COS30082 - Applied Machine Learning

Week 3 - Lab - Logistic Regression

Huynh Trung Chien - 104848770

1. Select the features you intend to use as independent variables and identify your target (dependent) variable. Split the data into training and testing sets. Create a logistic regression classifier and fit the model.

Before showing the features that I selected for my model, I will first demonstrate some of the data preprocessing steps that I have done with the Titanic Survival dataset.

a. Convert "Sex" column (categorical) to numerical.

```
# Convert 'sex' to numerical
label_encoder = LabelEncoder()
df['Sex'] = label_encoder.fit_transform(df['Sex'])
```

b. Handle missing values for "Age" column using median.

```
# Handle missing values for Age using median
df['Age'] = df['Age'].fillna(df['Age'].median())
```

c. Handle missing values for "Embarked" column.

For this column, I filled the missing values with "S".

```
# Handle missing values for Embarked
df['Embarked'] = df['Embarked'].fillna('S') # Filling missing values in Embarked with S
```

Then, I converted the "Embarked" column to numerical and convert True/False values to 0/1.

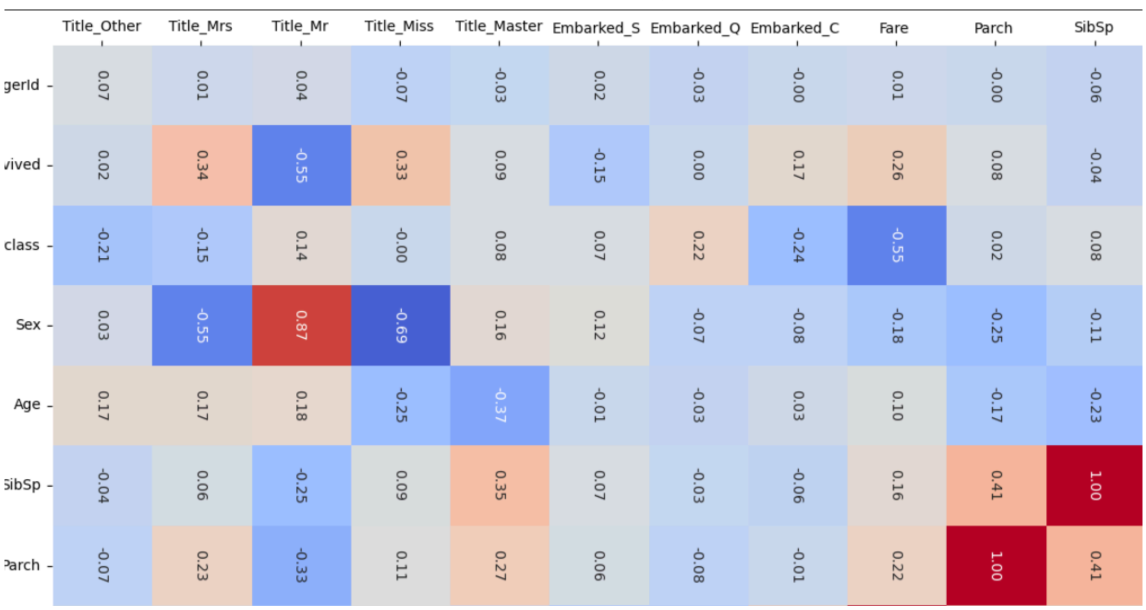
```
# Convert df['Embarked'] to numerical using get_dummies()
df = pd.get_dummies(df, columns=['Embarked'])

# Convert True/False values to 0/1
df['Embarked_C'] = df['Embarked_C'].astype(int)
df['Embarked_Q'] = df['Embarked_Q'].astype(int)
df['Embarked_S'] = df['Embarked_S'].astype(int)
```

d. Extract the “Name” column into “Title” columns.

Below are the steps that I used for the extraction.

```python
# extract Title from Name
df["Title"] = df["Name"].str.extract(r",\s*([^\.]+)\.")  # everything between ',' and '.'

# Count frequencies of each title
title_counts = df["Title"].value_counts()
print(title_counts)

# Get the list of rare titles (which appears under 10 times)
rare_titles = title_counts[title_counts < 10].index

# Replace rare titles with 'Other'
df["Title"] = df["Title"].replace(rare_titles, "Other")
print(df["Title"].nunique())

# One-hot encode the title
df = pd.get_dummies(df, columns=["Title"], prefix="Title")

# Convert True/False values to 0/1
df['Title_Mrs'] = df['Title_Mrs'].astype(int)
df['Title_Mr'] = df['Title_Mr'].astype(int)
df['Title_Master'] = df['Title_Master'].astype(int)
df['Title_Other'] = df['Title_Other'].astype(int)
df['Title_Miss'] = df['Title_Miss'].astype(int)
```

e. Summation of “SibSp” and “Parch” as a “FamilySize” column.



From the correlation matrix above, we see that the ***“SibSp”*** and ***“Parch”*** are quite correlated and they are all related to members in a family, so we will handle them by summing them and create a new feature named ***“FamilySize”***.

```
# Handle the correlation

df['FamilySize'] = df['SibSp'] + df['Parch']
```

The features that I dropped when training includes: ***"PassengerId"***, ***"Name"***, ***"SibSp"***, ***"Parch"***, ***"Cabin"***, ***"Ticket"***, ***"Title_Mr"***, and the reasons why I did not use them are described below:

- For ***"Cabin"***, there are too many missing values, so it is quite hard to fill in those missing values. Therefore, I won't include this column in training.

```
# Check the missing values in the dataset
check_missing(df)

=== Missing Values Information ===
Total Missing Values:  866
Missing values for each features:
 PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

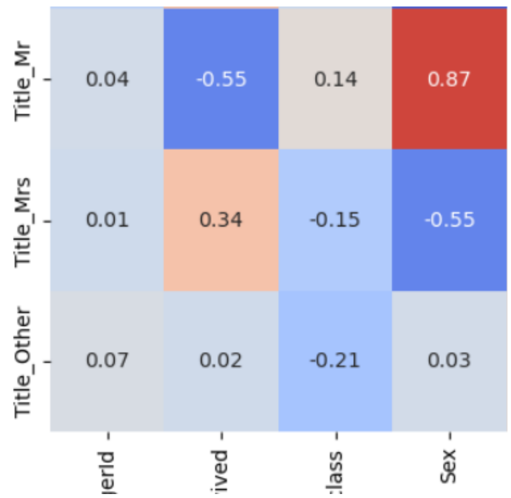
- For ***"Name"***, because I have extracted ***"Name"*** into ***"Title"*** columns, I will drop ***"Name"*** when training.

- For ***"Ticket"***, there are many unique values of tickets ***(681)***, so it does not represent any patterns, which is not a valuable feature for training. Therefore, I will drop it when training.

```
print(clr.G+"Number of unique values for Ticket -"+clr.E, df['Ticket'].nunique())

Number of unique values for Ticket - 681
```

- For ***"SibSp"*** and ***"Parch"***, because they are highly correlated and I have already done the feature engineering for them, I will drop them when training.

- For ***"Title_Mr"***, I dropped it because of its high correlation with ***"Sex"***.



- For ***"PassengerId"***, each passenger has a specific and unique id, so it is not valuable for training.

After defining the valuable features, I defined the ***X (inputs)*** and ***y (labels)*** with ***"Survived"*** column for ***y*** and other features for ***X***.

```
# Define X and y

X = df_new.drop(['Survived'], axis=1)
y = df_new['Survived']
```

Then, I splited the train and test sets using ***train_test_split()*** with ***test_size = 0.2***, ***random_state = 42***, and ***stratify = y***.

```
# Split the training and test set using train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size = 0.2,
    random_state = 42,
    stratify=y
)
```

After that, I used ***StandardScaler()*** for Normalization step.

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Finally, I created a baseline logistic regression model with the ***X_train***, ***y_train*** above and a ***max_iter = 100***.

```
# Baseline Logistic Regression

log_reg = LogisticRegression(max_iter=100)
log_reg.fit(X_train_scaled, y_train)

y_pred_test = log_reg.predict(X_test_scaled)
y_pred_train = log_reg.predict(X_train_scaled)

print(clr.G+"Baseline Logistic Regression \n\n"+clr.E)
print("Train Accuracy:", accuracy_score(y_train, y_pred_train), "\n\n")
print("Test Accuracy:", accuracy_score(y_test, y_pred_test))
print(classification_report(y_test, y_pred_test))
```

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

Below are the results of the evaluation metrics when I used my baseline model to make predictions on the train and test data. I got the accuracy using ***accuracy_score()*** and the metrics using ***classification_report()***.

```
Baseline Logistic Regression


Train Accuracy: 0.827247191011236


Test Accuracy: 0.8379888268156425
              precision    recall  f1-score   support

           0       0.85      0.89      0.87       110
           1       0.81      0.75      0.78        69

    accuracy                           0.84       179
   macro avg       0.83      0.82      0.83       179
weighted avg       0.84      0.84      0.84       179
```

3. Display the theta parameter values.

Below are the theta parameter values of my baseline model:

```
[26]: # Display Theta Parameters

theta = np.concatenate(([log_reg.intercept_[0]], log_reg.coef_[0]))
theta_df = pd.DataFrame({
    "Feature": ["Intercept"] + list(X.columns),
    "Theta": theta
})
print(theta_df)

        Feature     Theta
0     Intercept -0.680713
1        Pclass -0.864882
2           Sex -1.251583
3           Age -0.361463
4          Fare  0.162689
5    Embarked_C  0.059281
6    Embarked_Q  0.099575
7    Embarked_S -0.112486
8  Title_Master  0.601980
9    Title_Miss  0.088490
10    Title_Mrs  0.352034
11  Title_Other  0.078115
12   FamilySize -0.593529
```

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'.

Below are the results when I tested with the dummy list of passengers.

```
[27]: # Predictions on New Passengers

test_data = pd.DataFrame({
    "Pclass": [1, 3, 2],
    "Sex": [0, 1, 1],
    "Age": [25, 40, 18],
    "Fare": [80.00, 7.75, 13.50],
    "Embarked_C": [1, 0, 0],
    "Embarked_Q": [0, 1, 0],
    "Embarked_S": [0, 0, 1],
    "Title_Master":   [0, 1, 1],
    "Title_Miss": [1, 0, 0],
    "Title_Mrs":  [0, 0, 0],
    "Title_Other":[0, 0, 0],
    "FamilySize": [1, 4, 0],
})

test_scaled = scaler.transform(test_data)
preds = log_reg.predict(test_scaled)
results = ["Survived" if p == 1 else "Not Survived" for p in preds]

print(pd.DataFrame({"Passenger": [1,2,3], "Prediction": results}))

   Passenger    Prediction
0          1      Survived
1          2  Not Survived
2          3      Survived
```

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

After training and testing with the baseline model, I tried to tune the hyperparameters ***(C, penalty, solver, max_iter)*** with different ***test_size.*** I tried to tune with and without polynomial features.

a. **Without polynomial features**

```python
# --- Parameters ---

splits = [0.2, 0.3, 0.4]   # try different test sizes
param_grid = [ # try with valid combinations for penalty and solver
    {"C": [0.01, 0.1, 1, 10], "penalty": ["l1"], "solver": ["liblinear", "saga"], "max_iter": [100, 200, 500, 1000]},
    {"C": [0.01, 0.1, 1, 10], "penalty": ["l2"], "solver": ["liblinear", "saga", "lbfgs", "newton-cg", "sag"], "max_iter": [100, 200
]

results = []

for split in splits:
    print(clr.G+f"\n--- Train/Test Split: {1-split:.0%} train / {split:.0%} test ---"+clr.E)

    # Create split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=split, random_state=42, stratify=y
    )

    # Normalization
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Grid Search
    grid = GridSearchCV(LogisticRegression(), param_grid, cv=5, scoring="accuracy", n_jobs=-1)
    grid.fit(X_train_scaled, y_train)

    print("Best Parameters:", grid.best_params_)
    print("Best Training Accuracy:", grid.best_score_)

    # Evaluate on test set
    best_log_reg = grid.best_estimator_
    y_pred = best_log_reg.predict(X_test_scaled)

    test_acc = accuracy_score(y_test, y_pred)
    print("Test Accuracy:", test_acc)
    print(classification_report(y_test, y_pred))

    # Save results
    results.append({
        "split": split,
        "train_acc": grid.best_score_,
        "test_acc": test_acc
    })

# Show summary

df_results = pd.DataFrame(results)
print(clr.Y+"\n=== Summary Across Splits ==="+clr.E)
print(df_results)
```

And the results are as below:

```
--- Train/Test Split: 80% train / 20% test ---
Best Parameters: {'C': 1, 'max_iter': 100, 'penalty': 'l1', 'solver': 'liblinear'}
Best Training Accuracy: 0.8202895695853443
Test Accuracy: 0.8379888268156425
              precision    recall  f1-score   support

           0       0.85      0.89      0.87       110
           1       0.81      0.75      0.78        69

    accuracy                           0.84       179
   macro avg       0.83      0.82      0.83       179
weighted avg       0.84      0.84      0.84       179


--- Train/Test Split: 70% train / 30% test ---
Best Parameters: {'C': 0.1, 'max_iter': 100, 'penalty': 'l2', 'solver': 'saga'}
Best Training Accuracy: 0.8201806451612903
Test Accuracy: 0.832089552238806
              precision    recall  f1-score   support

           0       0.85      0.88      0.87       165
           1       0.80      0.76      0.78       103

    accuracy                           0.83       268
   macro avg       0.82      0.82      0.82       268
weighted avg       0.83      0.83      0.83       268


--- Train/Test Split: 60% train / 40% test ---
Best Parameters: {'C': 10, 'max_iter': 100, 'penalty': 'l1', 'solver': 'liblinear'}
Best Training Accuracy: 0.8109504496561453
Test Accuracy: 0.84593837535014
              precision    recall  f1-score   support

           0       0.85      0.90      0.88       220
           1       0.83      0.75      0.79       137

    accuracy                           0.85       357
   macro avg       0.84      0.83      0.83       357
weighted avg       0.85      0.85      0.84       357


=== Summary Across Splits ===
   split  train_acc  test_acc
0    0.2   0.820290  0.837989
1    0.3   0.820181  0.832090
2    0.4   0.810950  0.845938
```

b. With polynomial features

I used the same for training but with polynomial features, the differences in the code and the results are shown below, respectively:

```
# Polinomial Features
poly = PolynomialFeatures(degree=2, include_bias=False)
X_train_poly = poly.fit_transform(X_train_scaled)
X_test_poly = poly.transform(X_test_scaled)
```

```
=== Summary Across Splits ===
   split  train_acc  test_acc
0    0.2   0.827292  0.837989
1    0.3   0.837806  0.843284
2    0.4   0.822183  0.834734
```