Day50 _Logistic_Regression_Classification

July 23, 2025

Logistic Regression

Welcome!

Today, we're going to learn one of the most important and widely used classification algorithms in machine learning — **Logistic Regression**.

While the name includes "regression," don't let it confuse you — logistic regression is actually used for **classification problems**.

We'll cover:

- What is logistic regression and why it's used
- Difference between classification and regression
- Key evaluation metrics like **confusion matrix**, **accuracy**, **precision**, **recall**, and **F1 score**
- Implementing logistic regression in Python step-by-step (without using loops/functions)
- Comparing performance using different preprocessing techniques like **StandardScaler** and **Normalizer**
- Checking **bias** and **variance**
- Ending with a summary and real-world recommendations

Let's get started and understand how logistic regression helps in **predicting categories** like: - Will a customer buy or not? - Is an email spam or not? - Will a student pass or fail?

1 Introduction to Classification

- Classification is used when the **dependent variable is binary** (e.g., Yes/No, 1/0).
- Unlike regression (which predicts continuous values), classification predicts discrete categories.
- The performance of **classification** is evaluated using the **confusion matrix**, not
- The performance of **Regression** is evaluated using the **R^2 or Adjusted R^2**.

1.1 Steps to Build Classification Model:

1. Split data into x_train, x_test, y_train, y_test
2. Train model on x_train and y_train
3. Predict y_pred using x_test
4. Evaluate using y_test vs y_pred

2 Understanding Confusion Matrix

- Confusion Matrix helps compare actual labels vs predicted labels:

  - TP (True Positive)
  - TN (True Negative)
  - FP (False Positive)
  - FN (False Negative)

Example: Diagnosing COVID

Actual (Patient) vs Predicted (Doctor) - Patient no COVID, Doctor says no COVID: TN - Patient no COVID, Doctor says yes COVID: FP - Patient yes COVID, Doctor says no COVID: FN - Patient yes COVID, Doctor says yes COVID: TP

Actual Predicted	No COVID (0)	Yes COVID (1)
No COVID (0)	TN	FP
Yes COVID (1)	FN	TP

2.1 Key Metrics:

- Accuracy = (TP + TN) / Total
- Error Rate = (FP + FN) / Total
- Precision = TP / (TP + FP)
- Recall = TP / (TP + FN)
- F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

Sometimes TN and TP can flip depending on interpretation. Always define clearly.

Is Logistic Regression a classification algorithm? (Short) - YES - It uses a regression line to separate two classes - Applies a sigmoid function to model probabilities - Based on threshold (like 0.5), it classifies outputs

Also used in deep learning as Sigmoid Activation

Logistic Regression = MaxEnt Classifier

- y * mx > 0 => Correct classification
- y * mx < 0 => Misclassified

Is Logistic Regression a Classification Algorithm?

Yes, Logistic Regression **is** a classification algorithm. Here's the explanation:

1. **Despite the name**, Logistic Regression is **not** used for regression problems (predicting continuous values). It's used when the output variable is **categorical**, typically binary (0 or 1).

2. Logistic Regression works by drawing a **regression line** that separates two classes.

3. It then applies a **sigmoid function** (also called the logistic function) to convert the linear output into a probability between 0 and 1.

4. If the output probability is **greater than 0.5**, the model predicts class 1. If it's **less than 0.5**, it predicts class 0. You can change this threshold if needed.

5. This is why it's considered a **probability-based classification model**.

6. Logistic Regression is also used in **deep learning** under the name **sigmoid activation function**.
7. The algorithm is also known as the **Maximum Entropy (MaxEnt) Classifier**.

3 What is Logistic Regression?

Logistic Regression:

- Though the name has "regression", it's a **classification** algorithm.
- It models the probability that a given input belongs to a particular class.
- Logistic regression uses a **sigmoid (S-shaped)** curve to separate two classes (0 and 1).
- It works well for binary classification (like "Purchased" or "Not Purchased").

4 Importing Libraries

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, Normalizer
from sklearn.metrics import confusion_matrix, accuracy_score
```

5 Load and Explore Dataset

[2]:
```
dataset = pd.read_csv(r"C:\Users\Lenovo\Downloads\logit classification.csv") #␣
↪Change path if needed
print(dataset.head())
```

```
    User ID  Gender  Age  EstimatedSalary  Purchased
0  15624510    Male   19            19000          0
1  15810944    Male   35            20000          0
2  15668575  Female   26            43000          0
3  15603246  Female   27            57000          0
4  15804002    Male   19            76000          0
```

[3]:
```
print(dataset.isnull().sum())
```

```
User ID            0
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

[4]:
```
print(dataset.shape)
```

```
(400, 5)
```

6 Prepare Data

[5]:
```python
# Feature Selection
X = dataset.iloc[:, [2, 3]].values  # Age and EstimatedSalary
y = dataset.iloc[:, -1].values      # Purchased
```

7 Try 1: No Scaling, random_state = 0, test_size = 0.25

[6]:
```python
# Try 1: No Scaling, random_state = 0, test_size = 0.25

X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=0)
model1 = LogisticRegression()
model1.fit(X_train1, y_train1)
y_pred1 = model1.predict(X_test1)
cm1 = confusion_matrix(y_test1, y_pred1)
acc1 = accuracy_score(y_test1, y_pred1)
print("Confusion Matrix:\n", cm1)
print("Accuracy:", acc1)
```

```
Confusion Matrix:
 [[65  3]
 [ 8 24]]
Accuracy: 0.89
```

7.1 Try 2: StandardScaler, random_state = 0, test_size = 0.25

[7]:
```python
# Try 2: StandardScaler, random_state = 0, test_size = 0.25

scaler2 = StandardScaler()
X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=0)
X_train2 = scaler2.fit_transform(X_train2)
X_test2 = scaler2.transform(X_test2)
model2 = LogisticRegression()
model2.fit(X_train2, y_train2)
y_pred2 = model2.predict(X_test2)
cm2 = confusion_matrix(y_test2, y_pred2)
acc2 = accuracy_score(y_test2, y_pred2)
print("Confusion Matrix:\n", cm2)
print("Accuracy:", acc2)
```

```
Confusion Matrix:
 [[65  3]
```

```
 [ 8 24]]
Accuracy: 0.89
```

7.2 Try 3: Normalizer, random_state = 0, test_size = 0.25

[8]:
```python
# Try 3: Normalizer, random_state = 0, test_size = 0.25

norm3 = Normalizer()
X_train3, X_test3, y_train3, y_test3 = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=0)
X_train3 = norm3.fit_transform(X_train3)
X_test3 = norm3.transform(X_test3)
model3 = LogisticRegression()
model3.fit(X_train3, y_train3)
y_pred3 = model3.predict(X_test3)
cm3 = confusion_matrix(y_test3, y_pred3)
acc3 = accuracy_score(y_test3, y_pred3)
print("Confusion Matrix:\n", cm3)
print("Accuracy:", acc3)
```

```
Confusion Matrix:
 [[68  0]
 [32  0]]
Accuracy: 0.68
```

7.3 Try 4: StandardScaler, random_state = 100, test_size = 0.25

[9]:
```python
# Try 4: StandardScaler, random_state = 100, test_size = 0.25

scaler4 = StandardScaler()
X_train4, X_test4, y_train4, y_test4 = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=100)
X_train4 = scaler4.fit_transform(X_train4)
X_test4 = scaler4.transform(X_test4)
model4 = LogisticRegression()
model4.fit(X_train4, y_train4)
y_pred4 = model4.predict(X_test4)
cm4 = confusion_matrix(y_test4, y_pred4)
acc4 = accuracy_score(y_test4, y_pred4)
print("Confusion Matrix:\n", cm4)
print("Accuracy:", acc4)
```

```
Confusion Matrix:
 [[62  3]
 [12 23]]
Accuracy: 0.85
```

8 Try 5: StandardScaler, random_state = 51, test_size = 0.25

[10]:
```
#Try 5: StandardScaler, random_state = 51, test_size = 0.25

scaler5 = StandardScaler()
X_train5, X_test5, y_train5, y_test5 = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=51)
X_train5 = scaler5.fit_transform(X_train5)
X_test5 = scaler5.transform(X_test5)
model5 = LogisticRegression()
model5.fit(X_train5, y_train5)
y_pred5 = model5.predict(X_test5)
cm5 = confusion_matrix(y_test5, y_pred5)
acc5 = accuracy_score(y_test5, y_pred5)
print("Confusion Matrix:\n", cm5)
print("Accuracy:", acc5)
```

```
Confusion Matrix:
 [[61  4]
 [ 9 26]]
Accuracy: 0.87
```

9 Try 6: Function to run experiment for new test case

[11]:
```
# Try 6: Function to run experiment for new test case

def run_logistic_test(X, y, scaler, test_size, random_state):
    X_train, X_test, y_train, y_test = train_test_split(X, y,␣
 ↪test_size=test_size, random_state=random_state)
    if scaler:
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)
    model = LogisticRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    acc = accuracy_score(y_test, y_pred)
    print(f"Accuracy with {scaler.__class__.__name__ if scaler else 'No␣
 ↪Scaling'}, RS={random_state}, TS={test_size}: {acc:.4f}")
    print("Confusion Matrix:\n", cm)
    return acc
```

[12]:
```
acc6 = run_logistic_test(X, y, StandardScaler(), 0.25, 21)
acc7 = run_logistic_test(X, y, StandardScaler(), 0.25, 41)
acc8 = run_logistic_test(X, y, StandardScaler(), 0.25, 11)
acc9 = run_logistic_test(X, y, StandardScaler(), 0.25, 2)
```

```
Accuracy with StandardScaler, RS=21, TS=0.25: 0.8600
Confusion Matrix:
 [[65  2]
 [12 21]]
Accuracy with StandardScaler, RS=41, TS=0.25: 0.8300
Confusion Matrix:
 [[59  4]
 [13 24]]
Accuracy with StandardScaler, RS=11, TS=0.25: 0.8300
Confusion Matrix:
 [[63  3]
 [14 20]]
Accuracy with StandardScaler, RS=2, TS=0.25: 0.8100
Confusion Matrix:
 [[56  6]
 [13 25]]
```

[13]:

```
# 8. Compare Accuracies
print("\n--- Accuracy Comparisons ---")
print("1. No Scaling, rs=0:       ", acc1)
print("2. StandardScaler, rs=0:   ", acc2)
print("3. Normalizer, rs=0:       ", acc3)
print("4. StandardScaler, rs=100: ", acc4)
print("5. StandardScaler, rs=51:  ", acc5)
print("6. StandardScaler, rs=21:   ", acc6)
print("7. StandardScaler, rs=41:   ", acc7)
print("8. StandardScaler, rs=11:   ", acc8)
print("9. StandardScaler, rs=2:    ", acc9)
```

```
--- Accuracy Comparisons ---
1. No Scaling, rs=0:        0.89
2. StandardScaler, rs=0:    0.89
3. Normalizer, rs=0:        0.68
4. StandardScaler, rs=100:  0.85
5. StandardScaler, rs=51:   0.87
6. StandardScaler, rs=21:    0.86
7. StandardScaler, rs=41:    0.83
8. StandardScaler, rs=11:    0.83
9. StandardScaler, rs=2:     0.81
```

9.1 Bias and Variance Check

- Let's check how well the model performs on training and test data.
- High training accuracy means low bias → model has learned training patterns well.
- If test accuracy is close to training accuracy, variance is low → good generalization.
- If there's a large gap, model might be overfitting (high variance).

- You don't need to check bias and variance for every model.

- Just test it on key configurations:
  - Best performing model
  - Worst or unstable model (optional)
- This helps identify if your model is underfitting (high bias) or overfitting (high variance).
- Balanced train/test accuracy means good generalization.

In our notebook, we observed that the best model using **`StandardScaler`** (RS=0, TestSize=0.2) had high training and test accuracy — confirming good balance and low variance.

10 Summary

We tested Logistic Regression using different preprocessing methods and random states.

- Best overall accuracy: 0.89
- Scaling data with StandardScaler provided better results than using raw or normalized data.
- Normalizer performed worst, possibly due to the nature of the features (Age, Salary).
- Bias and Variance scores were close, suggesting that the model generalizes well and is not overfitting.
- In future, try regularization and cross-validation to further improve performance.

---

11 Using Functions, loops and dictionaries

The section above was written for beginners to understand each step manually.

But in real-world or advanced use cases, we use **Functions, loops and dictionaries** to write cleaner, more scalable code. Below is an example that does the **same thing** as above but in a short and elegant way using a loop.

[15]:
```
#  Test with different configurations
results = []
scalers = {
    'No Scaling': None,
    'StandardScaler': StandardScaler(),
    'Normalizer': Normalizer()
}

random_states = [100, 51, 21, 41, 11, 2, 0]
test_sizes = [0.25, 0.2]

for scaler_name, scaler in scalers.items():
    for rs in random_states:
        for ts in test_sizes:
            # Split data
            X_train, X_test, y_train, y_test = train_test_split(X, y,␣
 ↪test_size=ts, random_state=rs)

            # Apply scaling
```

```
            if scaler:
                X_train = scaler.fit_transform(X_train)
                X_test = scaler.transform(X_test)

            # Train model
            classifier = LogisticRegression()
            classifier.fit(X_train, y_train)
            y_pred = classifier.predict(X_test)

            # Metrics
            acc = accuracy_score(y_test, y_pred)
            cm = confusion_matrix(y_test, y_pred)
            tn, fp, fn, tp = cm.ravel()

            results.append({
                'Scaler': scaler_name,
                'RandomState': rs,
                'TestSize': ts,
                'Accuracy': round(acc, 4),
                'TP': tp, 'TN': tn, 'FP': fp, 'FN': fn
            })
```

11.1 Convert Results to DataFrame

[16]:
```
# 4. Convert Results to DataFrame
results_df = pd.DataFrame(results)
results_df_sorted = results_df.sort_values(by='Accuracy', ascending=False).
 ↪reset_index(drop=True)
```

11.2 Display All Configurations and Best One

[17]:
```
# Display All Configurations and Best One
print("\nAll Configurations Tested:")
print(results_df_sorted)

best = results_df_sorted.iloc[0]
print("\n Best Accuracy:")
print(best)
```

```
All Configurations Tested:
           Scaler  RandomState  TestSize  Accuracy  TP  TN  FP  FN
0  StandardScaler            0      0.20    0.9250  17  57   1   5
1      No Scaling            0      0.20    0.9125  17  56   2   5
2      No Scaling            0      0.25    0.8900  24  65   3   8
3  StandardScaler            0      0.25    0.8900  24  65   3   8
4  StandardScaler           51      0.20    0.8875  20  51   1   8
5      No Scaling           51      0.20    0.8750  20  50   2   8
```

6	StandardScaler	51	0.25	0.8700	26	61	4	9
7	StandardScaler	21	0.20	0.8625	15	54	2	9
8	No Scaling	21	0.20	0.8625	15	54	2	9
9	No Scaling	51	0.25	0.8600	26	60	5	9
10	No Scaling	21	0.25	0.8600	21	65	2	12
11	StandardScaler	21	0.25	0.8600	21	65	2	12
12	StandardScaler	100	0.25	0.8500	23	62	3	12
13	No Scaling	100	0.25	0.8500	23	62	3	12
14	No Scaling	11	0.25	0.8300	20	63	3	14
15	StandardScaler	41	0.25	0.8300	24	59	4	13
16	StandardScaler	11	0.25	0.8300	20	63	3	14
17	No Scaling	100	0.20	0.8250	20	46	3	11
18	StandardScaler	100	0.20	0.8250	20	46	3	11
19	StandardScaler	11	0.20	0.8250	16	50	3	11
20	No Scaling	41	0.25	0.8200	24	58	5	13
21	No Scaling	11	0.20	0.8125	16	49	4	11
22	StandardScaler	2	0.20	0.8125	20	45	3	12
23	No Scaling	2	0.20	0.8125	20	45	3	12
24	StandardScaler	2	0.25	0.8100	25	56	6	13
25	No Scaling	2	0.25	0.8100	25	56	6	13
26	StandardScaler	41	0.20	0.7875	17	46	4	13
27	No Scaling	41	0.20	0.7750	17	45	5	13
28	Normalizer	0	0.20	0.7250	0	58	0	22
29	Normalizer	21	0.20	0.7000	0	56	0	24
30	Normalizer	0	0.25	0.6800	0	68	0	32
31	Normalizer	21	0.25	0.6700	0	67	0	33
32	Normalizer	11	0.20	0.6625	0	53	0	27
33	Normalizer	11	0.25	0.6600	0	66	0	34
34	Normalizer	100	0.25	0.6500	0	65	0	35
35	Normalizer	51	0.25	0.6500	0	65	0	35
36	Normalizer	51	0.20	0.6500	0	52	0	28
37	Normalizer	41	0.25	0.6300	0	63	0	37
38	Normalizer	41	0.20	0.6250	0	50	0	30
39	Normalizer	2	0.25	0.6200	0	62	0	38
40	Normalizer	100	0.20	0.6125	0	49	0	31
41	Normalizer	2	0.20	0.6000	0	48	0	32

```
 Best Accuracy:
Scaler         StandardScaler
RandomState                 0
TestSize                  0.2
Accuracy                0.925
TP                         17
TN                         57
FP                          1
FN                          5
Name: 0, dtype: object
```

[18]:
```python
# Optional: Save to CSV
results_df_sorted.to_csv("logistic_regression_experiments.csv", index=False)
```

12 Final Results Table Summary

After testing 42 combinations (scalers, test sizes, random states), here's what we found:

- **Best Accuracy**: **92.5%**
- Best achieved with:
- **Scaler**: `StandardScaler`
- **RandomState**: `0`
- **TestSize**: `0.2`

12.1 Detailed Best Result:

Metric	Value
Accuracy	92.5%
True Positives (TP)	17
True Negatives (TN)	57
False Positives (FP)	1
False Negatives (FN)	5

12.2 Interpretation:

- This combination had the **highest accuracy** and **lowest error values**.
- Very few wrong predictions:
  - Only **1 FP** and **5 FN**
- Shows that **StandardScaler preprocessing** works best for this dataset.
- Normalizer consistently underperformed due to scale-sensitive features (like Age, Salary).

13 Conclusion:

- Use `StandardScaler` when working with Logistic Regression and numeric features.
- RandomState and TestSize also affect performance — it's a good practice to tune them.
- This method helps you identify the **most reliable configuration** quickly and cleanly.