Lab 03 - Applied Machine Learning

Bui Minh Thuan - 104486358

In this lab, we focus on applying machine learning techniques, specifically logistic regression, to predict passenger survival based on the widely known Titanic dataset. This dataset contains demographic and travel-related information for passengers aboard the Titanic, offering a structured platform to explore classification problems.

1. Select independent and dependent variables. Split the data into training and testing sets, and then create a logistic regression classifier to fit the model.

The Titanic dataset documents information about passengers aboard the Titanic, commonly used for survival prediction. As the aim of this task is to predict whether a person has survived or not, column "Survived" will be the dependent variable. For independent variables, I choose the following columns:

- Pclass
- Sex
- Age
- SibSp
- Parch
- Fare
- Embarked

I chose these columns as independent variables because they are likely to influence the survival status "Survived" based on logical reasoning and domain knowledge. After choosing the features and target, I start to split the dataset into training and testing, then train a simple Logistic Regression model with the "Scikit-learn" library. Below is the code for this task:

Identify independent variables

```
X = df[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']]
y = df['Survived']
```

✓ 0.0s

Split dataset into training and test set

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

✓ 0.0s

Model training

Create a logistic regression classifier and fit the model

Generate + Code + Markdown

```
# Train the logistic regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

✓ 0.0s

2. Utilize your model to make predictions on the testing data, calculate evaluation metrics such as accuracy and recall, and print the results

After having a trained model, we can validate whether it works well or not. “Scikit-learn” provides many effective built-in functions that we can use to evaluate the efficiency of the model. Here, with the given dataset and settings (80/20 for training and testing, 1000 as max-iter), we receive an accuracy of about 83%.

Utilize your model to make predictions on the testing data, calculate evaluation metrics such as accuracy and recall, and print the results.

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

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

[107] ✓ 0.0s Python

```
Accuracy: 0.8379888268156425

Classification Report:
               precision    recall  f1-score   support

           0       0.86      0.87      0.87       107
           1       0.80      0.79      0.80        72

    accuracy                           0.84       179
   macro avg       0.83      0.83      0.83       179
weighted avg       0.84      0.84      0.84       179
```

3. Display the theta parameter values

Theta parameters are all model attributes, meaning we can get it directly from the model itself.

Display the theta parameter values.

```python
print(f"Intercept: {model.intercept_}")
print(f"Coeficient: {model.coef_}")
```

✓ 0.0s

```
Intercept: [4.79799044]
Coeficient: [[-9.86723918e-01 -2.50140567e+00 -3.64974779e-02 -3.09892307e-01
  -5.35845281e-02  1.68774488e-03 -1.97888084e-01]]
```

4. Create a DataFrame with 3 records (for 3 persons), use your model to make predictions, and print the predicted results using text descriptions such as 'survived' and 'not survived'.

In the 4th step, it is required to create a sample record with “Pandas” to make predictions and print the predicted outcome. As the required record is relatively small (only 3 records), I choose to hard-code them instead of randomly creating or similar methods.

Create a DataFrame with 3 records (for 3 persons), use your model to make predic
as 'survived' and 'not survived'

```python
# Create a DataFrame with 3 records
new_data = pd.DataFrame({
    'Pclass': [1, 3, 2],
    'Sex': [0, 1, 0],
    'Age': [29, 24, 36],
    'SibSp': [0, 1, 1],
    'Parch': [0, 0, 2],
    'Fare': [100, 7.25, 26],
    'Embarked': [0, 2, 1]
})

# Make predictions
predictions = model.predict(new_data)

# Map predictions to text descriptions
predicted_results = ['survived' if pred == 1 else 'not survived' for pred in predictions]

# Print the results
for i in range(len(predicted_results)):
    print(f"Person {i+1}: {predicted_results[i]}")
```

[109] ✓ 0.0s

```
Person 1: survived
Person 2: not survived
Person 3: survived
```

5. Alter the training/testing split fraction and the maximum iteration of the logistic regression model, observe and print the different outcomes.

For the final task, it's like we are playing around with the model by setting different configurations. We can see that when I change the split fraction and reduce the max-iter, the model now only has an accuracy around 76%.

Alter the training/testing split fraction and the maximum iteration of the l

```python
# Split the dataset with different rates
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Train the logistic regression model with different max iter
model = LogisticRegression(max_iter=300)
model.fit(X_train, y_train)

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

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

[110] ✓ 0.0s

```
Accuracy: 0.7653631284916201

Classification Report:
               precision    recall  f1-score   support

           0       0.77      0.83      0.80        99
           1       0.76      0.69      0.72        80

    accuracy                           0.77       179
   macro avg       0.77      0.76      0.76       179
weighted avg       0.77      0.77      0.76       179
```