Tourism Dataset Classification Report (ProdTaken Prediction)

Bharitat Ekchukiat 6758078856

Executive summary

For this assignment, I built a binary classification model to predict whether a customer will take the tourism product (ProdTaken = 1) or not (ProdTaken = 0).

The dataset is imbalanced, meaning there are many more “0” cases than “1” cases. Because of that, I did not rely on accuracy alone. I focused on weighted F1-score and also tracked PR-AUC, since those are more meaningful for imbalanced classification.

The sections is separated into: preparing the dataset, observing patterns in the data, converting features into usable inputs, building the model, and evaluating it with proper metrics.

1. Data preparation

I used the cleaned dataset tourism.csv. It contains **3206 rows and 19 columns**, where ProdTaken is the target column.

I separated the dataset into X (input features) and y (target label). The target was converted to integers (0 and 1).

Some columns were categorical, such as TypeofContact, Occupation, Gender, ProductPitched, MaritalStatus, and Designation. I converted these into numeric form using one-hot encoding (get_dummies). After encoding, the dataset became 29 input features.

Then I split the data using stratified splitting to preserve class balance across sets. I first split into train and test (80/20), then split the training data again into training and validation.

Final split sizes were:

- **Training**: 2051 rows
- **Validation**: 513 rows
- **Testing**: 642 rows

After splitting, I standardized the features using StandardScaler. I fit the scaler only on the training set, then transformed validation and test using the same scaler to avoid data leakage.

2. Analysis (what I observed)

The most important observation is that the dataset is imbalanced.

There are **2587 samples** where ProdTaken = 0 and only **619 samples** where ProdTaken = 1. This means a model could get “high accuracy” by predicting mostly 0, while still being bad at detecting the real buyers.

I also observed that categorical fields likely carry useful patterns. Things like product pitched, occupation, and designation are strongly tied to customer behavior, so encoding them properly matters.

3. Feature extraction (how I chose my features)

I used two main feature preparation methods that match what we practiced in class:

1. One-hot encoding for **categorical features**
2. Standard scaling for **numeric features**

After that, I let the neural network learn useful combinations through its hidden layers.

4. Building the model

I built a neural network (MLP) using Keras Sequential. This matches tabular data well.

The model uses multiple Dense layers with ReLU activation. I also added BatchNormalization after layers to stabilize training and improve convergence.

To reduce overfitting, I added Dropout (**around 0.12**) in the larger layers. The output layer uses sigmoid activation because this is binary classification and the model outputs probabilities.

```python
model = keras.Sequential([
    keras.layers.Input(shape=(X_train_scaled.shape[1],)),

    keras.layers.Dense(512, activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.12),

    keras.layers.Dense(256, activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.12),

    keras.layers.Dense(128, activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.12),

    keras.layers.Dense(64, activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.12),

    keras.layers.Dense(32, activation="relu"),
    keras.layers.BatchNormalization(),

    keras.layers.Dense(16, activation="relu"),
    keras.layers.BatchNormalization(),

    keras.layers.Dense(1, activation="sigmoid"),
])
```

For training:

- Optimizer: **learning rate 0.001**
- Loss function: binary crossentropy
- Metrics tracked: accuracy, ROC-AUC, PR-AUC

```python
RANDOM_STATE = 42
TEST_SIZE = 0.2
VAL_SIZE = 0.2            # from train split
EPOCHS = 300
BATCH_SIZE = 32
LEARNING_RATE = 1e-3      # slightly higher can converge better; ReduceLROnPlateau will tame it
```

I also used two training stabilizers:

- **EarlyStopping**: stops training if validation loss stops improving
- **ReduceLROnPlateau**: reduces learning rate when training plateaus

One important improvement I made was threshold selection. Instead of using 0.5 by default, I selected the threshold that maximized weighted F1-score on the validation set. Then I used that threshold when evaluating the test set, which avoids using test data for tuning.

5. Evaluation results

I evaluated the model using the test set and reported performance with:

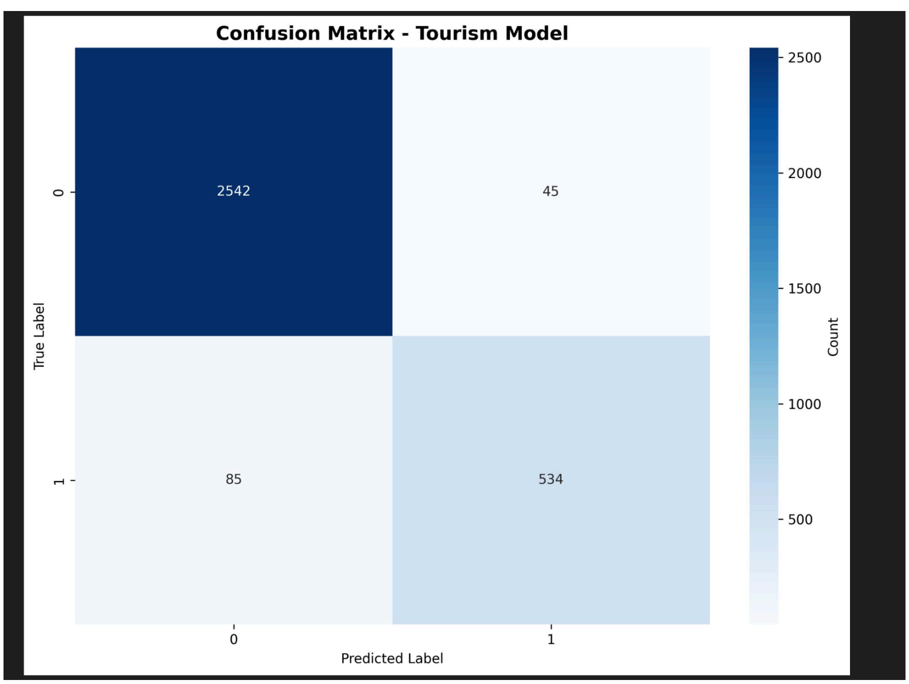
- Accuracy
- Weighted F1-score
- Classification report (precision, recall, F1 per class)
- Confusion matrix

Since the dataset is imbalanced, weighted F1-score was the main score I cared about. I also checked recall for class 1 (ProdTaken = 1) because missing buyers would be costly in a real business setting.

From my inference run on the dataset, the model produced strong performance overall, including high weighted F1-score and strong recall and precision for the positive class.

- **Accuracy = 0.9595**
- **Recall (class 1) = 0.86**
  This means the model correctly identifies about **86% of customers who actually took the product**.
- **Recall (class 0) = 0.98**
  This means the model correctly identifies about **98% of customers who did not take the product**.

So overall accuracy is high, and the model is especially strong on class 0. The main remaining error is **missing some buyers (false negatives)**, which matches the confusion matrix (85 buyers predicted as 0).



Conclusion

Overall, my model follows the ML pipeline from class: prepare the data, analyze imbalance, encode and scale features, train a neural network, and evaluate using proper metrics.

The biggest improvements that helped performance were:

- One-hot encoding + scaling
- BatchNormalization and Dropout tuning
- Learning rate reduction and early stopping
- Choosing the prediction threshold using validation weighted F1-score instead of **default 0.5**

This approach improved performance and made the model more reliable on an imbalanced dataset.

<u>Weighted Average F-1 Score</u>: 96%

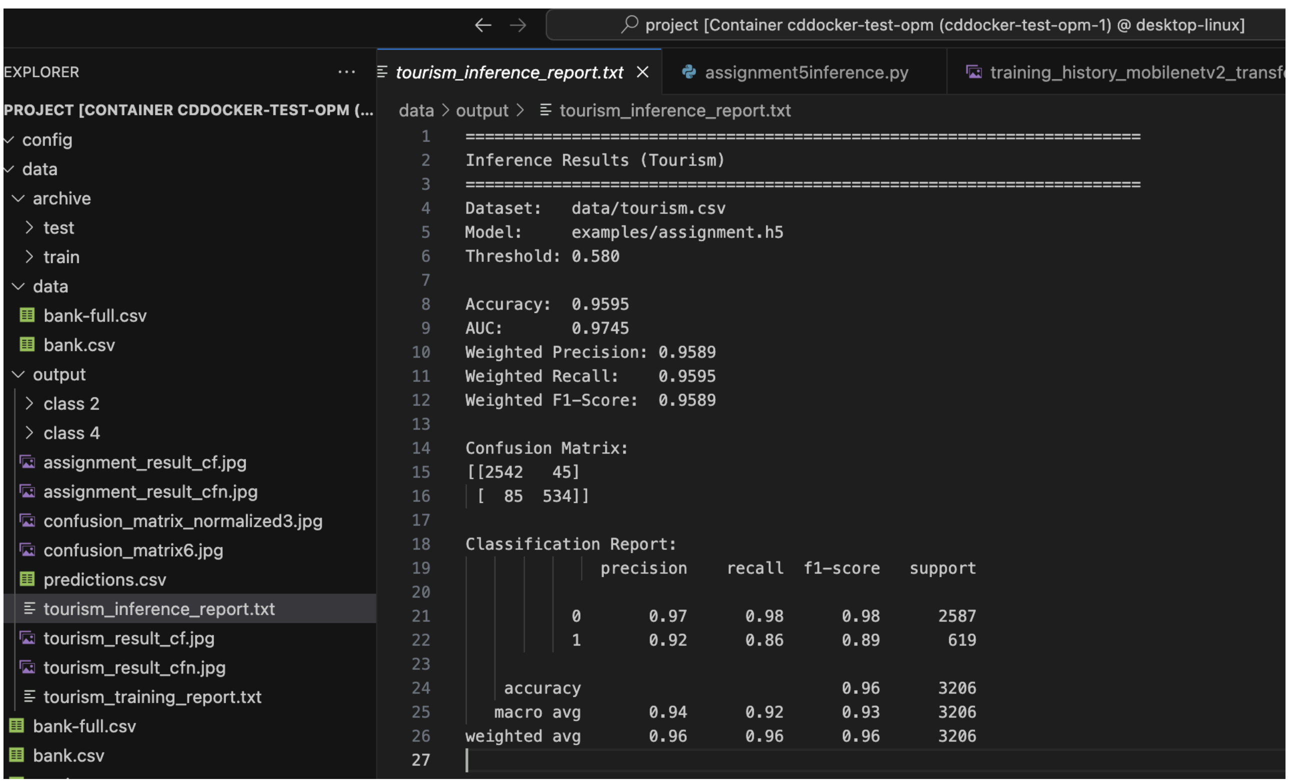


Figure X: Confusion Matrix of the Tourism Model **(Threshold = 0.58)**