IoT_SmartParking_XGBoost

February 25, 2025

0.0.1 XGBoost for Predicting Parking Occupancy Status on Feature Engineered Dataset

[1]:
```
# Install xgboost if already not installed
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\mahesh\anaconda3\lib\site-
packages (2.1.4)
Requirement already satisfied: numpy in c:\users\mahesh\anaconda3\lib\site-
packages (from xgboost) (1.26.4)
Requirement already satisfied: scipy in c:\users\mahesh\anaconda3\lib\site-
packages (from xgboost) (1.11.4)
```

[14]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import xgboost as xgb
from xgboost import XGBClassifier, plot_importance
from sklearn.preprocessing import MinMaxScaler, LabelEncoder  # Import␣
↪LabelEncoder here
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report
import seaborn as sns

# Load the feature engineered IoT Smart Parking dataset
df = pd.read_csv('IoT_SmartParking_Processed.csv')

# Inspect the data
print(df.head())

# Preprocessing
# Assuming 'occupied_spots' is the target variable and other columns are␣
↪features
# The original code used 'occupancy', which was not in the DataFrame
target_column = 'Occupancy_Status'
features = [col for col in df.columns if col != target_column and col !=␣
↪'Timestamp' and df[col].dtype != object]
```

```python
X = df[features].values
y = df[target_column].values

# Normalize features
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Encode target variable to numeric using LabelEncoder
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded,
 ↪test_size=0.2, random_state=42)

# Initialize and train the XGBoost model
model = XGBClassifier(
    objective='binary:logistic',  # For binary classification
    colsample_bytree=0.3,         # Fraction of features to use for building
 ↪trees
    learning_rate=0.1,            # Step size at each iteration while moving
 ↪towards a minimum
    max_depth=5,                  # Maximum depth of the trees
    alpha=10,                     # L1 regularization term on weights
    n_estimators=100,             # Number of trees to build
    n_jobs=-1                     # Use all CPU threads
)

# Train the model

# Prepare the evaluation data
evals = [(X_train, y_train), (X_test, y_test)]

# Fit the model while tracking evaluation results
evals_result = {}
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

# Get the loss values from evals_result (stored during training)
#vals_result = model.evals_result()

# Extract the logloss for training and validation data
#train_loss = evals_result['validation_0']['logloss']
#val_loss = evals_result['validation_1']['logloss']
```

```python
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Print the model parameters (Hyperparameters used for training)
print("Model Parameters:")
print(model.get_params())

# Example if you're using pandas DataFrame
#model.feature_names = X_train.columns.tolist()  # Assuming X_train is a pandas␣
 ↪DataFrame

print("Model Features:")
print(features)

# Print the number of trees and booster structure (model summary)
# Print the number of trees (using n_estimators)
print(f"\nNumber of trees in the model: {model.get_params()['n_estimators']}")

# Optionally, dump the model structure to a text file
model.get_booster().dump_model('xgboost_model_dump.txt', with_stats=True)
```

```
                      Timestamp  Parking_Spot_ID  Sensor_Reading_Proximity  \
0 2021-01-01 00:00:00.000000000               20                  1.023651
1 2021-01-02 06:39:16.756756756               49                  3.903349
2 2021-01-03 13:18:33.513513513               38                 10.315709
3 2021-01-04 19:57:50.270270270               31                  6.588039
4 2021-01-06 02:37:07.027027027                8                  8.213969

   Sensor_Reading_Pressure  Vehicle_Type_Weight  Vehicle_Type_Height  \
0                 1.541461          1831.770127             4.392528
1                 1.621719          1330.815754             4.595638
2                 6.292374          1255.134827             4.313721
3                 1.659870          1523.442919             3.567329
4                 3.278467          1758.490837             5.145836

    User_Type  Weather_Temperature  Weather_Precipitation  \
0     Visitor            18.092553                      1
1  Registered            13.397533                      0
2  Registered            21.687410                      0
3     Visitor            18.683461                      0
4     Visitor            19.214876                      0

  Nearby_Traffic_Level  …  DayOfWeek_4  DayOfWeek_5  DayOfWeek_6  \
0                  Low  …          1.0          0.0          0.0
1                  Low  …          0.0          1.0          0.0
2                 High  …          0.0          0.0          1.0
3               Medium  …          0.0          0.0          0.0
```

```
4                 High  …          0.0          0.0          0.0

   DayOfWeek_0.1  DayOfWeek_1.1  DayOfWeek_2.1 DayOfWeek_3.1 DayOfWeek_4.1  \
0            0.0            0.0            0.0           0.0           1.0
1            0.0            0.0            0.0           0.0           0.0
2            0.0            0.0            0.0           0.0           0.0
3            1.0            0.0            0.0           0.0           0.0
4            0.0            0.0            1.0           0.0           0.0

  DayOfWeek_5.1 DayOfWeek_6.1
0           0.0           0.0
1           1.0           0.0
2           0.0           1.0
3           0.0           0.0
4           0.0           0.0

[5 rows x 53 columns]
Accuracy: 100.00%
Model Parameters:
{'objective': 'binary:logistic', 'base_score': None, 'booster': None,
'callbacks': None, 'colsample_bylevel': None, 'colsample_bynode': None,
'colsample_bytree': 0.3, 'device': None, 'early_stopping_rounds': None,
'enable_categorical': False, 'eval_metric': None, 'feature_types': None,
'gamma': None, 'grow_policy': None, 'importance_type': None,
'interaction_constraints': None, 'learning_rate': 0.1, 'max_bin': None,
'max_cat_threshold': None, 'max_cat_to_onehot': None, 'max_delta_step': None,
'max_depth': 5, 'max_leaves': None, 'min_child_weight': None, 'missing': nan,
'monotone_constraints': None, 'multi_strategy': None, 'n_estimators': 100,
'n_jobs': -1, 'num_parallel_tree': None, 'random_state': None, 'reg_alpha':
None, 'reg_lambda': None, 'sampling_method': None, 'scale_pos_weight': None,
'subsample': None, 'tree_method': None, 'validate_parameters': None,
'verbosity': None, 'alpha': 10}
Model Features:
['Parking_Spot_ID', 'Sensor_Reading_Proximity', 'Sensor_Reading_Pressure',
'Vehicle_Type_Weight', 'Vehicle_Type_Height', 'Weather_Temperature',
'Weather_Precipitation', 'Entry_Time', 'Exit_Time', 'Electric_Vehicle',
'Reserved_Status', 'Occupancy_Rate', 'Payment_Amount', 'Parking_Violation',
'Sensor_Reading_Ultrasonic', 'Parking_Duration', 'Environmental_Noise_Level',
'Dynamic_Pricing_Factor', 'Proximity_To_Exit', 'User_Parking_History', 'Hour',
'Month', 'IsWeekend', 'Occupancy_Status_Numeric', 'RollingAvg_Occupancy',
'Prev_Occupancy', 'Prev2_Occupancy', 'Rainfall', 'Temperature',
'Hourly_Occupancy', 'Daily_Occupancy', 'DayOfWeek_0', 'DayOfWeek_1',
'DayOfWeek_2', 'DayOfWeek_3', 'DayOfWeek_4', 'DayOfWeek_5', 'DayOfWeek_6',
'DayOfWeek_0.1', 'DayOfWeek_1.1', 'DayOfWeek_2.1', 'DayOfWeek_3.1',
'DayOfWeek_4.1', 'DayOfWeek_5.1', 'DayOfWeek_6.1']

Number of trees in the model: 100
```

```
[10]: import pandas as pd
import numpy as np
import xgboost as xgb
from xgboost import XGBClassifier, plot_importance
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset (replace 'data.csv' with your actual file)
df = pd.read_csv("IoT_SmartParking_Processed.csv")

# Preprocessing
target_column = 'Occupancy_Status'
features = [col for col in df.columns if col != target_column and col !=␣
↪'Timestamp' and df[col].dtype != object]

X = df[features].values
y = df[target_column].values

# Normalize features
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Encode target variable to numeric using LabelEncoder
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded,␣
↪test_size=0.2, random_state=42)

# Initialize and train the XGBoost model
model = XGBClassifier(
    objective='binary:logistic',  # For binary classification
    colsample_bytree=0.3,         # Fraction of features to use for building␣
↪trees
    learning_rate=0.1,            # Step size at each iteration while moving␣
↪towards a minimum
    max_depth=5,                  # Maximum depth of the trees
    alpha=10,                     # L1 regularization term on weights
    n_estimators=100,             # Number of trees to build
    n_jobs=-1                     # Use all CPU threads
)
```

```python
# Prepare the evaluation data
evals = [(X_train, y_train), (X_test, y_test)]

# Fit the model while tracking evaluation results
evals_result = {}
model.fit(X_train, y_train, eval_set=evals, verbose=True)

# Get the predictions
y_pred = model.predict(X_test)

# Get the loss values from evals_result (stored during training)
evals_result = model.evals_result()

# Extract the logloss for training and validation data
train_loss = evals_result['validation_0']['logloss']
val_loss = evals_result['validation_1']['logloss']

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Print the model parameters
print("Model Parameters:")
print(model.get_params())
```

```
[0]     validation_0-logloss:0.68531    validation_1-logloss:0.69696
[1]     validation_0-logloss:0.68305    validation_1-logloss:0.69632
[2]     validation_0-logloss:0.67933    validation_1-logloss:0.69447
[3]     validation_0-logloss:0.67674    validation_1-logloss:0.69494
[4]     validation_0-logloss:0.58898    validation_1-logloss:0.60463
[5]     validation_0-logloss:0.58686    validation_1-logloss:0.60376
[6]     validation_0-logloss:0.58368    validation_1-logloss:0.60326
[7]     validation_0-logloss:0.58173    validation_1-logloss:0.60090
[8]     validation_0-logloss:0.57975    validation_1-logloss:0.60043
[9]     validation_0-logloss:0.50915    validation_1-logloss:0.52737
[10]    validation_0-logloss:0.50745    validation_1-logloss:0.52748
[11]    validation_0-logloss:0.44853    validation_1-logloss:0.46636
[12]    validation_0-logloss:0.44801    validation_1-logloss:0.46567
[13]    validation_0-logloss:0.44661    validation_1-logloss:0.46528
[14]    validation_0-logloss:0.39684    validation_1-logloss:0.41356
[15]    validation_0-logloss:0.39607    validation_1-logloss:0.41271
[16]    validation_0-logloss:0.39607    validation_1-logloss:0.41271
[17]    validation_0-logloss:0.39555    validation_1-logloss:0.41193
[18]    validation_0-logloss:0.35302    validation_1-logloss:0.36774
[19]    validation_0-logloss:0.35302    validation_1-logloss:0.36774
[20]    validation_0-logloss:0.31625    validation_1-logloss:0.32953
[21]    validation_0-logloss:0.31625    validation_1-logloss:0.32953
[22]    validation_0-logloss:0.31057    validation_1-logloss:0.32401
```

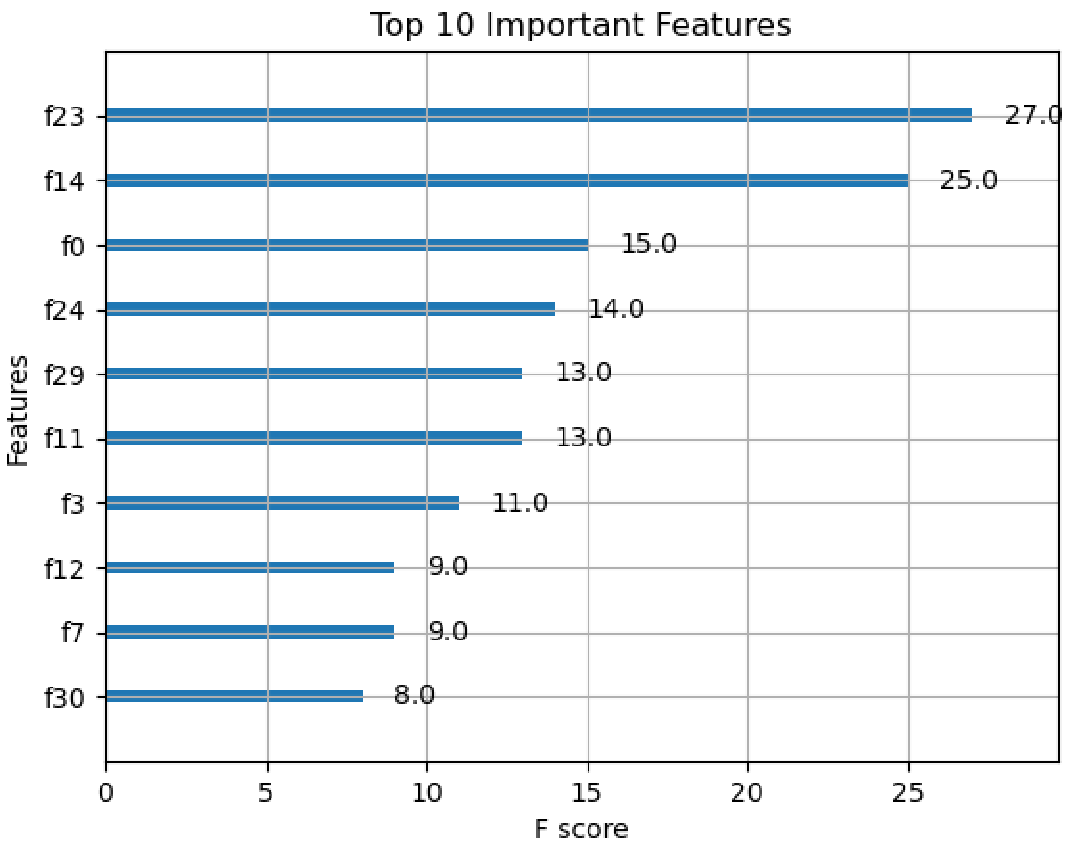
```
[23]	validation_0-logloss:0.30540	validation_1-logloss:0.31978
[24]	validation_0-logloss:0.27478	validation_1-logloss:0.28782
[25]	validation_0-logloss:0.27478	validation_1-logloss:0.28782
[26]	validation_0-logloss:0.24796	validation_1-logloss:0.25981
[27]	validation_0-logloss:0.24796	validation_1-logloss:0.25981
[28]	validation_0-logloss:0.24465	validation_1-logloss:0.25718
[29]	validation_0-logloss:0.24465	validation_1-logloss:0.25718
[30]	validation_0-logloss:0.22144	validation_1-logloss:0.23286
[31]	validation_0-logloss:0.22144	validation_1-logloss:0.23286
[32]	validation_0-logloss:0.22144	validation_1-logloss:0.23286
[33]	validation_0-logloss:0.21941	validation_1-logloss:0.23126
[34]	validation_0-logloss:0.21741	validation_1-logloss:0.23009
[35]	validation_0-logloss:0.21741	validation_1-logloss:0.23009
[36]	validation_0-logloss:0.21566	validation_1-logloss:0.22878
[37]	validation_0-logloss:0.21566	validation_1-logloss:0.22878
[38]	validation_0-logloss:0.21566	validation_1-logloss:0.22878
[39]	validation_0-logloss:0.21566	validation_1-logloss:0.22878
[40]	validation_0-logloss:0.21566	validation_1-logloss:0.22878
[41]	validation_0-logloss:0.21566	validation_1-logloss:0.22878
[42]	validation_0-logloss:0.21425	validation_1-logloss:0.22781
[43]	validation_0-logloss:0.19457	validation_1-logloss:0.20697
[44]	validation_0-logloss:0.19457	validation_1-logloss:0.20697
[45]	validation_0-logloss:0.17713	validation_1-logloss:0.18849
[46]	validation_0-logloss:0.17713	validation_1-logloss:0.18849
[47]	validation_0-logloss:0.17713	validation_1-logloss:0.18849
[48]	validation_0-logloss:0.17713	validation_1-logloss:0.18849
[49]	validation_0-logloss:0.17624	validation_1-logloss:0.18809
[50]	validation_0-logloss:0.17624	validation_1-logloss:0.18809
[51]	validation_0-logloss:0.17624	validation_1-logloss:0.18809
[52]	validation_0-logloss:0.17550	validation_1-logloss:0.18750
[53]	validation_0-logloss:0.17550	validation_1-logloss:0.18750
[54]	validation_0-logloss:0.17550	validation_1-logloss:0.18750
[55]	validation_0-logloss:0.16018	validation_1-logloss:0.17120
[56]	validation_0-logloss:0.14654	validation_1-logloss:0.15667
[57]	validation_0-logloss:0.14654	validation_1-logloss:0.15667
[58]	validation_0-logloss:0.13438	validation_1-logloss:0.14372
[59]	validation_0-logloss:0.13438	validation_1-logloss:0.14372
[60]	validation_0-logloss:0.13438	validation_1-logloss:0.14372
[61]	validation_0-logloss:0.13438	validation_1-logloss:0.14372
[62]	validation_0-logloss:0.13408	validation_1-logloss:0.14353
[63]	validation_0-logloss:0.12325	validation_1-logloss:0.13197
[64]	validation_0-logloss:0.12310	validation_1-logloss:0.13189
[65]	validation_0-logloss:0.11342	validation_1-logloss:0.12156
[66]	validation_0-logloss:0.10475	validation_1-logloss:0.11230
[67]	validation_0-logloss:0.09698	validation_1-logloss:0.10399
[68]	validation_0-logloss:0.09698	validation_1-logloss:0.10399
[69]	validation_0-logloss:0.09698	validation_1-logloss:0.10399
[70]	validation_0-logloss:0.09698	validation_1-logloss:0.10399
```

```
[71]	validation_0-logloss:0.09698	validation_1-logloss:0.10399
[72]	validation_0-logloss:0.09698	validation_1-logloss:0.10399
[73]	validation_0-logloss:0.09698	validation_1-logloss:0.10399
[74]	validation_0-logloss:0.09698	validation_1-logloss:0.10399
[75]	validation_0-logloss:0.09000	validation_1-logloss:0.09653
[76]	validation_0-logloss:0.09000	validation_1-logloss:0.09653
[77]	validation_0-logloss:0.09000	validation_1-logloss:0.09653
[78]	validation_0-logloss:0.09000	validation_1-logloss:0.09653
[79]	validation_0-logloss:0.09000	validation_1-logloss:0.09653
[80]	validation_0-logloss:0.08374	validation_1-logloss:0.08982
[81]	validation_0-logloss:0.08374	validation_1-logloss:0.08982
[82]	validation_0-logloss:0.08374	validation_1-logloss:0.08982
[83]	validation_0-logloss:0.07810	validation_1-logloss:0.08379
[84]	validation_0-logloss:0.07303	validation_1-logloss:0.07836
[85]	validation_0-logloss:0.07303	validation_1-logloss:0.07836
[86]	validation_0-logloss:0.07303	validation_1-logloss:0.07836
[87]	validation_0-logloss:0.07303	validation_1-logloss:0.07836
[88]	validation_0-logloss:0.07303	validation_1-logloss:0.07836
[89]	validation_0-logloss:0.07303	validation_1-logloss:0.07836
[90]	validation_0-logloss:0.07303	validation_1-logloss:0.07836
[91]	validation_0-logloss:0.06846	validation_1-logloss:0.07347
[92]	validation_0-logloss:0.06846	validation_1-logloss:0.07347
[93]	validation_0-logloss:0.06846	validation_1-logloss:0.07347
[94]	validation_0-logloss:0.06435	validation_1-logloss:0.06906
[95]	validation_0-logloss:0.06063	validation_1-logloss:0.06508
[96]	validation_0-logloss:0.06063	validation_1-logloss:0.06508
[97]	validation_0-logloss:0.06063	validation_1-logloss:0.06508
[98]	validation_0-logloss:0.05729	validation_1-logloss:0.06150
[99]	validation_0-logloss:0.05427	validation_1-logloss:0.05826
Accuracy: 100.00%
Model Parameters:
{'objective': 'binary:logistic', 'base_score': None, 'booster': None,
'callbacks': None, 'colsample_bylevel': None, 'colsample_bynode': None,
'colsample_bytree': 0.3, 'device': None, 'early_stopping_rounds': None,
'enable_categorical': False, 'eval_metric': None, 'feature_types': None,
'gamma': None, 'grow_policy': None, 'importance_type': None,
'interaction_constraints': None, 'learning_rate': 0.1, 'max_bin': None,
'max_cat_threshold': None, 'max_cat_to_onehot': None, 'max_delta_step': None,
'max_depth': 5, 'max_leaves': None, 'min_child_weight': None, 'missing': nan,
'monotone_constraints': None, 'multi_strategy': None, 'n_estimators': 100,
'n_jobs': -1, 'num_parallel_tree': None, 'random_state': None, 'reg_alpha':
None, 'reg_lambda': None, 'sampling_method': None, 'scale_pos_weight': None,
'subsample': None, 'tree_method': None, 'validate_parameters': None,
'verbosity': None, 'alpha': 10}
```

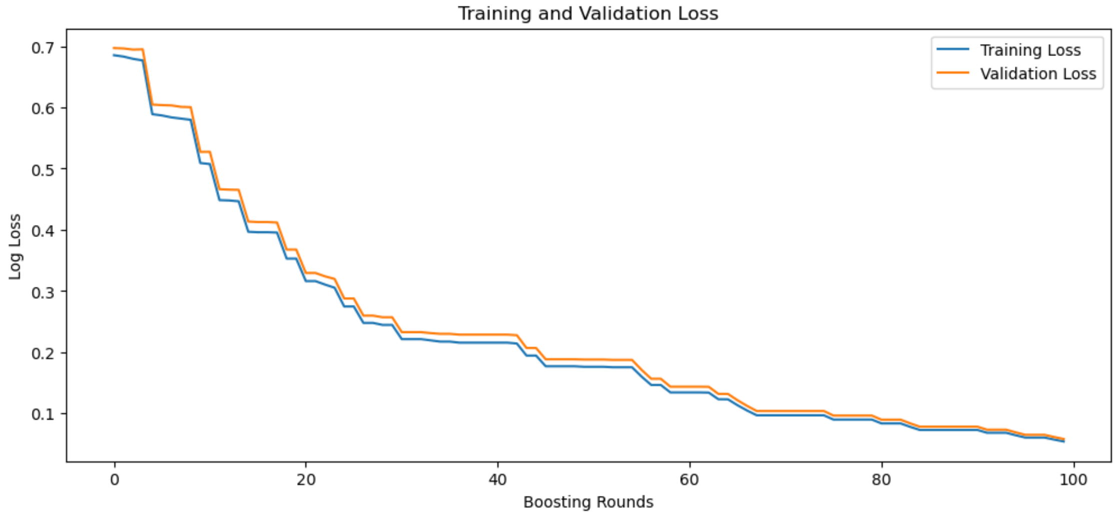
[15]:
```
# Print feature importance
plt.figure(figsize=(10, 6))
```

```
plot_importance(model, importance_type='weight', max_num_features=10,
    title="Top 10 Important Features")
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



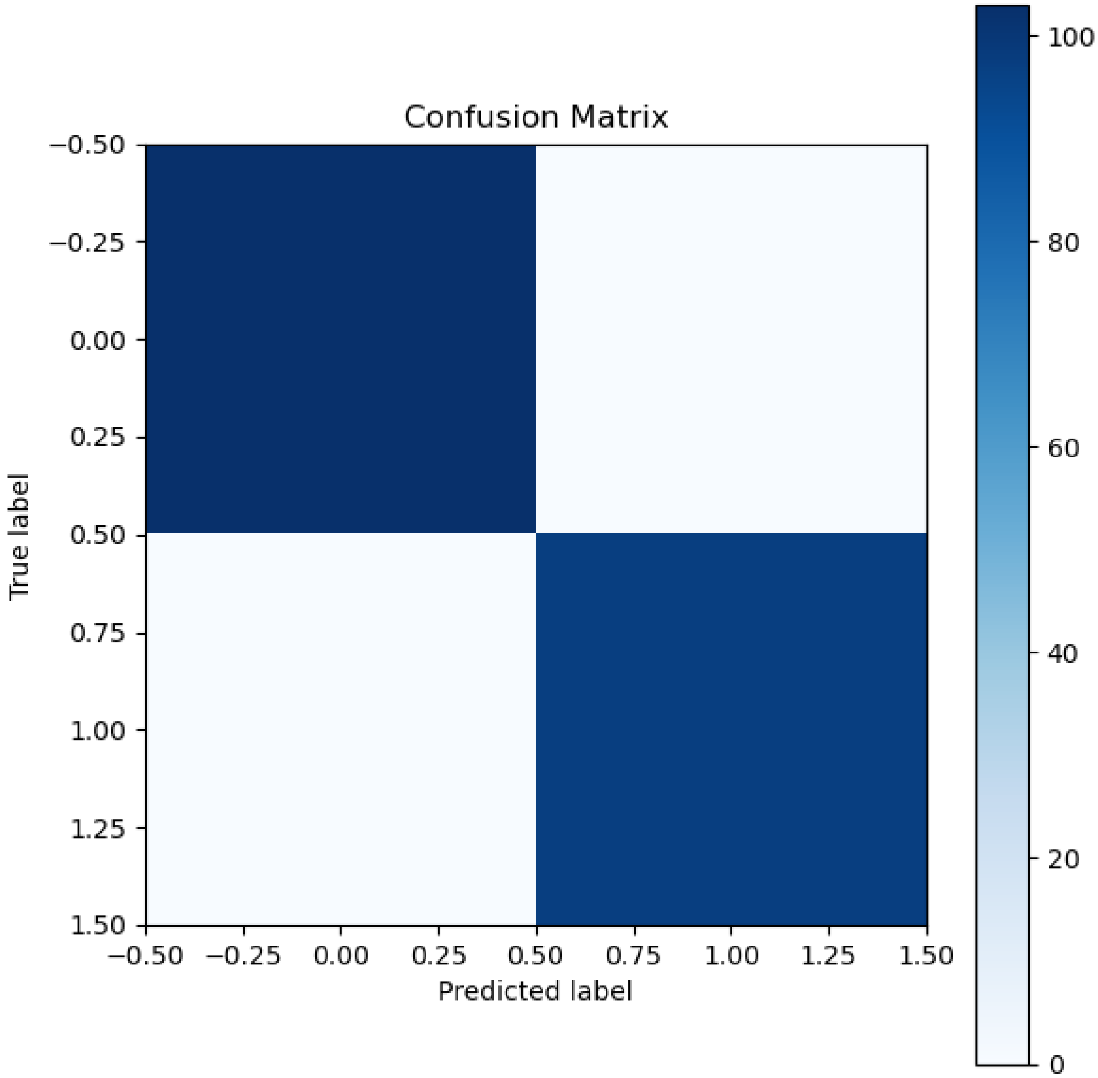
[16]:
```
# Plot the training and validation loss
plt.figure(figsize=(12, 5))
plt.plot(train_loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.xlabel('Boosting Rounds')
plt.ylabel('Log Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.show()
```



0.0.2 As can be seen above, the training and validation loss curves are well aligned

0.0.3 and loss exponentially reduced

[17]:
```python
# Plot the confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 6))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

# Print the classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

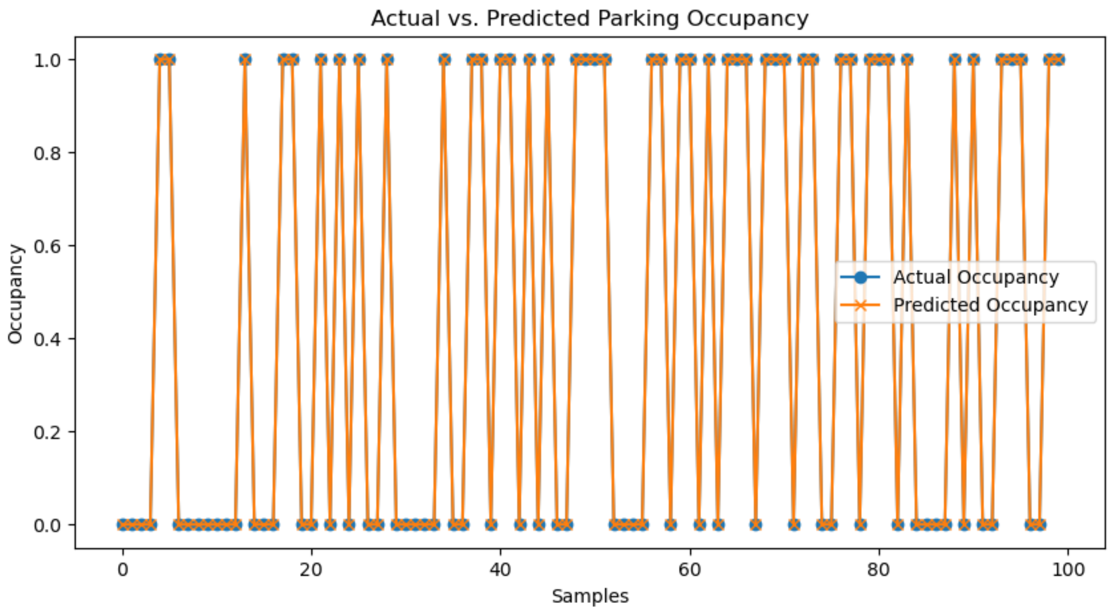



```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       103
           1       1.00      1.00      1.00        97

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200
```

[18]:

```
# Plot actual vs predicted values
plt.figure(figsize=(10, 5))
```

```
plt.plot(y_test[:100], label='Actual Occupancy', marker='o')
plt.plot(y_pred[:100], label='Predicted Occupancy', marker='x')
plt.xlabel('Samples')
plt.ylabel('Occupancy')
plt.legend()
plt.title('Actual vs. Predicted Parking Occupancy')
plt.show()
```



0.0.4 As can be seen above F1-Score, Accuracy, Precision and Recall of 100% achieved using XGBoost

0.0.5 for prediction of Parking Occupancy Status using Feature Engineered Dataset