Multivariate Logistic Regression

```
# importing the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style('whitegrid')
```

```
# loading the breast cancer dataset
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()

print(cancer.DESCR)
```

```
        area (worst):                         185.2  4254.0
        smoothness (worst):                   0.071  0.223
        compactness (worst):                  0.027  1.058
        concavity (worst):                    0.0    1.252
        concave points (worst):               0.0    0.291
        symmetry (worst):                     0.156  0.664
        fractal dimension (worst):            0.055  0.208
        ===================================== ====== ======

        :Missing Attribute Values: None

        :Class Distribution: 212 - Malignant, 357 - Benign

        :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

        :Donor: Nick Street

        :Date: November, 1995

    This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
    https://goo.gl/U2Uwz2

    Features are computed from a digitized image of a fine needle
    aspirate (FNA) of a breast mass.  They describe
    characteristics of the cell nuclei present in the image.

    Separating plane described above was obtained using
    Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
    Construction Via Linear Programming." Proceedings of the 4th
    Midwest Artificial Intelligence and Cognitive Science Society,
    pp. 97-101, 1992], a classification method which uses linear
    programming to construct a decision tree.  Relevant features
    were selected using an exhaustive search in the space of 1-4
    features and 1-3 separating planes.

    The actual linear program used to obtain the separating plane
    in the 3-dimensional space is that described in:
    [K. P. Bennett and O. L. Mangasarian: "Robust Linear
    Programming Discrimination of Two Linearly Inseparable Sets",
    Optimization Methods and Software 1, 1992, 23-34].

    This database is also available through the UW CS ftp server:

    ftp ftp.cs.wisc.edu
    cd math-prog/cpo-dataset/machine-learn/WDBC/

    .. topic:: References

       - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction
         for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
         Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
         San Jose, CA, 1993.
       - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and
         prognosis via linear programming. Operations Research, 43(4), pages 570-577,
         July-August 1995.
       - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques
         to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994)
         163-171.
```

```
# creating a dataframe
cancer_df = pd.DataFrame(cancer['data'], columns=cancer['feature_names'])
cancer_df['target'] = cancer['target']

# checking the head of the dataframe
cancer_df.head(10)
```

an ss	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity
40	0.27760	0.30010	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622	0.6656	0.7119
74	0.07864	0.08690	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238	0.1866	0.2416
60	0.15990	0.19740	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444	0.4245	0.4504
50	0.28390	0.24140	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098	0.8663	0.6869
30	0.13280	0.19800	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374	0.2050	0.4000
80	0.17000	0.15780	0.08089	0.2087	0.07613	...	23.75	103.40	741.6	0.1791	0.5249	0.5355
63	0.10900	0.11270	0.07400	0.1794	0.05742	...	27.66	153.20	1606.0	0.1442	0.2576	0.3784
90	0.16450	0.09366	0.05985	0.2196	0.07451	...	28.14	110.60	897.0	0.1654	0.3682	0.2678
30	0.19320	0.18590	0.09353	0.2350	0.07389	...	30.73	106.20	739.3	0.1703	0.5401	0.5390
60	0.23960	0.22730	0.08543	0.2030	0.08243	...	40.68	97.65	711.4	0.1853	1.0580	1.1050

```
# splitting the data into x and y
X = cancer.data
y = cancer.target

print(X.shape)
print(y.shape)
```

```
(569, 30)
(569,)
```

```
# shuffling the data to avoid any bias
def shuffle_data(X, y):
    # Create an array of indices ranging from 0 to X.shape[0] - 1
    idx = np.arange(X.shape[0])
    # Shuffle the indices in place
    np.random.shuffle(idx)
    # Index X and y using the shuffled indices to return the shuffled matrix and labels
    return X[idx], y[idx]

X, y = shuffle_data(X, y)
```

```
# splitting the data into train ,dev and test sets
from sklearn.model_selection import train_test_split

# split the dataset into train/dev/test sets (60/20/20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_dev, y_train, y_dev = train_test_split(X_train, y_train, test_size=0.25, random_state=42)

print(f"X_train.shape: {X_train.shape}")
print(f"y_test.shape: {X_test.shape}")
print(f"X_dev.shape: {X_dev.shape}")
```

```
X_train.shape: (341, 30)
y_test.shape: (114, 30)
X_dev.shape: (114, 30)
```

```
# scaling the data
from sklearn.preprocessing import StandardScaler
#scaler = StandardScaler()
#X_train = scaler.fit_transform(X_train)
```

```
# checking for class imbalance
print('Number of samples in class 0: {}'.format(np.sum(y_train == 0)))
print('Number of samples in class 1: {}'.format(np.sum(y_train == 1)))

print('Proportion of samples in class 0: {}'.format(round(np.sum(y_train == 0) / len(y_train),2)))
```

```
Number of samples in class 0: 117
Number of samples in class 1: 224
Proportion of samples in class 0: 0.34
```

Modelling

```python
# implementing multivariate logistic regression from scratch
class LogisticRegression:
    def __init__(self, learning_rate=0.01, n_iters=1000):
        # Constructor to initialize hyperparameters
        self.lr = learning_rate
        self.n_iters = n_iters
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        # Method to train the model using input matrix X and labels y
        n_samples, n_features = X.shape
        # initialize weights to 0
        self.weights = np.zeros(n_features)
        # initialize bias to 0
        self.bias = 0

        # Gradient descent optimization to update weights and bias
        for _ in range(self.n_iters):
            # compute linear model
            linear_model = np.dot(X, self.weights) + self.bias
             # apply sigmoid activation function
            y_predicted = self._sigmoid(linear_model)

            # compute gradient of weights and bias
            dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y))
            db = (1 / n_samples) * np.sum(y_predicted - y)

            # Update weights and bias
            self.weights -= self.lr * dw
            self.bias -= self.lr * db

    def predict(self, X):
        # Method to make predictions on input matrix X
        linear_model = np.dot(X, self.weights) + self.bias  # compute linear model
        y_predicted = self._sigmoid(linear_model)  # apply sigmoid activation function
        y_predicted_cls = [1 if i > 0.5 else 0 for i in y_predicted]  # threshold predicted probabilities
        return y_predicted_cls

    def _sigmoid(self, x):
        # Helper method to compute sigmoid activation function
        return 1 / (1 + np.exp(-x))
```

```python
# training the model
model = LogisticRegression(learning_rate=0.0001, n_iters=1000)
model.fit(X_train, y_train)

# predicting the values
y_pred_dev = model.predict(X_dev)
```

Model Evaluation

```python
# Evaluating the model
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, roc_curve

print("Dev set Confusion Matrix: \nTN, FP,\nFN, TP\n")
print(confusion_matrix(y_dev, y_pred_dev))
print(classification_report(y_dev, y_pred_dev))
print(accuracy_score(y_dev, y_pred_dev))

# plotting the roc curve
print("\n")
y_pred_prob = model.predict(X_dev)
fpr, tpr, thresholds = roc_curve(y_dev, y_pred_prob)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr, label='Logistic Regression')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Logistic Regression ROC Curve for Dev set')
plt.show()
```

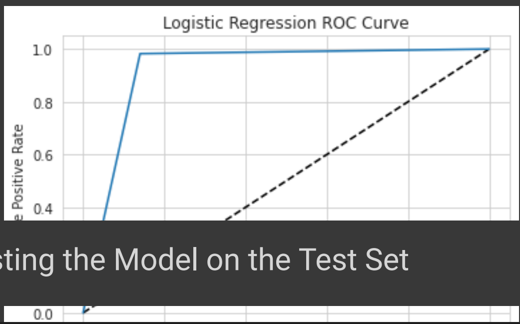
```
Dev set Confusion Matrix:
TN, FP,
FN, TP

[[49  8]
 [ 1 56]]
              precision    recall  f1-score   support

           0       0.98      0.86      0.92        57
           1       0.88      0.98      0.93        57

    accuracy                           0.92       114
   macro avg       0.93      0.92      0.92       114
weighted avg       0.93      0.92      0.92       114

0.9210526315789473
```



Testing the Model on the Test Set

```
# testing the model on the dev set
y_pred_test = model.predict(X_test)

# Evaluating the model
print("Test set Confusion Matrix: \nTN, FP,\nFN, TP\n")
print(confusion_matrix(y_test, y_pred_test))
print(classification_report(y_test, y_pred_test))
print(accuracy_score(y_test, y_pred_test))

# plotting the roc curve
print("\n")
y_pred_prob2 = model.predict(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob2)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr, label='Logistic Regression')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Logistic Regression ROC Curve for Test set')
plt.show()
```

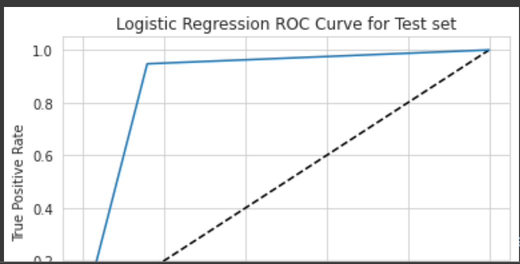
The model performed well with accuracy of 91% and an F1-score of 94% on the test data

```
[[32  6]
 [ 4 72]]
              precision    recall  f1-score   support

           0       0.89      0.84      0.86        38
           1       0.92      0.95      0.94        76

    accuracy                           0.91       114
   macro avg       0.91      0.89      0.90       114
weighted avg       0.91      0.91      0.91       114

0.9122807017543859
```



aid products - Cancel contracts here

✓ 0s completed at 2:14 AM