Exemplar_Build an XGBoost model

January 7, 2024

1 Exemplar: Build an XGBoost model

1.1 Introduction

In this activity, you'll build on the skills and techniques you learned in the decision tree and random forest lessons to construct your own XGBoost classification model. The XGBoost model is a very powerful extension of decision trees, so having a strong working familiarity with this process will strengthen your skills and resume as a data professional.

This activity is a continuation of the airlines project in which you built decision tree and random forest models. You will use the same data, but this time you will train, tune, and evaluate an XGBoost model. You'll then compare the performance of all three models and decide which model is best. Finally, you'll explore the feature importances of your model and identify the features that most contribute to customer satisfaction.

1.2 Step 1: Imports

1.2.1 Import packages

Begin with your import statements. First, import `pandas`, `numpy`, and `matplotlib` for data preparation. Next, import scikit-learn (`sklearn`) for model preparation and evaluation. Then, import `xgboost`, which provides the classification algorithm you'll implement to formulate your predictive model.

[1]:
```
# Import relevant libraries and modules.

import numpy as np
import pandas as pd
import matplotlib as plt
import pickle

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn import metrics

from xgboost import XGBClassifier
from xgboost import plot_importance
```

1.2.2 Load the dataset

To formulate your model, `pandas` is used to import a csv of airline passenger satisfaction data called `Invistico_Airline.csv`. This DataFrame is called `airline_data`. As shown in this cell, the dataset has been automatically loaded in for you. You do not need to download the .csv file, or provide more code, in order to access the dataset and proceed with this lab. Please continue with this activity by completing the following instructions.

[2]:
```
# RUN THIS CELL TO IMPORT YOUR DATA.

### YOUR CODE HERE ###

airline_data = pd.read_csv('Invistico_Airline.csv', error_bad_lines=False)
```

1.2.3 Display the data

Examine the first 10 rows of data to familiarize yourself with the dataset.

[3]:
```
# Display first ten rows of data.

### YOUR CODE HERE ###

airline_data.head(10)
```

[3]:

	satisfaction	Customer Type	Age	Type of Travel	Class	\
0	satisfied	Loyal Customer	65	Personal Travel	Eco	
1	satisfied	Loyal Customer	47	Personal Travel	Business	
2	satisfied	Loyal Customer	15	Personal Travel	Eco	
3	satisfied	Loyal Customer	60	Personal Travel	Eco	
4	satisfied	Loyal Customer	70	Personal Travel	Eco	
5	satisfied	Loyal Customer	30	Personal Travel	Eco	
6	satisfied	Loyal Customer	66	Personal Travel	Eco	
7	satisfied	Loyal Customer	10	Personal Travel	Eco	
8	satisfied	Loyal Customer	56	Personal Travel	Business	
9	satisfied	Loyal Customer	22	Personal Travel	Eco	

	Flight Distance	Seat comfort	Departure/Arrival time convenient	\
0	265	0	0	
1	2464	0	0	
2	2138	0	0	
3	623	0	0	
4	354	0	0	
5	1894	0	0	
6	227	0	0	
7	1812	0	0	
8	73	0	0	
9	1556	0	0	

	Food and drink	Gate location	…	Online support	Ease of Online booking	\
0	0	2	…	2	3	
1	0	3	…	2	3	
2	0	3	…	2	2	
3	0	3	…	3	1	
4	0	3	…	4	2	
5	0	3	…	2	2	
6	0	3	…	5	5	
7	0	3	…	2	2	
8	0	3	…	5	4	
9	0	3	…	2	2	

	On-board service	Leg room service	Baggage handling	Checkin service	\
0	3	0	3	5	
1	4	4	4	2	
2	3	3	4	4	
3	1	0	1	4	
4	2	0	2	4	
5	5	4	5	5	
6	5	0	5	5	
7	3	3	4	5	
8	4	0	1	5	
9	2	4	5	3	

	Cleanliness	Online boarding	Departure Delay in Minutes	\
0	3	2	0	
1	3	2	310	
2	4	2	0	
3	1	3	0	
4	2	5	0	
5	4	2	0	
6	5	3	17	
7	4	2	0	
8	4	4	0	
9	4	2	30	

	Arrival Delay in Minutes
0	0.0
1	305.0
2	0.0
3	0.0
4	0.0
5	0.0
6	15.0
7	0.0
8	0.0

```
9                      26.0

[10 rows x 22 columns]
```

1.2.4 Display the data type for each column

Next, observe the types of data present within this dataset.

[4]:
```
# Display the data type for each column in your DataFrame.

### YOUR CODE HERE ###

airline_data.dtypes
```

[4]:
```
satisfaction                          object
Customer Type                         object
Age                                    int64
Type of Travel                        object
Class                                 object
Flight Distance                        int64
Seat comfort                           int64
Departure/Arrival time convenient      int64
Food and drink                         int64
Gate location                          int64
Inflight wifi service                  int64
Inflight entertainment                 int64
Online support                         int64
Ease of Online booking                 int64
On-board service                       int64
Leg room service                       int64
Baggage handling                       int64
Checkin service                        int64
Cleanliness                            int64
Online boarding                        int64
Departure Delay in Minutes             int64
Arrival Delay in Minutes             float64
dtype: object
```

Hint 1

Recall the methods for exploring DataFrames.

Hint 2

Recall a property of a `pandas` DataFrame that allows you to view the data type for each column.

Hint 3

Call `.dtypes` on your DataFrame `airline_data` to view the data type of each column.

Question: Identify the target (or predicted) variable for passenger satisfaction. What is your initial hypothesis about which variables will be valuable in predicting satisfaction?

- `satisfaction` represents the classification variable to be predicted.
- Many of these variables seem like meaningful predictors of satisfaction. In particular, delays (either departure or arrival) may be negatively correlated with satisfaction.

1.3 Step 2: Model preparation

Before you proceed with modeling, consider which metrics you will ultimately want to leverage to evaluate your model.

Question: Which metrics are most suited to evaluating this type of model?

- As this is a binary classfication problem, it will be important to evaluate not just accuracy, but the balance of false positives and false negatives that the model's predictions provide. Therefore, precision, recall, and ultimately the F1 score will be excellent metrics to use.
- The ROC AUC (Area Under the Receiver Operating Characteristic) score is also suited to this type of modeling.

1.3.1 Prepare your data for predictions

You may have noticed when previewing your data that there are several non-numerical variables (`object` data types) within the dataset.

To prepare this DataFrame for modeling, first convert these variables into a numerical format.

[5]:
```
# Convert the object predictor variables to numerical dummies.

### YOUR CODE HERE ###

airline_data_dummies = pd.get_dummies(airline_data,
                                      columns=['satisfaction','Customer␣
 ↪Type','Type of Travel','Class'])
```

Hint 1

Refer to the content about feature engineering.

Hint 2

Use the `pandas` function for transforming categorical data into "dummy" variables.

Hint 3

Use the `get_dummies()` function on your DataFrame `airline_data` to create dummies for the categorical variables in your dataset. Note that your target variable will also need this treatment.

1.3.2 Isolate your target and predictor variables

Separately define the target variable (`satisfaction`) and the features.

[6]:
```
# Define the y (target) variable.

### YOUR CODE HERE ###
y = airline_data_dummies['satisfaction_satisfied']

# Define the X (predictor) variables.

### YOUR CODE HERE ###
X = airline_data_dummies.
 ↪drop(['satisfaction_satisfied','satisfaction_dissatisfied'], axis = 1)
```

Hint 1

Refer to the content about splitting your data into x and y.

Hint 2

In `pandas`, use square brackets `[]` to subset your DataFrame by specifying which column(s) to select. Also, quickly subset a DataFrame to exclude a particular column by using the `drop()` function and specifying the column to drop.

Hint 3

In this case, your target variable was split into two columns from the dummy split. Be sure to include only the column which assigns a positive (i.e., "satisfied") outcome as 1.

1.3.3 Divide your data

Divide your data into a training set (75% of the data) and test set (25% of the data). This is an important step in the process, as it allows you to reserve a part of the data that the model has not used to test how well the model generalizes (or performs) on new data.

[7]:
```
# Perform the split operation on your data.
# Assign the outputs as follows: X_train, X_test, y_train, y_test.

### YOUR CODE HERE ###

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,␣
 ↪random_state = 0)
```

Hint 1

Refer to the content about splitting your data between a training and test set.

Hint 2

To perform the splitting, call the function in the `model_selection` module of `sklearn` on the features and target variable.

Hint 3

Call the `train_test_split()` function, passing in both `features` and `target`, while configuring the appropriate `test_size`. Assign the output of this split as `X_train`, `X_test`, `y_train`, `y_test`.

1.4 Step 3: Model building

1.4.1 "Instantiate" your XGBClassifer

Before you fit your model to your airline dataset, first create the XGB Classifier model and define its objective. You'll use this model to fit and score different hyperparameters during the GridSearch cross-validation process.

[8]:
```
# Define xgb to be your XGBClassifier.

### YOUR CODE HERE ###

xgb = XGBClassifier(objective='binary:logistic', random_state=0)
```

Hint 1

Refer to the content about constructing a classifier model from `xgboost`.

Hint 2

Note that the target variable in this case is a binary variable.

Hint 3

Use the `XGBClassifier()` from `xgboost`. Set the objective as `binary:logistic`.

1.4.2 Define the parameters for hyperparameter tuning

To identify suitable parameters for your `xgboost` model, first define the parameters for hyperparameter tuning. Specifically, define a range of values for `max_depth`, `min_child_weight`, `learning_rate`, `n_estimators`, `subsample`, and `colsample_bytree`.

Consider a more limited range for each parameter to allow for timely iteration and model training.

[9]:
```
# Define parameters for tuning as `cv_params`.

### YOUR CODE HERE ###

cv_params = {'max_depth': [4, 6],
             'min_child_weight': [3, 5],
             'learning_rate': [0.1, 0.2, 0.3],
             'n_estimators': [5,10,15],
             'subsample': [0.7],
```

```
              'colsample_bytree': [0.7]
              }
```

Hint 1

Refer to the content about hyperparameter tuning using GridSearch cross-validation.

Hint 2

Consider a range of values for each parameter, similar to what you observed in the lesson.

Hint 3

Define these parameters using a Python dictionary in the following format: `{'parameter1': [range,of,values]}`

Question: What is the likely effect of adding more estimators to your GridSearch?

- More estimators will initially improve the model's performance. However, increasing the number of estimators will also considerably increase the time spent during the GridSearch process, and there will be diminishing returns as the number of estimators continues to increase.

1.4.3 Define how the models will be evaluated

Define how the models will be evaluated for hyperparameter tuning. To yield the best understanding of model performance, utilize a suite of metrics.

[10]:
```
# Define your criteria as `scoring`.

### YOUR CODE HERE ###

scoring = {'accuracy', 'precision', 'recall', 'f1'}
```

Hint 1

Recall what you've learned about using metric evaluation to determine the metrics you include.

Hint 2

Consider what you've learned about the limitations of only including a single metric, such as `accuracy`.

Hint 3

Define metrics which balance the false positives and false negatives in binary classification problems.

1.4.4 Construct the GridSearch cross-validation

Construct the GridSearch cross-validation using the model, parameters, and scoring metrics you defined. Additionally, define the number of folds and specify *which metric* from above will guide the refit strategy.

[11]:
```
# Construct your GridSearch.

### YOUR CODE HERE ###

xgb_cv = GridSearchCV(xgb,
                      cv_params,
                      scoring = scoring,
                      cv = 5,
                      refit = 'f1'
                     )
```

Hint 1

Recall what you've learned about constructing a GridSearch for cross-validation.

Hint 2

Balance the time spent on validation with the number of folds you choose.

Hint 3

Choose the refit method which simultaneously balances false positives and false negatives.

1.4.5 Fit the GridSearch model to your training data

If your GridSearch takes too long, revisit the parameter ranges above and consider narrowing the range and reducing the number of estimators.

Note: The following cell might take several minutes to run.

[12]:
```
%%time
# fit the GridSearch model to training data

### YOUR CODE HERE

xgb_cv = xgb_cv.fit(X_train, y_train)
xgb_cv
```

```
CPU times: user 3min 38s, sys: 3.32 s, total: 3min 42s
Wall time: 1min 57s
```

[12]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None, colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None,
                                     early_stopping_rounds=None,
                                     enable_categorical=False, eval_metric=None,
                                     gamma=None, gpu_id=None, grow_policy=None,
                                     importance_type=None,
```

```
                              interaction_constraints=None,
                              learning_rate=None, max…
                              predictor=None, random_state=0,
                              reg_alpha=None, …),
             iid='deprecated', n_jobs=None,
             param_grid={'colsample_bytree': [0.7],
                         'learning_rate': [0.1, 0.2, 0.3], 'max_depth': [4, 6],
                         'min_child_weight': [3, 5],
                         'n_estimators': [5, 10, 15], 'subsample': [0.7]},
             pre_dispatch='2*n_jobs', refit='f1', return_train_score=False,
             scoring={'f1', 'accuracy', 'precision', 'recall'}, verbose=0)
```

Question: Which optimal set of parameters did the GridSearch yield?

Through accessing the `best_params_` attribute of the fitted GridSearch model, the optimal set of hyperparameters was:

```
{'colsample_bytree': 0.7,  'learning_rate': 0.3,  'max_depth': 6,
'min_child_weight': 5,  'n_estimators': 15,  'subsample': 0.7}
```

Note: Your results may vary from this example response.

Hint 1

Recall what you've learned about the result of the GridSearch.

Hint 2

Once you've fitted the GridSearch model to your training data, there will be an attribute to access which yields to the optimal parameter set.

Hint 3

Access the `best_params_` attribute from your fitted model.

1.4.6 Save your model for reference using `pickle`

Use the `pickle` library you've already imported to save the output of this model.

[13]:
```
# Use `pickle` to save the trained model.

### YOUR CODE HERE ###

pickle.dump(xgb_cv, open('xgb_cv.sav', 'wb'))
```

Hint 1

Refer to the content about "pickling" prior models.

Hint 2

The model to be pickled is the fitted GridSearch model from above.

Hint 3

Call `pickle.dump()`, reference the fitted GridSearch model, and provide a name for the pickle file.

1.5 Step 4: Results and evaluation

1.5.1 Formulate predictions on your test set

To evaluate the predictions yielded from your model, leverage a series of metrics and evaluation techniques from scikit-learn by examining the actual observed values in the test set relative to your model's prediction.

First, use your trained model to formulate predictions on your test set.

[14]:
```
# Apply your model to predict on your test data. Call this output "y_pred".

### YOUR CODE HERE ###

y_pred = xgb_cv.predict(X_test)
```

Hint 1

Recall what you've learned about creating predictions from trained models.

Hint 2

Use the fitted GridSearch model from your training set and predict the predictor variables you reserved in the train-test split.

Hint 3

Call `predict()` on your fitted model and reference `X_test` to create these predictions.

1.5.2 Leverage metrics to evaluate your model's performance

Apply a series of metrics from scikit-learn to assess your model. Specifically, print the accuracy score, precision score, recall score, and f1 score associated with your test data and predicted values.

[15]:
```
# 1. Print your accuracy score.

### YOUR CODE HERE ###
ac_score = metrics.accuracy_score(y_test, y_pred)
print('accuracy score:', ac_score)

# 2. Print your precision score.

### YOUR CODE HERE ###
pc_score = metrics.precision_score(y_test, y_pred)
print('precision score:', pc_score)

# 3. Print your recall score.

```

```
### YOUR CODE HERE ###
rc_score = metrics.recall_score(y_test, y_pred)
print('recall score:', rc_score)

# 4. Print your f1 score.

### YOUR CODE HERE ###
f1_score = metrics.f1_score(y_test, y_pred)
print('f1 score:', f1_score)
```

```
accuracy score: 0.9340314136125655
precision score: 0.9465036952814099
recall score: 0.9327170868347339
f1 score: 0.9395598194130925
```

Hint 1

Refer to the content about model evaluation for detail on these metrics.

Hint 2

Use the function in the `metrics` module in `sklearn` to compute each of these metrics.

Hint 3

Call `accuracy_score()`, `precision_score()`, `recall_score()`, and `f1_score()`, passing `y_test` and `y_pred` into each.

Question: How should you interpret your accuracy score?

The accuracy score for this model is 0.939, or 93.9% accurate.

Question: Is your accuracy score alone sufficient to evaluate your model?

In classification problems, accuracy is useful to know but may not be the best metric to evaluate this model.

Question: When observing the precision and recall scores of your model, how do you interpret these values, and is one more accurate than the other?

Precision and recall scores are both useful to evaluate the correct predictive capability of the model because they balance the false positives and false negatives inherent in prediction. The model shows a precision score of 0.948, suggesting the model is very good at predicting true positives. This means the model correctly predicts whether the airline passenger will be satisfied. The recall score of 0.940 is also very good. This means that the model does a good job of correctly identifying dissatisfied passengers within the dataset. These two metrics combined give a better assessment of model performance than the accuracy metric does alone.

Question: What does your model's F1 score tell you, beyond what the other metrics provide?*

The F1 score balances the precision and recall performance to give a combined assessment of how well this model delivers predictions. In this case, the F1 score is 0.944, which suggests very strong predictive power in this model.

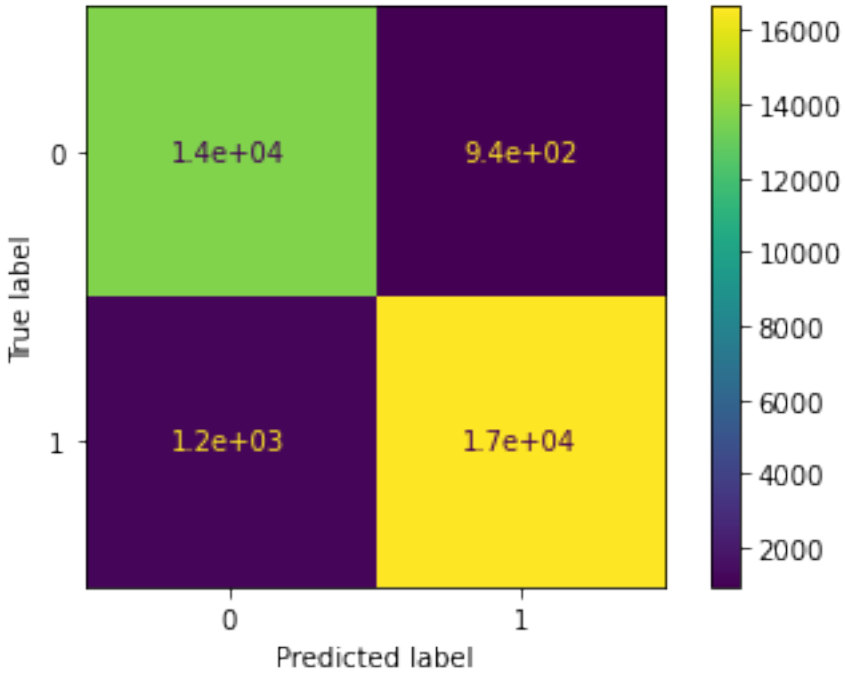
1.5.3 Gain clarity with the confusion matrix

Recall that a **confusion matrix** is a graphic that shows a model's true and false positives and true and false negatives. It helps to create a visual representation of the components feeding into the metrics above.

Create a confusion matrix based on your predicted values for the test set.

[16]:
```
# Construct and display your confusion matrix.

# Construct the confusion matrix for your predicted and test values.

### YOUR CODE HERE ###

cm = metrics.confusion_matrix(y_test, y_pred)

# Create the display for your confusion matrix.

### YOUR CODE HERE ###

disp = metrics.ConfusionMatrixDisplay(confusion_matrix=cm,␣
↪display_labels=xgb_cv.classes_)

# Plot the visual in-line.

### YOUR CODE HERE ###

disp.plot()
```

[16]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f8c0dd82a10>
```



Hint 1

Refer to the content about model evaluation.

Hint 2

Use the functions in the `metrics` module to create a confusion matrix.

Hint 3

Call `confusion_matrix`, passing in `y_test` and `y_pred`. Next, utilize `ConfusionMatrixDisplay()` to display your confusion matrix.

Question: When observing your confusion matrix, what do you notice? Does this correlate to any of your other calculations?

The top left to bottom right diagonal in the confusion matrix represents the correct predictions, and the ratio of these squares showcases the accuracy.

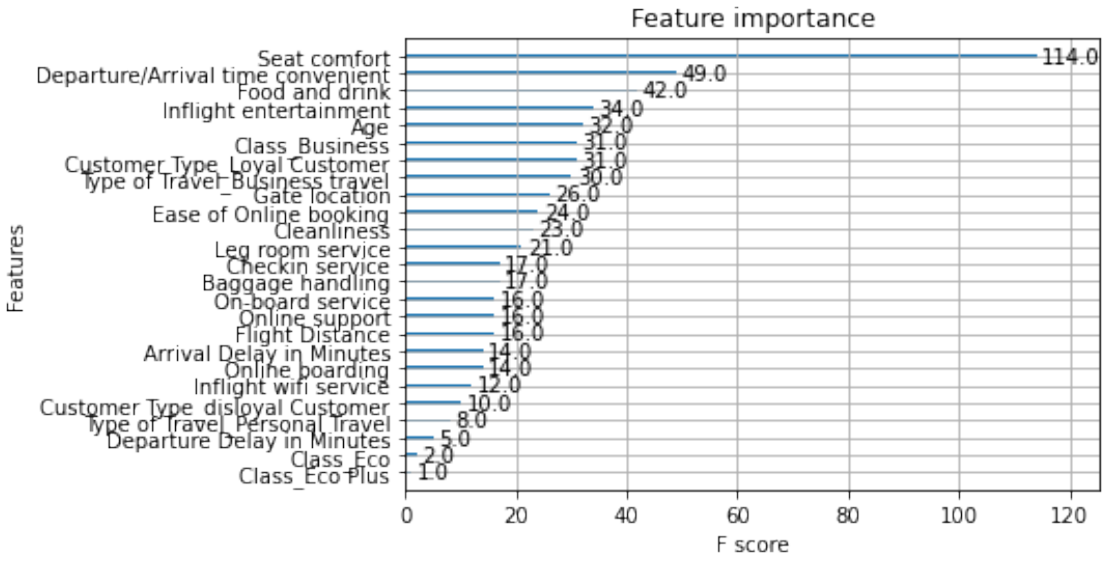
Additionally, the concentration of true positives and true negatives stands out relative to false positives and false negatives, respectively. This ratio is why the precision score is so high (0.944).

1.5.4 Visualize most important features

`xgboost` has a built-in function to visualize the relative importance of the features in the model using `matplotlib`. Output and examine the feature importance of your model.

[17]:
```
# Plot the relative feature importance of the predictor variables in your model.

### YOUR CODE HERE ###

plot_importance(xgb_cv.best_estimator_)
```

[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8c0bfa6ed0>
```



Hint 1

Recall the attributes that are provided once the model is fitted to training data.

Hint 2

Examine the `best_estimator_` attribute of your fitted model.

Hint 3

To easily visualize feature importance, call the built-in `plot_importance` function `xgboost` offers on the `best_estimator_`.

Question: Examine the feature importances outputted above. What is your assessment of the result? Did anything surprise you?

- By a wide margin, "seat comfort" rated as most important in the model.The type of seating is very different between first class and coach seating. However, the perks of being in first class also go beyond the seating type, so perhaps that is an underlying explanation of this feature's importance.
- Surprisingly, delays (both arrival and departure) did not score as highly important.

1.5.5 Compare models

Create a table of results to compare model performance.

[18]:
```
# Create a table of results to compare model performance.

### YOUR CODE HERE ###
table = pd.DataFrame({'Model': ["Tuned Decision Tree", "Tuned Random Forest",␣
 ↪"Tuned XGBoost"],
                      'F1': [0.945422, 0.947306, f1_score],
                      'Recall': [0.935863, 0.944501, rc_score],
                      'Precision': [0.955197, 0.950128, pc_score],
                      'Accuracy': [0.940864, 0.942450, ac_score]
                     }
                    )
table
```

[18]:

	Model	F1	Recall	Precision	Accuracy
0	Tuned Decision Tree	0.945422	0.935863	0.955197	0.940864
1	Tuned Random Forest	0.947306	0.944501	0.950128	0.942450
2	Tuned XGBoost	0.939560	0.932717	0.946504	0.934031

Hint 1

Create a DataFrame using the `pd.DataFrame()` function.

Question: How does this model compare to the decision tree and random forest models you built in previous labs?

Based on the results shown in the table above, the F1, precision, recall, and accuracy scores of the XGBoost model are similar to the corresponding scores of the decision tree and random forest models. The random forest model seemed to outperform the decision tree model as well as the XGBoost model.

1.6 Considerations

What are some key takeaways you learned from this lab? - The evaluation of the model is important to inform if the model has delivered accurate predictions. - Splitting the data is important for ensuring that there is new data for the model to test its predictive performance. - Each metric provides an evaluation from a different standpoint, and accuracy alone is not a strong way to evaluate a model. - Effective assessments balance the true/false positives versus true/false negatives through the confusion matrix and F1 score.

How would you share your findings with your team? - Showcase the data used to create the prediction and the performance of the model overall. - Review the sample output of the features and the confusion matrix to reference the model's performance. - Highlight the metric values, emphasizing the F1 score. - Visualize the feature importance to showcase what drove the model's predictions.

What would you share with and recommend to stakeholders? - The model created is highly effective at predicting passenger satisfaction. - The feature importance of seat comfort warrants additional investigation. It will be important to ask domain experts why they believe this feature scores so highly in this model.

Congratulations! You've completed this lab. However, you may not notice a green check mark next to this item on Coursera's platform. Please continue your progress regardless of the check mark. Just click on the "save" icon at the top of this notebook to ensure your work has been logged