What is Model Evaluation?

Model evaluation is the process of using different evaluation metrics to understand a machine learning model's performance, as well as its strengths and weaknesses. Model evaluation is important to assess the efficacy of a model during initial research phases, and it also plays a role in model monitoring.

To understand if your model(s) is working well with new data, you can leverage a number of evaluation metrics.

Classification

The most popular metrics for measuring classification performance include accuracy, precision, confusion matrix, log-loss, and AUC (area under the ROC curve).

Accuracy measures how often the classifier makes the correct predictions, as it is the ratio between the number of correct predictions and the total number of predictions.

Precision measures the proportion of predicted Positives that are truly Positive. Precision is a good choice of evaluation metrics when you want to be very sure of your prediction. For example, if you are building a system to predict whether to decrease the credit limit on a particular account, you want to be very sure about the prediction or it may result in customer dissatisfaction.

A confusion matrix (or confusion table) shows a more detailed breakdown of correct and incorrect classifications for each class. Using a confusion matrix is useful when you want to understand the distinction between classes, particularly when the cost of misclassification might differ for the two classes, or you have a lot more test data on one class than the other. For example, the consequences of making a false positive or false negative in a cancer diagnosis are very different.

```
#Importing the libraries

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.metrics import accuracy_score, classification_report, confusion_matrix
```

```
#Loading the titanic dataset

path = ("https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/stuff/titanic.csv")

# Read the CSV file into a DataFrame
df = pd.read_csv(path)

# Display the DataFrame
df.head()
```

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833
2	1	3	Miss. Laina	female	26.0	0	0	7.9250

```
#Shape of the data

df.shape
```

```
(887, 8)
```

```
#Check for null values

df.isnull().sum()
```

```
Survived                  0
Pclass                    0
Name                      0
Sex                       0
Age                       0
Siblings/Spouses Aboard   0
Parents/Children Aboard   0
Fare                      0
dtype: int64
```

```
import seaborn as sns

# Load the Titanic dataset from seaborn
titanic_data = sns.load_dataset('titanic')

# Select the specified columns
selected_columns = ['age', 'fare', 'sex', 'sibsp', 'parch', 'pclass', 'embarked', 'survived'
titanic_subset = titanic_data[selected_columns]

# Rename the 'pclass' column to 'Pclass' and 'survived' to 'x2survived'
titanic_subset = titanic_subset.rename(columns={'pclass': 'Pclass', 'survived': 'x2survived'

# Display the subset of the dataset
print(titanic_subset)
```

```
      age     fare     sex  sibsp  parch  Pclass embarked  x2survived
0    22.0   7.2500    male      1      0       3        S           0
1    38.0  71.2833  female      1      0       1        C           1
2    26.0   7.9250  female      0      0       3        S           1
3    35.0  53.1000  female      1      0       1        S           1
4    35.0   8.0500    male      0      0       3        S           0
..    ...      ...     ...    ...    ...     ...      ...         ...
886  27.0  13.0000    male      0      0       2        S           0
887  19.0  30.0000  female      0      0       1        S           1
888   NaN  23.4500  female      1      2       3        S           0
889  26.0  30.0000    male      0      0       1        C           1
890  32.0   7.7500    male      0      0       3        Q           0

[891 rows x 8 columns]
```

```
#Dataframe

t_df = pd.DataFrame(titanic_subset)

t_df.head()
```

	age	fare	sex	sibsp	parch	Pclass	embarked	x2survived
0	22.0	7.2500	male	1	0	3	S	0
1	38.0	71.2833	female	1	0	1	C	1
2	26.0	7.9250	female	0	0	3	S	1
3	35.0	53.1000	female	1	0	1	S	1
4	35.0	8.0500	male	0	0	3	S	0

```
t_df.isnull().sum()
```

```
age          177
fare           0
```

```
sex             0
sibsp           0
parch           0
Pclass          0
embarked        2
x2survived      0
dtype: int64
```

```
#Feature Engeneering


#combine sibsp and parch to one column called company (this is passanger children and siblir

t_df.loc[(t_df.parch + t_df.sibsp < 1), 'Company'] = 0
t_df.loc[(t_df.parch + t_df.sibsp > 0), 'Company'] = 1

t_df.head()
```

	age	fare	sex	sibsp	parch	Pclass	embarked	x2survived	Company
0	22.0	7.2500	male	1	0	3	S	0	1.0
1	38.0	71.2833	female	1	0	1	C	1	1.0
2	26.0	7.9250	female	0	0	3	S	1	0.0
3	35.0	53.1000	female	1	0	1	S	1	1.0
4	35.0	8.0500	male	0	0	3	S	0	0.0

```
#Group fare and age into fare and age groups

# Define the age groups
bins = [0, 18, 35, 55, 75, float('inf')]
labels = [1, 2, 3, 4, 5]

# Create the 'Age_group' column based on the specified age groups
t_df['Age_group'] = pd.cut(titanic_subset['age'], bins=bins, labels=labels, right=False)

t_df.head()


#Same for fare

# Get the range of the 'fare' column
fare_range = t_df['fare'].min(), t_df['fare'].max()# Display the fare range
print("Fare Range:", fare_range)
```

```
Fare Range: (0.0, 512.3292)
```

```
# Define the fare groups
bins = [0, 100, 300,  float('inf')]
labels = [1, 2, 3]

# Create the 'Age_group' column based on the specified age groups
t_df['Fare_group'] = pd.cut(titanic_subset['fare'], bins=bins, labels=labels, right=False)

t_df.head()
```

	age	fare	sex	sibsp	parch	Pclass	embarked	x2survived	Company	Age_group
0	22.0	7.2500	male	1	0	3	S	0	1.0	2
1	38.0	71.2833	female	1	0	1	C	1	1.0	3
2	26.0	7.9250	female	0	0	3	S	1	0.0	2
3	35.0	53.1000	female	1	0	1	S	1	1.0	3
4	35.0	8.0500	male	0	0	3	S	0	0.0	3

```
t_df.dtypes
```

```
age            float64
fare           float64
sex             object
sibsp            int64
parch            int64
Pclass           int64
embarked        object
x2survived       int64
Company        float64
Age_group     category
Fare_group    category
dtype: object
```

```
# Convert 'sex' column to 0 and 1
t_df['sex'] = t_df['sex'].map({'male': 0, 'female': 1})
t_df.head()
```

	age	fare	sex	sibsp	parch	Pclass	embarked	x2survived	Company	Age_group	Fa
0	22.0	7.2500	0	1	0	3	S	0	1.0	2	
1	38.0	71.2833	1	1	0	1	C	1	1.0	3	
2	26.0	7.9250	1	0	0	3	S	1	0.0	2	
3	35.0	53.1000	1	1	0	1	S	1	1.0	3	
4	35.0	8.0500	0	0	0	3	S	0	0.0	3	

```
# Drop specified columns ('age', 'fare', 'sibsp', 'parch')
t_df = t_df.drop(['age', 'fare', 'sibsp', 'parch'], axis=1)

t_df.head()
```

	sex	Pclass	embarked	x2survived	Company	Age_group	Fare_group
0	0	3	S	0	1.0	2	1
1	1	1	C	1	1.0	3	1
2	1	3	S	1	0.0	2	1
3	1	1	S	1	1.0	3	1
4	0	3	S	0	0.0	3	1

```
# Convert 'Embarked' column to 0, 1 and 2
t_df['embarked'] = t_df['embarked'].map({'S': 2, 'C': 0, 'Q': 1})
t_df.head()
```

	sex	Pclass	embarked	x2survived	Company	Age_group	Fare_group
0	0	3	2.0	0	1.0	2	1
1	1	1	0.0	1	1.0	3	1
2	1	3	2.0	1	0.0	2	1
3	1	1	2.0	1	1.0	3	1
4	0	3	2.0	0	0.0	3	1

Training the models

```
#Reset index
dft = t_df.reset_index()
```

```
dft.shape
```

```
(891, 8)
```

```
dft.isnull().sum()
```

```
index          0
sex            0
```

```
Pclass          0
embarked        2
x2survived      0
Company         0
Age_group     177
Fare_group      0
dtype: int64
```

```
#Drop null
dft = dft.dropna()

dft.isnull().sum()
```

```
index         0
sex           0
Pclass        0
embarked      0
x2survived    0
Company       0
Age_group     0
Fare_group    0
dtype: int64
```

Split the data

```
#Split the data

x = dft.drop(['index','x2survived', 'embarked','Company'], axis = 1)
y = dft['x2survived']
```

```
y.head()
```

```
0    0
1    1
2    1
3    1
4    0
Name: x2survived, dtype: int64
```

```
x.head()
```

	sex	Pclass	Age_group	Fare_group
0	0	3	2	1
1	1	1	3	1
2	1	3	2	1
3	1	1	3	1
4	0	3	3	1

```
#Split the data into x_train and x_test

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=30)

 #Classifier (Classify as survived or not survived)

clf = LogisticRegression()

# Train the classifier on the training data
clf.fit(x_train, y_train)
```

▾ LogisticRegression
LogisticRegression()

```
# Make predictions on the testing data
y_pred = clf.predict(x_test)

# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
con_matrix = confusion_matrix(y_test, y_pred)
clas_report = classification_report(y_test, y_pred)


print("Accuracy:", accuracy)

print("Confussion Matrix:", con_matrix)

print("classification Report:", clas_report)
```

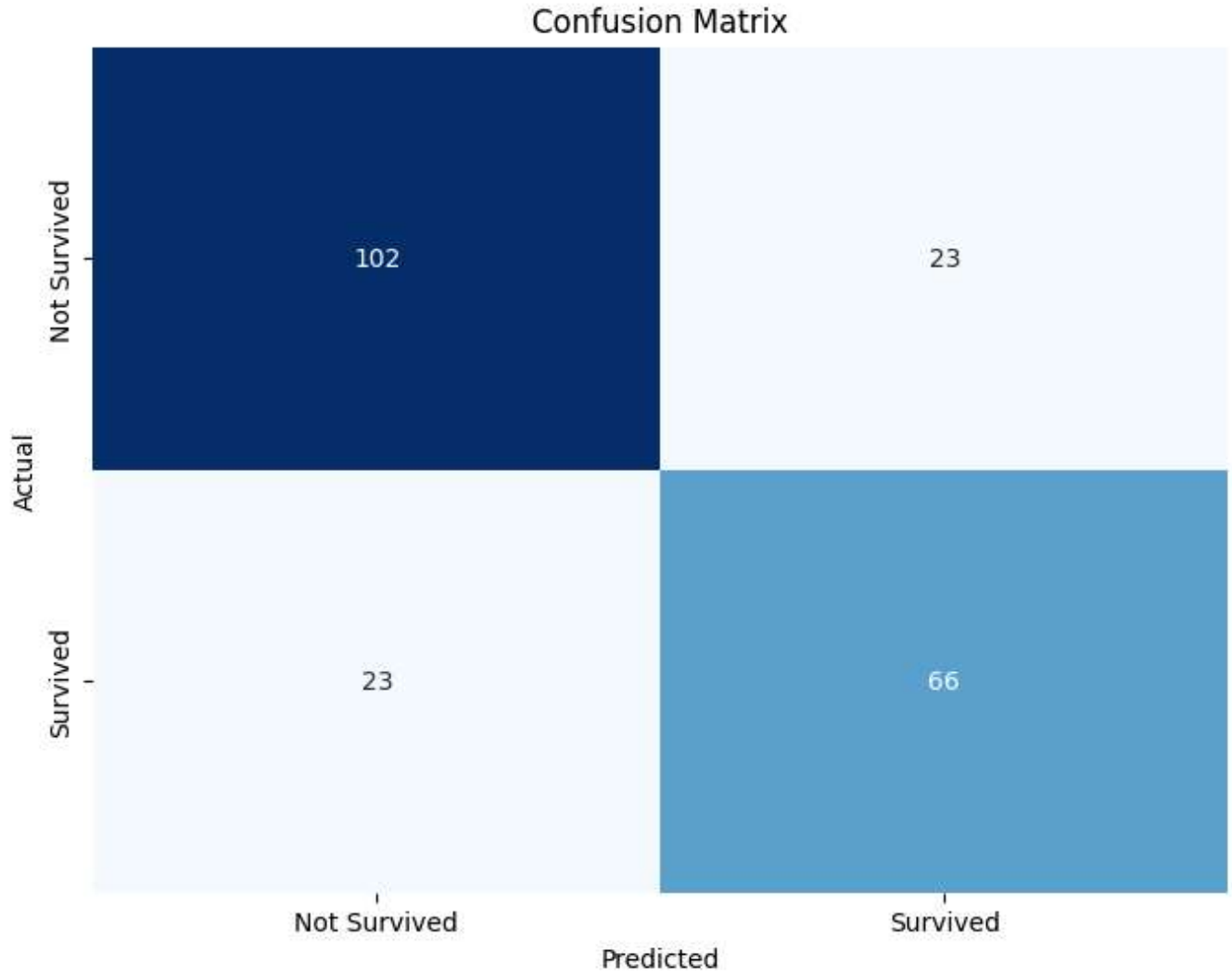
```
Accuracy: 0.7990654205607477
Confussion Matrix: [[114  21]
 [ 22  57]]
classification Report:               precision    recall  f1-score   support

           0       0.84      0.84      0.84       135
```

```
                1       0.73      0.72      0.73        79

         accuracy                           0.80       214
        macro avg       0.78      0.78      0.78       214
     weighted avg       0.80      0.80      0.80       214
```

```
# Plot the confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(con_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Not Survived', 'Survived'],
            yticklabels=['Not Survived', 'Survived'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



The confusion matrix shows the counts of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions. In this case:

True Positive (TP): 102

True Negative (TN): 66

False Positive (FP): 23

False Negative (FN): 23

Precision:

Precision for class 0 (not survived): 0.82

Precision for class 1 (survived): 0.74

Precision is the ratio of correctly predicted positive observations to the total predicted positives. A high precision indicates a low false positive rate.

Recall (Sensitivity or True Positive Rate):

Recall for class 0: 0.82

Recall for class 1: 0.74

Recall is the ratio of correctly predicted positive observations to the total actual positives. A high recall indicates a low false negative rate.

F1-Score:

F1-score for class 0: 0.82

F1-score for class 1: 0.74

The F1-score is the weighted average of precision and recall. It is a balance between precision and recall, providing a single metric that considers both false positives and false negatives.

Support:

Support for class 0: 125

Support for class 1: 89

Support is the number of actual occurrences of the class in the specified dataset.

Accuracy:

Overall accuracy: 0.79

Accuracy is the ratio of correctly predicted observations to the total observations. It provides a general measure of model performance.

Macro Avg:

Macro-averaged precision, recall, and F1-score consider each class equally, providing an unweighted average across classes.

Weighted Avg:

Weighted-averaged precision, recall, and F1-score consider each class with weight proportional to its support (number of occurrences), providing an average that considers the imbalance in class distribution.

In summary, this classification report offers a comprehensive view of how well your logistic regression model is performing for each class and overall. It's a valuable tool for understanding the strengths and weaknesses of your model in different aspects of classification.

Model Evaluation:

Model evaluation refers to the process of assessing how well a machine learning model performs on a specific task. The goal is to understand how the model generalizes to new, unseen data and whether it meets the desired criteria or performance metrics. Common evaluation metrics depend on the nature of the problem (classification, regression, clustering, etc.) and may include:

Classification Metrics:

Accuracy: The proportion of correctly classified instances.

Precision, Recall, F1 Score: Metrics that provide insights into the trade-off between false positives and false negatives.

Area Under the Receiver Operating Characteristic (ROC) Curve (AUC-ROC): Useful for binary classification problems.

Regression Metrics:

Mean Squared Error (MSE): Measures the average squared difference between predicted and actual values.

Mean Absolute Error (MAE): Measures the average absolute difference between predicted and actual values.

R-squared: Indicates the proportion of variance in the dependent variable explained by the model.

Clustering Metrics:

Silhouette Score: Measures how similar an object is to its own cluster compared to other clusters.

Davies-Bouldin Index: Measures the average similarity between each cluster and its most similar cluster.

Other Metrics:

Cross-Validation: Techniques like k-fold cross-validation to assess model performance on different subsets of the data.

Confusion Matrix: Provides a detailed breakdown of correct and incorrect predictions.

Model Refinement:

Model refinement involves iteratively improving the model based on the insights gained during the evaluation phase. It aims to enhance the model's predictive performance, generalization, and robustness. Key steps in model refinement include:

Hyperparameter Tuning:

Adjusting hyperparameters (e.g., learning rate, regularization strength) to find the optimal configuration for the model.

Feature Engineering:

Creating new features or transforming existing ones to improve the model's ability to capture patterns in the data.

Data Cleaning and Preprocessing:

Handling missing data, outliers, or skewed distributions.

Scaling or normalizing features to ensure consistent impact on the model.

Ensemble Methods:

Exploring ensemble techniques such as bagging (e.g., Random Forest) or boosting (e.g., Gradient Boosting) to combine multiple models for better performance.

Model Selection:

Trying different algorithms or architectures to identify the most suitable model for the task.

Regularization:

Applying regularization techniques to prevent overfitting and improve model generalization.

Error Analysis:

Analyzing model errors to identify patterns or areas where the model consistently struggles, and taking corrective actions.

Validation and Cross-Validation:

Continuously validating the model on different datasets to ensure its performance generalizes well.

The process of model evaluation and refinement is often iterative, with continuous feedback and adjustments to enhance the model's effectiveness in real-world scenarios.