Project ARI3205 Interpretable AI for Deep Learning Models *(Part 3.1)*

Name: Sean David Muscat

ID No: 0172004L

Importing Necessary Libraries

In [1]:
```
# Check and install required libraries from the libraries.json file
import json

# Read the libraries from the text file
with open('../Libraries/Part3.1_Lib.json', 'r') as file:
    libraries = json.load(file)

# ANSI escape codes for colored output
GREEN = "\033[92m"  # Green text
RED = "\033[91m"    # Red text
RESET = "\033[0m"   # Reset to default color

# Function to check and install libraries
def check_and_install_libraries(libraries):
    for lib, import_name in libraries.items():
        try:
            # Attempt to import the library
            __import__(import_name)
            print(f"[{GREEN}✓{RESET}] Library '{lib}' is already installed.")
        except ImportError:
            # If import fails, try to install the library
            print(f"[{RED}✗{RESET}] Library '{lib}' is not installed. Installin
            %pip install {lib}

# Execute the function to check and install libraries
check_and_install_libraries(libraries)

# Import necessary libraries for data analysis and modeling
import warnings
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.formula.api as smf
# Alibi imports for the MNIST example
import tensorflow as tf
tf.get_logger().setLevel(40)  # suppress deprecation messages
tf.compat.v1.disable_v2_behavior()  # disable TF2 behaviour as Alibi code still
tf.compat.v1.reset_default_graph()
tf.keras.backend.clear_session()
from tensorflow.keras.layers import Conv2D, Dropout, Flatten, MaxPooling2D, UpSa
from tensorflow.keras.models import Model, Sequential, load_model
```

```
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import StandardScaler,  OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.inspection import PartialDependenceDisplay, permutation_importance
from alibi.explainers import ALE, plot_ale
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
import statsmodels.api as sm
from alibi.explainers import Counterfactual
from time import time                                                             # St









import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
import os
from time import time
from alibi.explainers import CounterfactualProto

# Suppress specific warnings
warnings.filterwarnings("ignore", message="X does not have valid feature names")
warnings.filterwarnings("ignore", category=RuntimeWarning)
warnings.filterwarnings("ignore", category=UserWarning)
```

```
[✓] Library 'tensorflow' is already installed.
[✓] Library 'scikit-learn' is already installed.
[✓] Library 'matplotlib' is already installed.
[✓] Library 'seaborn' is already installed.
[✓] Library 'pandas' is already installed.
[✓] Library 'numpy' is already installed.
[✓] Library 'scipy' is already installed.
C:\Users\Sean Muscat\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_
qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\tqdm\auto.py:21:
TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See http
s://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
[✓] Library 'alibi' is already installed.
[✓] Library 'statsmodels' is already installed.
[✓] Library 'time' is already installed.
[✓] Library 'os' is already installed.
```

In [2]:

```
# Define the filenames
train_filename = '../Datasets/Titanic/train.csv'
test_filename = '../Datasets/Titanic/test.csv'
gender_submission_filename = '../Datasets/Titanic/gender_submission.csv'

# Load the datasets
try:
    train_data = pd.read_csv(train_filename)
    test_data = pd.read_csv(test_filename)
    gender_submission_data = pd.read_csv(gender_submission_filename)
    print(f"'{train_filename}' dataset loaded successfully.")
```

```
    print(f"'{test_filename}' dataset loaded successfully.")
    print(f"'{gender_submission_filename}' dataset loaded successfully.")
except FileNotFoundError as e:
    print(f"Error: {e.filename} was not found. Please ensure it is in the correc
    exit()
except pd.errors.EmptyDataError as e:
    print(f"Error: {e.filename} is empty.")
    exit()
except pd.errors.ParserError as e:
    print(f"Error: There was a problem parsing {e.filename}. Please check the fi
    exit()

# Dataset insights
print("\nTrain Dataset Overview:")
print(train_data.info())
print("\nTrain Dataset Statistical Summary:")
print(train_data.describe())

print("\nTest Dataset Overview:")
print(test_data.info())
print("\nTest Dataset Statistical Summary:")
print(test_data.describe())

print("\nGender Submission Dataset Overview:")
print(gender_submission_data.info())
```

```
'../Datasets/Titanic/train.csv' dataset loaded successfully.
'../Datasets/Titanic/test.csv' dataset loaded successfully.
'../Datasets/Titanic/gender_submission.csv' dataset loaded successfully.

Train Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None

Train Dataset Statistical Summary:
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000   
mean    446.000000    0.383838    2.308642   29.699118    0.523008   
std     257.353842    0.486592    0.836071   14.526497    1.102743   
min       1.000000    0.000000    1.000000    0.420000    0.000000   
25%     223.500000    0.000000    2.000000   20.125000    0.000000   
50%     446.000000    0.000000    3.000000   28.000000    0.000000   
75%     668.500000    1.000000    3.000000   38.000000    1.000000   
max     891.000000    1.000000    3.000000   80.000000    8.000000   

            Parch        Fare  
count  891.000000  891.000000  
mean     0.381594   32.204208  
std      0.806057   49.693429  
min      0.000000    0.000000  
25%      0.000000    7.910400  
50%      0.000000   14.454200  
75%      0.000000   31.000000  
max      6.000000  512.329200  

Test Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
```

```
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.1+ KB
None

Test Dataset Statistical Summary:
       PassengerId      Pclass         Age       SibSp       Parch        Fare
count   418.000000  418.000000  332.000000  418.000000  418.000000  417.000000
mean   1100.500000    2.265550   30.272590    0.447368    0.392344   35.627188
std     120.810458    0.841838   14.181209    0.896760    0.981429   55.907576
min     892.000000    1.000000    0.170000    0.000000    0.000000    0.000000
25%     996.250000    1.000000   21.000000    0.000000    0.000000    7.895800
50%    1100.500000    3.000000   27.000000    0.000000    0.000000   14.454200
75%    1204.750000    3.000000   39.000000    1.000000    0.000000   31.500000
max    1309.000000    3.000000   76.000000    8.000000    9.000000  512.329200

Gender Submission Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Survived     418 non-null    int64
dtypes: int64(2)
memory usage: 6.7 KB
None
```

Feed-Forward Neural Network

In [3]:

```python
# Load the Titanic dataset
train_data = pd.read_csv('../Datasets/Titanic/train.csv')

# Preprocessing
# Separate features and target
y = train_data['Survived']  # Target
X = train_data.drop(columns=['Survived', 'PassengerId', 'Name', 'Ticket', 'Cabin

# Handle categorical variables with one-hot encoding
categorical_features = ['Sex', 'Embarked']
one_hot_encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
categorical_encoded = one_hot_encoder.fit_transform(X[categorical_features])
categorical_encoded_df = pd.DataFrame(categorical_encoded, columns=one_hot_encod

# Drop original categorical columns and append the encoded columns
X = X.drop(columns=categorical_features)
X = pd.concat([X.reset_index(drop=True), categorical_encoded_df.reset_index(drop

# Handle missing values with mean imputation
imputer = SimpleImputer(strategy='mean')
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

# Standardize the features
scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X_imputed), columns=X.columns)
```

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
print("Training data shape:", X_train.shape)
print("Test data shape:", X_test.shape)
```

```
Training data shape: (712, 11)
Test data shape: (179, 11)
```

In [ ]:
```
# Build the feed-forward neural network
model = Sequential([
    Input(shape=(X_train.shape[1],)),  # Define input shape explicitly
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')  # Output layer for binary classification
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', m

# Train the model
history = model.fit(X_train, y_train, validation_split=0.2, epochs=50, batch_siz

# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=1)
print(f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}")
```

Surrogate Model - MLPClassifier

In [5]:
```
# Train a surrogate model (MLPClassifier)
surrogate_model = MLPClassifier(hidden_layer_sizes=(32,), activation='logistic',
print('Accuracy (MLPClassifier): ' + str(surrogate_model.score(X_train, y_train)
```

```
Accuracy (MLPClassifier): 0.800561797752809
```

Part 3.1

Set up Counterfactuals

We begin by predicting labels on the test set and identifying which samples the model misclassifies. For each misclassified passenger, we record their scaled features and define a prediction function that converts our model's single sigmoid output into a two-column probability array: [p(died), p(survived)]. Alibi's counterfactual explainer then searches within specified min/max bounds for a new set of feature values that shifts the model's predicted outcome (for example, from "died" to "survived"). Finally, we compare these counterfactual features with the originals to see how small changes in attributes like Age or Fare can flip the prediction.

In [6]:
```
# 1. Make predictions on the test set
y_pred_probs = model.predict(X_test)
y_pred = (y_pred_probs > 0.5).astype(int).flatten()

# 2. Identify misclassified samples
incorrect_indices = np.where(y_pred != y_test.values)[0]
print(f"Number of incorrectly predicted samples: {len(incorrect_indices)}")
```

```
# Make sure we have at least 2 misclassified samples
if len(incorrect_indices) < 2:
    print("Fewer than 2 misclassified samples found. Cannot generate two counter
else:
    #####################################
    # LOOP OVER THE FIRST 2 MISCLASSIFIED
    #####################################
    for i in range(2):  # generate counterfactual for the first two misclassifie
        print(f"\n*** COUNTERFACTUAL #{i+1} ***")

        # 3. Select one misclassified example
        sample_idx = incorrect_indices[i]  # pick the i-th misclassified sample
        x_test_sample = X_test.iloc[[sample_idx]].values
        actual_label = y_test.values[sample_idx]
        print(f"Sample index: {sample_idx}, Actual label: {actual_label}, Predic
        print("\nSample features (scaled):")
        display(X_test.iloc[[sample_idx]])

        # 4. Define a new predict_fn that outputs [p(died), p(survived)] for eac
        def predict_fn(x: np.ndarray) -> np.ndarray:
            if x.ndim == 1:
                x = x.reshape(1, -1)
            p_survived = model.predict(x).flatten()
            p_died = 1.0 - p_survived
            return np.vstack([p_died, p_survived]).T

        # 5. Determine feature_range from training data
        lower_bounds = X_train.min(axis=0).values
        upper_bounds = X_train.max(axis=0).values
        feature_range = (lower_bounds, upper_bounds)

        # 6. Decide on target_proba to 'flip' the original label
        desired_proba = 0.8 if actual_label == 0 else 0.2

        # 7. Instantiate the Counterfactual explainer
        cf_explainer = Counterfactual(
            predict_fn=predict_fn,
            shape=(1, X_train.shape[1]),
            target_proba=desired_proba,
            max_iter=1000,
            feature_range=feature_range,
            lam_init=1e-1,
            max_lam_steps=10,
            learning_rate_init=1e-2
        )

        # 8. Generate a counterfactual explanation
        explanation = cf_explainer.explain(x_test_sample)

        # 9. Print results
        print("\n--- Counterfactual Explanation ---")
        print("Original 2-column probability:", predict_fn(x_test_sample))
        if explanation.cf is not None:
            cf_sample = explanation.cf['X']  # shape => (1, n_features)
            print("\nCounterfactual feature values (scaled):")
            display(cf_sample)

            print("Counterfactual 2-column probability:", predict_fn(cf_sample))
```

```
            # Show the numerical difference
            changes = cf_sample[0] - x_test_sample[0]
            print("\nDifference between CF and original sample:")
            for col, diff in zip(X_test.columns, changes):
                print(f"{col}: {diff:.3f}")
        else:
            print("No counterfactual found within the specified parameters.")
```

```
Number of incorrectly predicted samples: 64

*** COUNTERFACTUAL #1 ***
Sample index: 0, Actual label: 1, Predicted: 0

Sample features (scaled):
```

	Pclass	Age	SibSp	Parch	Fare	Sex_female	Sex_male	Embarked_C	En
709	0.827377	0.0	0.432793	0.76763	-0.341452	-0.737695	0.737695	2.074505	

```
--- Counterfactual Explanation ---
Original 2-column probability: [[0.64625597 0.353744  ]]

Counterfactual feature values (scaled):
array([[ 0.82737726, -1.394982  ,  1.6170563 ,  2.1170812 , -0.64842165,
        -0.73769516,  0.73769516,  2.074505  ,  0.58823454, -1.6147097 ,
         1.3070657 ]], dtype=float32)
Counterfactual 2-column probability: [[0.75010574 0.24989426]]

Difference between CF and original sample:
Pclass: 0.000
Age: -1.395
SibSp: 1.184
Parch: 1.349
Fare: -0.307
Sex_female: -0.000
Sex_male: 0.000
Embarked_C: -0.000
Embarked_Q: 0.896
Embarked_S: -0.000
Embarked_nan: 1.354

*** COUNTERFACTUAL #2 ***
Sample index: 4, Actual label: 1, Predicted: 0

Sample features (scaled):
```

	Pclass	Age	SibSp	Parch	Fare	Sex_female	Sex_male	Embarked_
39	0.827377	-1.208115	0.432793	-0.473674	-0.422074	1.355574	-1.355574	2.0745

```
--- Counterfactual Explanation ---
Original 2-column probability: [[0.54528    0.45472002]]

Counterfactual feature values (scaled):
array([[ 0.82737726, -1.2068506 ,  0.43296716, -0.4736736 , -0.422202  ,
         1.3555735 , -1.3555735 ,  2.074505  ,  0.4825647 ,  0.6193064 ,
         2.7697735 ]], dtype=float32)
```

```
Counterfactual 2-column probability: [[0.79656625 0.20343377]]

Difference between CF and original sample:
Pclass: 0.000
Age: 0.001
SibSp: 0.000
Parch: -0.000
Fare: -0.000
Sex_female: -0.000
Sex_male: 0.000
Embarked_C: -0.000
Embarked_Q: 0.790
Embarked_S: 2.234
Embarked_nan: 2.817
```

Output Explanation:

Counterfactual #1 (Sample index: 0, Actual label: 1, Predicted: 0) In this example, the model originally assigns a probability of approximately 64.63% to class 0 (and 35.37% to class 1). Several features are then adjusted—most notably, Age decreases substantially (by -1.395 in scaled units), while SibSp (number of siblings/spouses) and Parch (number of parents/children) both increase. Additionally, there are changes in Embarked_Q and Embarked_nan. After these modifications, the model's probability for class 0 rises to about 75.01%, moving further away from predicting the correct label of 1. This indicates that these specific adjustments to the features cause the model to become even more confident in the incorrect prediction. It suggests that age and the number of family members travelling (as encoded in SibSp and Parch) may be influential in pushing the prediction toward non-survival under this particular counterfactual setting.

Counterfactual #2 (Sample index: 4, Actual label: 1, Predicted: 0) Here, the original probabilities are roughly 54.53% for class 0 versus 45.47% for class 1—still an incorrect prediction, though the model is slightly less certain compared with Counterfactual #1. The counterfactual modifies the feature representation of passenger embarkation, with notable jumps in Embarked_Q, Embarked_S, and Embarked_nan. Despite these changes, the probability for class 0 increases further to approximately 79.66%. This outcome indicates that shifts in certain embarkation features, under the current model, do not bring the prediction closer to the correct label for this sample but instead reinforce the model's belief that the passenger did not survive.

3.1 b

Counterfactual explanations are vital because they tell us how to alter specific features in a model's input so that its prediction changes to a desired outcome. When we see how even small changes in passenger attributes (for instance, lowering their age or increasing their fare) flip the prediction from "died" to "survived," we gain insights into what the model deems crucial for its decision.

In debugging models, counterfactuals help us pinpoint problematic behaviours and potential biases. If the counterfactual requires unrealistic feature shifts—such as setting the fare far above any real-world range—then our model may be over-reliant on that feature, or it might not generalise well. We can use this knowledge to refine data preprocessing or adjust hyperparameters, ensuring our model bases decisions on more sensible factors.

Counterfactuals can direct real-world interventions from a decision-making perspective. For instance, if a passenger's survival probability increases significantly with a slight increase in fare, this indicates that socioeconomic position (as measured by fare) has a significant impact on the model. Managers, legislators, or end users can then evaluate the fairness or realism of these elements. Counterfactuals essentially assist stakeholders in understanding how to modify inputs in a meaningful way, increasing the transparency of model outputs and enabling more informed choices in practical situations.