6758002056

Karn Sakulngam

Class 5 Assignment

1. Collecting and preparing data

In this stage, I focused strictly on preparing the raw dataset (class5data.csv) to ensure it was clean, consistent, and free from data-entry errors before any modeling steps. The dataset contains 3,208 records and 19 columns. The main objective of this stage was to detect and correct inconsistent categorical values (e.g., “Fe Male” instead of “Female”), verify data types, and ensure there were no structural errors or missing values that could negatively affect downstream processing.

```
if "Gender" in df.columns:

   df["Gender"] = df["Gender"].replace({"Fe Male": "Female", "fe male": "Female",
"male ": "Male", "female ": "Female"})

   df["Gender"] = df["Gender"].replace({"M": "Male", "F": "Female"})
```

```
root@aca88d6803c7:/opt/project# python src/cleandata.py
(3208, 19)
Series([], dtype: int64)
```

Initial data:

```
18    38.0,Self Enquiry,3,6.0,Salaried,Fe Male,2,4.0,Deluxe,3.0,Unmarried,5.0,0,1,0,1.0,Manager,23686.0,0
```

Cleaned data:

```
18    38.0,Self Enquiry,3,6.0,Salaried,Female,2,4.0,Deluxe,3.0,Unmarried,5.0,0,1,0,1.0,Manager,23686.0,0
```

2. Splitting the data

After the raw dataset was cleaned and saved as class5data_clean.csv, I split the dataset before any model training to prevent data leakage and to ensure the model is evaluated on truly unseen data. Following the assignment recommendation, I used a 75% / 25% split, where 75% is used for training and validation and 25% is held out for final testing . The split was performed using train_test_split with shuffle=True and random_state=42 to make the split reproducible. As a result, the dataset was divided into 2,406 rows for train_validation.csv and 802 rows for test.csv, matching the total of 3,208 rows. Both files were saved permanently so the same exact test set can be reused for inference and final evaluation. This separation ensures that model performance metrics reflect real generalization rather than memorizing patterns from the training data.

```
root@aca88d6803c7:/opt/project# python src/splitdata.py
train_validation: (2406, 19)
test: (802, 19)
```

3. Feature extraction and engineering

Feature engineering was performed on all 18 input features in the dataset to prepare them for neural network training. The target variable is ProdTaken, and all other columns were used as predictors. The numerical features used were: Age, CityTier, DurationOfPitch, NumberOfPersonVisiting, NumberOfFollowups, PreferredPropertyStar, NumberOfTrips, Passport, PitchSatisfactionScore, OwnCar, NumberOfChildrenVisiting, and MonthlyIncome. These were kept because they represent measurable customer characteristics and behavior that may influence purchase decisions. They were standardized using StandardScaler to ensure that

large-value features such as MonthlyIncome do not dominate smaller-scale features like Age during model training.

The categorical features used were: TypeofContact, Occupation, Gender, ProductPitched, MaritalStatus, and Designation. These variables describe customer profile and marketing interaction context, which are relevant for predicting whether a product was taken. Since neural networks cannot process text values directly, these features were converted using one-hot encoding to transform each category into binary indicator columns. After preprocessing, the feature space expanded from 18 original input features to 34 processed features. All transformations were fitted only on the training data to prevent data leakage and ensure fair evaluation.

```
root@aca88d6803c7:/opt/project# python src/feature_engineering.py
X shape: (2406, 18)
Processed X shape: (2406, 34)
Numerical columns: ['Age', 'CityTier', 'DurationOfPitch', 'NumberOfPersonVisiting', 'NumberOfFollowups', 'PreferredPropertyStar', 'Nu
mberOfTrips', 'Passport', 'PitchSatisfactionScore', 'OwnCar', 'NumberOfChildrenVisiting', 'MonthlyIncome']
Categorical columns: ['TypeofContact', 'Occupation', 'Gender', 'ProductPitched', 'MaritalStatus', 'Designation']
Target distribution:
ProdTaken
0    1947
1     459
Name: count, dtype: int64
```

```
num_pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

cat_pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("encoder", OneHotEncoder(handle_unknown="ignore"))
])

preprocessor = ColumnTransformer([
    ("num", num_pipeline, numerical_cols),
    ("cat", cat_pipeline, categorical_cols)
])
```

4. Model building

Model building was performed using a feedforward neural network designed for binary classification. Since the target variable ProdTaken is binary (0 or 1), the output layer consists of one neuron with a sigmoid activation function to produce a probability between 0 and 1. The input layer size was set to 34 neurons, matching the number of processed features after feature engineering.

The model consists of three hidden Dense layers followed by one output layer. The hidden layers use ReLU activation to allow the network to learn non-linear relationships between customer characteristics and purchase behavior. Dropout layers were included between hidden layers to reduce overfitting by randomly deactivating neurons during training. In total, the network contains four Dense layers (3 hidden + 1 output).

The model was compiled using the Adam optimizer because it adapts the learning rate efficiently and works well for structured tabular data. The loss function used was Binary Crossentropy, which is appropriate for binary classification problems. Accuracy and AUC were selected as evaluation metrics. The model was trained on 2,406 training samples and validated before final evaluation on the 802-sample test set. The architecture was kept moderately deep to capture non-linear patterns while avoiding unnecessary complexity that could increase overfitting risk.

```
model = keras.Sequential([
    keras.layers.Input(shape=(input_dim,)),

    keras.layers.Dense(256, activation="relu", kernel_regularizer=keras.regularizers.l2(1e-4)),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.20),

    keras.layers.Dense(128, activation="relu", kernel_regularizer=keras.regularizers.l2(1e-4)),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.15),

    keras.layers.Dense(64, activation="relu", kernel_regularizer=keras.regularizers.l2(1e-4)),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.10),

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

Training results:

```
Confusion Matrix (Validation):
[[364  26]
 [ 35  57]]

Classification Report (Validation):
              precision    recall  f1-score   support

           0     0.9123    0.9333    0.9227       390
           1     0.6867    0.6196    0.6514        92

    accuracy                         0.8734       482
   macro avg     0.7995    0.7764    0.7871       482
weighted avg     0.8692    0.8734    0.8709       482
```

Inference result:

```
Confusion Matrix (Test):
[[586  56]
 [ 54 106]]

Classification Report (Test):
              precision    recall  f1-score   support

           0     0.9156    0.9128    0.9142       642
           1     0.6543    0.6625    0.6584       160

    accuracy                         0.8628       802
   macro avg     0.7850    0.7876    0.7863       802
weighted avg     0.8635    0.8628    0.8632       802
```

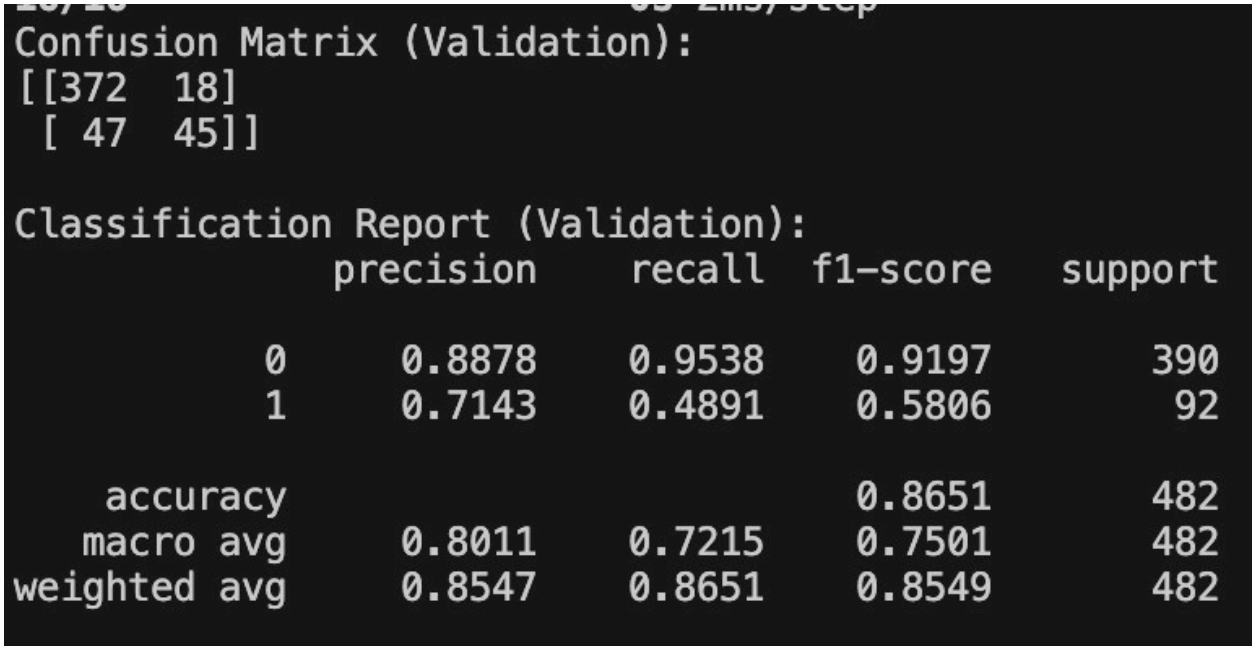
5. Evaluation

During evaluation, the initial model was trained with class weights to address the class imbalance in the training dataset (approximately 1,947 class 0 vs 459 class 1, or about 4.2:1). The purpose of applying class weights was to penalize misclassification of the minority class

(ProdTaken = 1) more heavily, thereby improving recall for that class. However, after evaluation, it was observed that although recall for class 1 improved, overall accuracy did not increase as expected and remained relatively low. This occurred because increasing the weight of the minority class forces the model to predict more positive cases, which can increase false positives and reduce overall accuracy. To balance this trade-off, the class weights were reduced (and later removed), allowing the model to learn from the natural distribution of the data. After adjusting the class weight, the model achieved higher overall accuracy (around 86%), while still maintaining reasonable recall for the minority class. This adjustment was made to achieve a better balance between accuracy and minority class performance rather than over-prioritizing one metric.

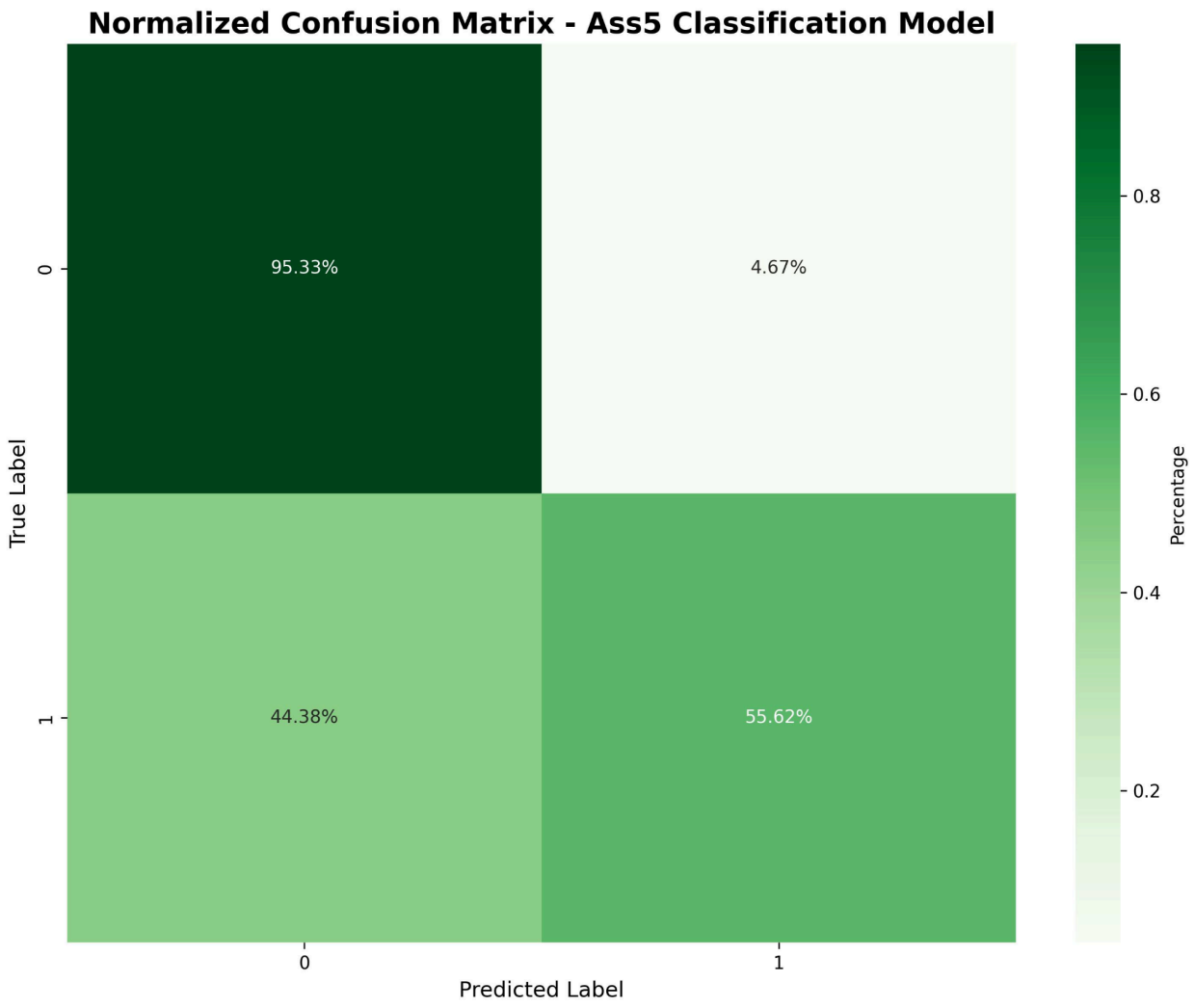
```
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=120,
    batch_size=128,
    callbacks=[early_stop, reduce_lr],
    verbose=1
)
```

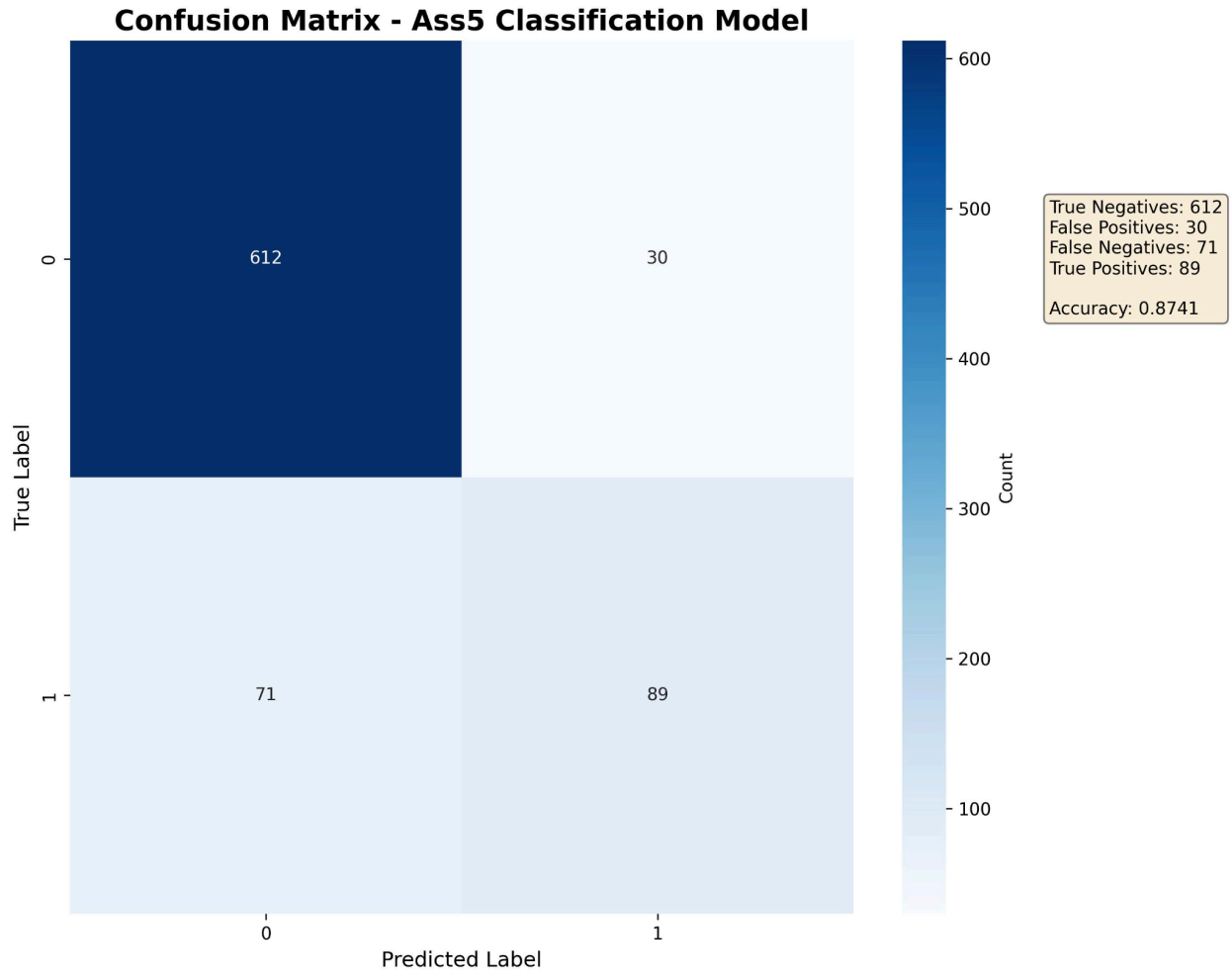
Training 2 result:



```
Confusion Matrix (Validation):
[[372  18]
 [ 47  45]]

Classification Report (Validation):
              precision    recall  f1-score   support

           0     0.8878    0.9538    0.9197       390
           1     0.7143    0.4891    0.5806        92

    accuracy                         0.8651       482
   macro avg     0.8011    0.7215    0.7501       482
weighted avg     0.8547    0.8651    0.8549       482
```

Inference 2 result:


Normalized Confusion Matrix - Ass5 Classification Model
95.33%
4.67%
44.38%
55.62%
True Label
Predicted Label
0
1
0
1
Percentage
0.8
0.6
0.4
0.2



Test 3:

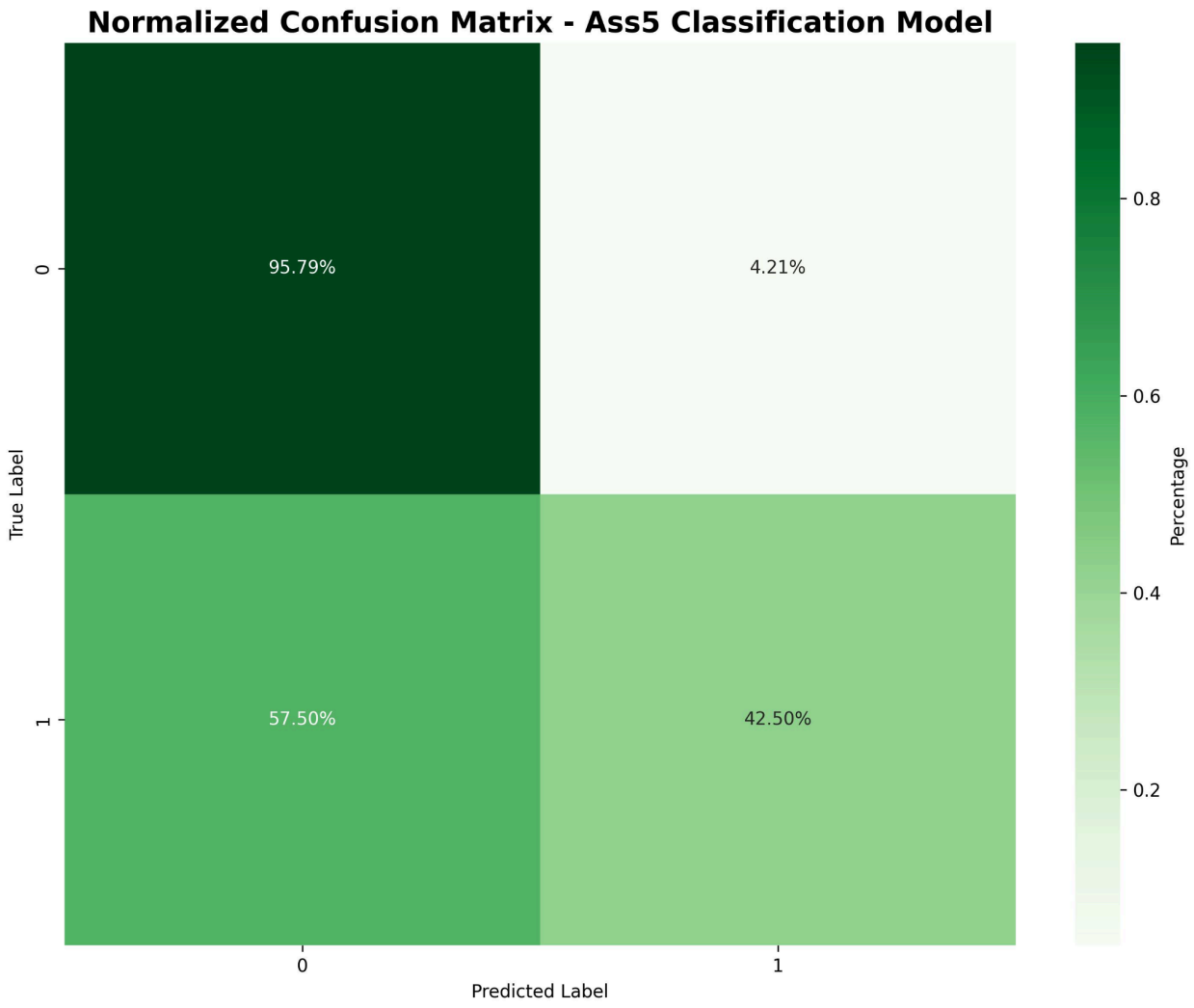
In the final test, I replaced standard binary cross-entropy with Binary Focal Crossentropy (γ=2.0) to handle class imbalance more effectively. The alpha parameter was automatically set from the training positive rate to give more weight to the minority class (ProdTaken = 1). I also increased regularization slightly (L2 = 2e-5) and applied stronger dropout (0.35 → 0.10 across layers) to reduce overfitting. Instead of using a fixed threshold of 0.5, I searched for the threshold that maximized validation accuracy. The model monitored val_auc and saved the best checkpoint before final evaluation.

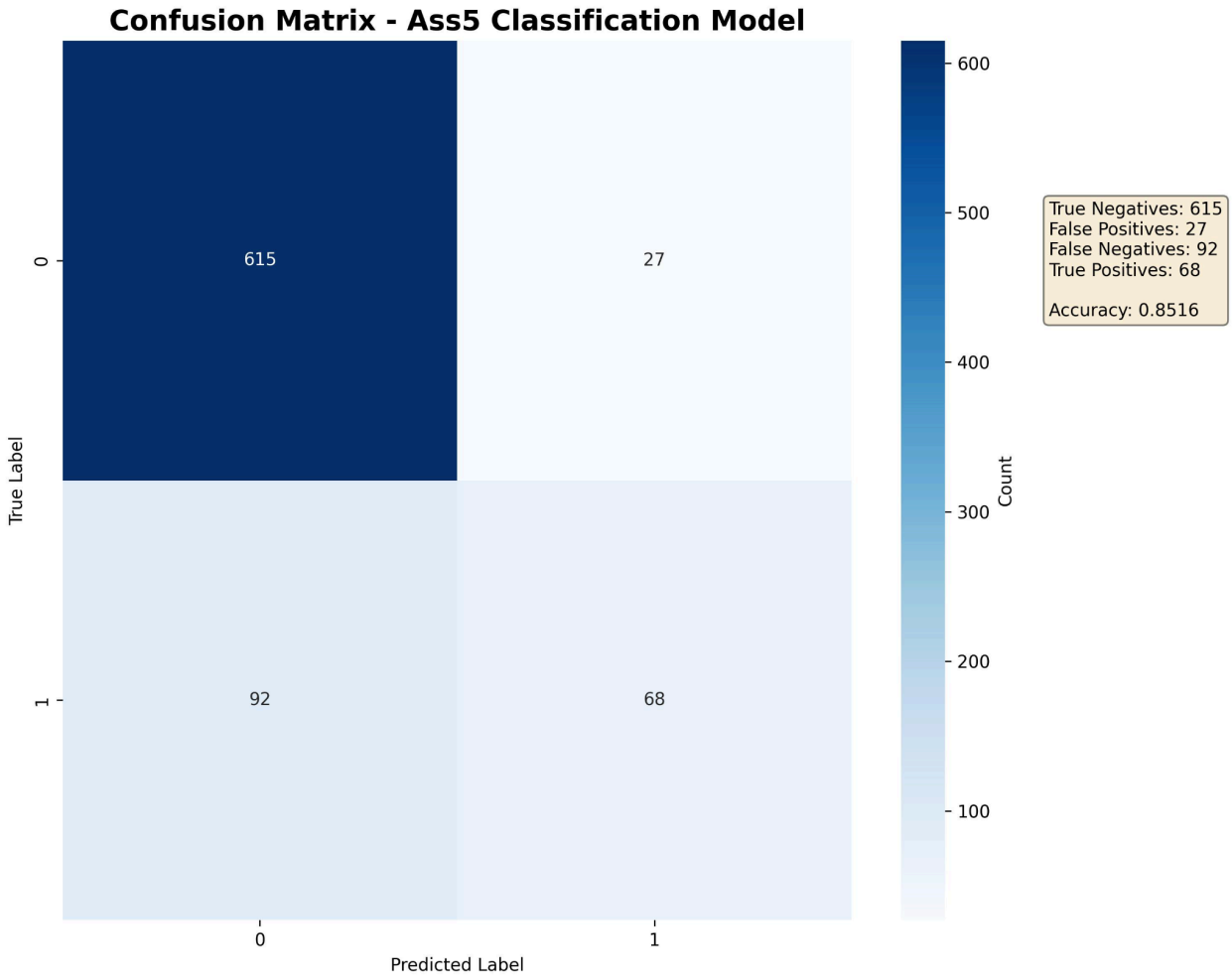
The final confusion matrix shows TN=615, FP=27, FN=92, TP=68, resulting in accuracy = 0.8516. The model performs very well on class 0 (95.79% correct) but struggles on class 1 (recall = 42.50%). Overall, focal loss improved training stability and minority focus, but false negatives for class 1 remain the main limitation.

Training 3 result:

```
Train positive rate: 0.1907  -> focal alpha used: 0.8093
Best threshold (val accuracy): 0.41
Best val accuracy: 0.8651

Confusion Matrix (Validation):
[[360  30]
 [ 35  57]]

Classification Report (Validation):
              precision    recall  f1-score   support

           0     0.9114    0.9231    0.9172       390
           1     0.6552    0.6196    0.6369        92

    accuracy                         0.8651       482
   macro avg     0.7833    0.7713    0.7770       482
weighted avg     0.8625    0.8651    0.8637       482
```

Inference 3 result:


Normalized Confusion Matrix - Ass5 Classification Model
95.79%
4.21%
57.50%
42.50%
True Label
0
1
Predicted Label
0
1
Percentage
0.8
0.6
0.4
0.2



6. Real-life application

In real-life application, this model can be used as a customer purchase prediction system to support marketing decisions. It predicts whether a customer will take the offered travel product (ProdTaken = 1) based on demographic and behavioral features such as age, income, number of trips, and contact type. The model can be integrated into a CRM system to generate a probability score for each new lead, allowing the company to prioritize high-potential customers. Based on the final results (accuracy ≈ 85% and class-1 recall ≈ 42.5%), the model is strong at identifying customers who are unlikely to purchase but still misses many actual buyers. Therefore, it is more

suitable as a lead-ranking or filtering tool rather than a fully automated decision system. In practice, the company can adjust the decision threshold depending on business goals—for example, lowering it to increase recall if maximizing sales is the priority, or keeping it higher to reduce marketing costs. Overall, the model is practical for lead scoring and campaign optimization, but further improvement in minority-class detection would enhance its real-world impact.