-null     int64
 1   marital_status  19 non-null     float64
dtypes: float64(1), int64(1)
memory usage: 452.0 bytes
```

[9]:
```
handle = df['marital_status'].median()
```

[10]:
```
handle
```

[10]:
```
1.0
```

4 filling null values with median means handle

[11]:
```
df.marital_status=  df.marital_status.fillna(handle)
```

[12]:
```
df
```

[12]:

	age	marital_status
0	25	0.0
1	30	1.0
2	60	0.0
3	45	1.0
4	35	1.0
5	28	0.0
6	40	1.0
7	55	1.0
8	37	1.0
9	27	0.0
10	26	0.0
11	32	1.0
12	35	0.0
13	47	1.0
14	37	1.0
15	29	0.0
16	41	1.0
17	53	1.0
18	45	1.0
19	25	0.0

[13]:
```
df.isnull().sum()
```

[13]:
```
age               0
marital_status    0
dtype: int64
```

[14]:
```
df.marital_status.value_counts()
```

[14]:
```
marital_status
1.0    12
0.0     8
Name: count, dtype: int64
```

[15]:
```
x= df[['age']]
```

[16]:
```
x
```

[16]:
```
    age
0    25
1    30
2    60
3    45
4    35
5    28
6    40
7    55
8    37
9    27
10   26
11   32
12   35
13   47
14   37
15   29
16   41
17   53
18   45
19   25
```

5 y value is basically level , so we place single [] after df but x is features so we place [[]] after df though using [[]] with also diplay y's values

like below

y= df[‘marital_status’] x= df[[‘age’]]

[17]:
```
y= df['marital_status']
```

[18]:
```
y
```

[18]:
```
0     0.0
1     1.0
2     0.0
3     1.0
4     1.0
5     0.0
6     1.0
7     1.0
8     1.0
9     0.0
```

```
10    0.0
11    1.0
12    0.0
13    1.0
14    1.0
15    0.0
16    1.0
17    1.0
18    1.0
19    0.0
Name: marital_status, dtype: float64
```

6 Now we need to split dataset for train and test

[19]:
```
from sklearn.model_selection import train_test_split
```

[20]:
```
#xtrain,ytrain,xtest,ytest = train_test_split(x,y,test_size= .20, random_state
 ↪= 1)
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.30,
 ↪random_state =1)
```

[21]:
```
xtest
```

[21]:
```
    age
3    45
16   41
6    40
10   26
2    60
14   37
```

[22]:
```
ytest
```

[22]:
```
3     1.0
16    1.0
6     1.0
10    0.0
2     0.0
14    1.0
Name: marital_status, dtype: float64
```

[23]:
```
from sklearn.linear_model import LogisticRegression
```

[24]:
```
model = LogisticRegression ()
```

7 Now Training the model

[25]:
```
model.fit(xtrain, ytrain)
```

[25]:
```
LogisticRegression()
```

[26]:
```
model.predict(xtest)
```

[26]:
```
array([1., 1., 1., 0., 1., 1.])
```

[27]:
```
#Finding out accuracy
model.score(xtest,ytest)
```

[27]:
```
0.8333333333333334
```

[28]:
```
#predicting performance using sigmoid function
model.predict_proba(xtest)

# here first values in the array is for no, second value is for yes
```

[28]:
```
array([[4.28664393e-03, 9.95713356e-01],
       [2.17640342e-02, 9.78235966e-01],
       [3.24559767e-02, 9.67544023e-01],
       [9.13231426e-01, 8.67685743e-02],
       [9.10031684e-06, 9.99990900e-01],
       [1.03119980e-01, 8.96880020e-01]])
```

Predicting status of anyother single value providing age

[30]:
```
# Predicting status of anyother single value providing age  (one of the ways)
# Assuming you have a trained logistic regression model called 'model'
# model = LogisticRegression() # Your trained model here

def predict_marriage_status(model, age):
    # Prepare the input as a 2D array
    age_input = np.array([[age]])

    # Get the prediction: 1 for married (Yes), 0 for not married (No)
    prediction = model.predict(age_input)[0]

    # Convert prediction to readable format
    return "Yes" if prediction == 1 else "No"

# Example usage
age = 42  # Replace with any age you want to predict
status = predict_marriage_status(model, age)
```

```
print(f"Prediction: {status}")
```

```
Prediction: Yes

C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning: X
does not have valid feature names, but LogisticRegression was fitted with
feature names
  warnings.warn(
```

[33]:
```
# Predicting status of anyother single value providing age  (second another way)
age = 30  # Replace with any age you want to predict
status = "Yes" if model.predict([[age]])[0] == 1 else "NO"

print(f"Prediction: {status}")
```

```
Prediction: NO

C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning: X
does not have valid feature names, but LogisticRegression was fitted with
feature names
  warnings.warn(
```

[40]:
```
# Predicting status of anyother single value providing age  (third another way)
# Assuming you have a trained logistic regression model called 'model'
# model = LogisticRegression() # Your trained model here

# Predict marriage status for a given age
age = 30  # Replace with any age you want to predict
status = ["No", "Yes"] [int(model.predict([[age]])[0]]]

print(f"Prediction: {status}")
```

```
  Cell In[40], line 7
    status = ["No", "Yes"] [int(model.predict([[age]])[0]]]
                                                        ^
SyntaxError: closing parenthesis ']' does not match opening parenthesis '('
```

[36]:
```
# Predicting status of anyother single value providing age  (fourth another␣
↪way) >> i prefer
# Assuming you have a trained logistic regression model called 'model'
# model = LogisticRegression() # Your trained model here

# Predict marriage status for a given age
#age = 30  # Replace with any age you want to predict
#status = ["No", "Yes"][model.predict([[age]])[0]]
#print(f"Prediction: {status}")
```

```
# Assuming you have a trained logistic regression model called 'model'
# model = LogisticRegression() # Your trained model here

# Predict marriage status for a given age
age = 30  # Replace with any age you want to predict
status = ["No", "Yes"][int(model.predict([[age]])[0])]

print(f"Prediction: {status}")
```

```
Prediction: No

C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning: X
does not have valid feature names, but LogisticRegression was fitted with
feature names
  warnings.warn(
```

[ ]:

SVM Classifier

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Load the dataset
data = pd.read_csv('svm.csv')

data
```

	Length	Weight	Snout Width	Habitat	Species
0	3.4	400	20	Saltwater	Crocodile
1	4.2	500	22	Freshwater	Alligator
2	3.9	450	19	Saltwater	Crocodile
3	4.5	600	24	Freshwater	Alligator
4	2.8	300	21	Saltwater	Crocodile
5	3.6	480	23	Freshwater	Alligator
6	5.0	700	20	Saltwater	Crocodile
7	4.1	550	25	Freshwater	Alligator
8	3.3	420	18	Saltwater	Crocodile
9	4.3	530	24	Freshwater	Alligator

```
# Encode the categorical 'Habitat' and 'Species' columns
label_encoder = LabelEncoder()
data['Habitat'] = label_encoder.fit_transform(data['Habitat'])  #
Convert habitat to numeric
data['Species'] = label_encoder.fit_transform(data['Species'])  #
Convert species to numeric

data
```

	Length	Weight	Snout Width	Habitat	Species
0	3.4	400	20	1	1
1	4.2	500	22	0	0
2	3.9	450	19	1	1
3	4.5	600	24	0	0
4	2.8	300	21	1	1
5	3.6	480	23	0	0
6	5.0	700	20	1	1
7	4.1	550	25	0	0
8	3.3	420	18	1	1
9	4.3	530	24	0	0

```
# Separate features and target
X = data[['Length', 'Weight', 'Snout Width', 'Habitat']]
y = data['Species']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Standardize the feature values
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and train the SVM model
svm_model = SVC(kernel='linear', C=1.0)  # Linear kernel SVM for
binary classification
svm_model.fit(X_train, y_train)
```

```
SVC(kernel='linear')
```

```
# Make predictions on the test set
y_pred = svm_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred,
target_names=['Alligator', 'Crocodile'])

# Output the results
print("Accuracy:", accuracy)
print("Classification Report:\n", report)
```

```
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

   Alligator       1.00      1.00      1.00         2
   Crocodile       1.00      1.00      1.00         1

    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3
```

```
# Define the habitat mapping outside the function
habitat_mapping = {
    'Freshwater': 0,
    'Saltwater': 1
}

# Define species mapping as well, for completeness
species_mapping = {0: 'Alligator', 1: 'Crocodile'}
```

```
# Function to predict whether a new input is a Crocodile or Alligator
def predict_species(length, weight, snout_width, habitat):
    # Convert habitat to numeric using the previously saved mapping
    if habitat in habitat_mapping:
        habitat_num = habitat_mapping[habitat]
    else:
        print("Invalid habitat. Please use 'Freshwater' or
'Saltwater'.")
        return

    # Prepare the feature vector and scale it
    new_data = scaler.transform([[length, weight, snout_width,
habitat_num]])

    # Predict using the SVM model
    prediction = svm_model.predict(new_data)

    # Map the numeric prediction back to species name
    species_name = species_mapping[prediction[0]]
    print(f"The predicted species is: {species_name}")

# Example input to check the species
# Example: Length=4.0 meters, Weight=500 kg, Snout Width=22 cm,
Habitat='Freshwater'
predict_species(4.0, 500, 22, 'Freshwater')
```

```
The predicted species is: Alligator
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:464:
UserWarning: X does not have valid feature names, but StandardScaler
was fitted with feature names
  warnings.warn(
```

```
predict_species(3.4, 400, 20, 'Saltwater')
```

```
The predicted species is: Crocodile
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:464:
UserWarning: X does not have valid feature names, but StandardScaler
was fitted with feature names
  warnings.warn(
```