Exemplar_Build a random forest model

January 7, 2024

1 Exemplar: Build a random forest model

1.1 Introduction

As you're learning, random forests are popular statistical learning algorithms. Some of their primary benefits include reducing variance, bias, and the chance of overfitting.

This activity is a continuation of the project you began modeling with decision trees for an airline. Here, you will train, tune, and evaluate a random forest model using data from spreadsheet of survey responses from 129,880 customers. It includes data points such as class, flight distance, and inflight entertainment. Your random forest model will be used to predict whether a customer will be satisfied with their flight experience.

Note: Because this lab uses a real dataset, this notebook first requires exploratory data analysis, data cleaning, and other manipulations to prepare it for modeling.

1.2 Step 1: Imports

Import relevant Python libraries and modules, including `numpy` and `pandas`libraries for data processing; the `pickle` package to save the model; and the `sklearn` library, containing: - The module `ensemble`, which has the function `RandomForestClassifier` - The module `model_selection`, which has the functions `train_test_split`, `PredefinedSplit`, and `GridSearchCV` - The module `metrics`, which has the functions `f1_score`, `precision_score`, `recall_score`, and `accuracy_score`

[1]:
```
# Import `numpy`, `pandas`, `pickle`, and `sklearn`.
# Import the relevant functions from `sklearn.ensemble`, `sklearn.
 ↪model_selection`, and `sklearn.metrics`.

### YOUR CODE HERE ###

import numpy as np
import pandas as pd

import pickle as pkl

from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.model_selection import train_test_split, PredefinedSplit,␣
↪GridSearchCV
from sklearn.metrics import f1_score, precision_score, recall_score,␣
↪accuracy_score
```

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 ###

air_data = pd.read_csv("Invistico_Airline.csv")
```

Hint 1

The `read_csv()` function from the `pandas` library can be helpful here.

Now, you're ready to begin cleaning your data.

1.3 Step 2: Data cleaning

To get a sense of the data, display the first 10 rows.

[3]:
```
# Display first 10 rows.

### YOUR CODE HERE ###

air_data.head(10)
```

[3]:

	satisfaction	Customer Type	Age	Type of Travel	Class	Flight Distance	Seat comfort	Departure/Arrival time convenient
0	satisfied	Loyal Customer	65	Personal Travel	Eco	265	0	0
1	satisfied	Loyal Customer	47	Personal Travel	Business	2464	0	0
2	satisfied	Loyal Customer	15	Personal Travel	Eco	2138	0	0
3	satisfied	Loyal Customer	60	Personal Travel	Eco	623	0	0
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			

```
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]
```

Hint 1

The `head()` function from the `pandas` library can be helpful here.

Now, display the variable names and their data types.

[4]:
```
# Display variable names and types.

### YOUR CODE HERE ###

air_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

DataFrames have an attribute that outputs variable names and data types in one result.

Question: What do you observe about the differences in data types among the variables included in the data?

There are three types of variables included in the data: int64, float64, and object. The object variables are satisfaction, customer type, type of travel, and class.

Next, to understand the size of the dataset, identify the number of rows and the number of columns.

[5]:
```
# Identify the number of rows and the number of columns.

### YOUR CODE HERE ###

air_data.shape
```

[5]:
```
(129880, 22)
```

Hint 1

There is a method in the `pandas` library that outputs the number of rows and the number of columns in one result.

Now, check for missing values in the rows of the data. Start with .isna() to get Booleans indicating whether each value in the data is missing. Then, use .any(axis=1) to get Booleans indicating whether there are any missing values along the columns in each row. Finally, use .sum() to get the number of rows that contain missing values.

[6]:
```
# Get Booleans to find missing values in data.
# Get Booleans to find missing values along columns.
# Get the number of rows that contain missing values.

### YOUR CODE HERE ###

air_data.isna().any(axis=1).sum()
```

[6]:
```
393
```

Question: How many rows of data are missing values?**

There are 393 rows with missing values.

Drop the rows with missing values. This is an important step in data cleaning, as it makes the data more useful for analysis and regression. Then, save the resulting pandas DataFrame in a variable named `air_data_subset`.

[7]:
```
# Drop missing values.
# Save the DataFrame in variable `air_data_subset`.

### YOUR CODE HERE ###

air_data_subset = air_data.dropna(axis=0)
```

Hint 1

The `dropna()` function is helpful here.

Hint 2

The axis parameter passed in to this function should be set to 0 (if you want to drop rows containing missing values) or 1 (if you want to drop columns containing missing values).

Next, display the first 10 rows to examine the data subset.

[8]:
```
# Display the first 10 rows.

### YOUR CODE HERE ###

air_data_subset.head(10)
```

[8]:

	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]
```

Confirm that it does not contain any missing values.

[9]:
```
# Count of missing values.

### YOUR CODE HERE ###

air_data_subset.isna().sum()
```

[9]:
```
satisfaction                          0
Customer Type                         0
Age                                   0
Type of Travel                        0
Class                                 0
Flight Distance                       0
Seat comfort                          0
Departure/Arrival time convenient     0
Food and drink                        0
Gate location                         0
Inflight wifi service                 0
Inflight entertainment                0
Online support                        0
Ease of Online booking                0
On-board service                      0
Leg room service                      0
Baggage handling                      0
Checkin service                       0
Cleanliness                           0
Online boarding                       0
Departure Delay in Minutes            0
Arrival Delay in Minutes              0
dtype: int64
```

Hint 1

You can use the `.isna().sum()` to get the number of missing values for each variable.

Next, convert the categorical features to indicator (one-hot encoded) features.

Note: The `drop_first` argument can be kept as default (`False`) during one-hot encoding for random forest models, so it does not need to be specified. Also, the target variable, `satisfaction`, does not need to be encoded and will be extracted in a later step.

[10]:
```
# Convert categorical features to one-hot encoded features.

### YOUR CODE HERE ###

air_data_subset_dummies = pd.get_dummies(air_data_subset,
                                         columns=['Customer Type','Type of␣
 ↪Travel','Class'])
```

Hint 1

You can use the `pd.get_dummies()` function to convert categorical variables to one-hot encoded variables.

Question: Why is it necessary to convert categorical data into dummy variables?**

It is necessary because the sklearn implementation of `RandomForestClassifier()` requires that categorical features be encoded to numeric, which can be done using dummy variables or one-hot

encoding.

Next, display the first 10 rows to review the `air_data_subset_dummies`.

[11]:
```
# Display the first 10 rows.

### YOUR CODE HERE ###

air_data_subset_dummies.head(10)
```

[11]:

	satisfaction	Age	Flight Distance	Seat comfort	\
0	satisfied	65	265	0	
1	satisfied	47	2464	0	
2	satisfied	15	2138	0	
3	satisfied	60	623	0	
4	satisfied	70	354	0	
5	satisfied	30	1894	0	
6	satisfied	66	227	0	
7	satisfied	10	1812	0	
8	satisfied	56	73	0	
9	satisfied	22	1556	0	

	Departure/Arrival time convenient	Food and drink	Gate location	\
0	0	0	2	
1	0	0	3	
2	0	0	3	
3	0	0	3	
4	0	0	3	
5	0	0	3	
6	0	0	3	
7	0	0	3	
8	0	0	3	
9	0	0	3	

	Inflight wifi service	Inflight entertainment	Online support	…	\
0	2	4	2	…	
1	0	2	2	…	
2	2	0	2	…	
3	3	4	3	…	
4	4	3	4	…	
5	2	0	2	…	
6	2	5	5	…	
7	2	0	2	…	
8	5	3	5	…	
9	2	0	2	…	

	Online boarding	Departure Delay in Minutes	Arrival Delay in Minutes	\
0	2	0	0.0	

```
1                 2                        310                     305.0
2                 2                          0                       0.0
3                 3                          0                       0.0
4                 5                          0                       0.0
5                 2                          0                       0.0
6                 3                         17                      15.0
7                 2                          0                       0.0
8                 4                          0                       0.0
9                 2                         30                      26.0

   Customer Type_Loyal Customer  Customer Type_disloyal Customer  \
0                             1                                 0
1                             1                                 0
2                             1                                 0
3                             1                                 0
4                             1                                 0
5                             1                                 0
6                             1                                 0
7                             1                                 0
8                             1                                 0
9                             1                                 0

   Type of Travel_Business travel  Type of Travel_Personal Travel  \
0                               0                                1
1                               0                                1
2                               0                                1
3                               0                                1
4                               0                                1
5                               0                                1
6                               0                                1
7                               0                                1
8                               0                                1
9                               0                                1

   Class_Business  Class_Eco  Class_Eco Plus
0               0          1               0
1               1          0               0
2               0          1               0
3               0          1               0
4               0          1               0
5               0          1               0
6               0          1               0
7               0          1               0
8               1          0               0
9               0          1               0

[10 rows x 26 columns]
```

Then, check the variables of air_data_subset_dummies.

[12]:
```
# Display variables.

### YOUR CODE HERE ###

air_data_subset_dummies.dtypes
```

[12]:
```
satisfaction                          object
Age                                    int64
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
Customer Type_Loyal Customer           uint8
Customer Type_disloyal Customer        uint8
Type of Travel_Business travel         uint8
Type of Travel_Personal Travel         uint8
Class_Business                         uint8
Class_Eco                              uint8
Class_Eco Plus                         uint8
dtype: object
```

Question: What changes do you observe after converting the string data to dummy variables?**

All of the following changes could be observed:

- Customer Type –> Customer Type_Loyal Customer and Customer Type_disloyal Customer
- Type of Travel –> Type of Travel_Business travel and Type of Travel_Personal travel
- Class –> Class_Business, Class_Eco, Class_Eco Plus

1.4 Step 3: Model building

The first step to building your model is separating the labels (y) from the features (X).

[13]:
```
# Separate the dataset into labels (y) and features (X).

### YOUR CODE HERE ###

y = air_data_subset_dummies["satisfaction"]
X = air_data_subset_dummies.drop("satisfaction", axis=1)
```

Hint 1

Save the labels (the values in the `satisfaction` column) as `y`.

Save the features as `X`.

Hint 2

To obtain the features, drop the `satisfaction` column from the DataFrame.

Once separated, split the data into train, validate, and test sets.

[14]:
```
# Separate into train, validate, test sets.

### YOUR CODE HERE ###

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

Hint 1

Use the `train_test_split()` function twice to create train/validate/test sets, passing in `random_state` for reproducible results.

Hint 1

Split `X`, `y` to get `X_train`, `X_test`, `y_train`, `y_test`. Set the `test_size` argument to the proportion of data points you want to select for testing.

Split `X_train`, `y_train` to get `X_tr`, `X_val`, `y_tr`, `y_val`. Set the `test_size` argument to the proportion of data points you want to select for validation.

1.4.1 Tune the model

Now, fit and tune a random forest model with separate validation set. Begin by determining a set of hyperparameters for tuning the model using GridSearchCV.

[15]:
```
# Determine set of hyperparameters.

### YOUR CODE HERE ###

cv_params = {'n_estimators' : [50,100],
```

```
               'max_depth' : [10,50],
               'min_samples_leaf' : [0.5,1],
               'min_samples_split' : [0.001, 0.01],
               'max_features' : ["sqrt"],
               'max_samples' : [.5,.9]}
```

Hint 1

Create a dictionary `cv_params` that maps each hyperparameter name to a list of values. The GridSearch you conduct will set the hyperparameter to each possible value, as specified, and determine which value is optimal.

Hint 2

The main hyperparameters here include `'n_estimators'`, `'max_depth'`, `'min_samples_leaf'`, `'min_samples_split'`, `'max_features'`, and `'max_samples'`. These will be the keys in the dictionary `cv_params`.

Next, create a list of split indices.

[16]:
```
# Create list of split indices.

### YOUR CODE HERE ###

split_index = [0 if x in X_val.index else -1 for x in X_train.index]
custom_split = PredefinedSplit(split_index)
```

Hint 1

Use list comprehension, iterating over the indices of `X_train`. The list can consists of 0s to indicate data points that should be treated as validation data and -1s to indicate data points that should be treated as training data.

Hint 2

Use `PredfinedSplit()`, passing in `split_index`, saving the output as `custom_split`. This will serve as a custom split that will identify which data points from the train set should be treated as validation data during GridSearch.

Now, instantiate your model.

[17]:
```
# Instantiate model.

### YOUR CODE HERE ###

rf = RandomForestClassifier(random_state=0)
```

Hint 1

Use `RandomForestClassifier()`, specifying the `random_state` argument for reproducible results. This will help you instantiate a random forest model, `rf`.

Next, use GridSearchCV to search over the specified parameters.

[18]:
```
# Search over specified parameters.

### YOUR CODE HERE ###

rf_val = GridSearchCV(rf, cv_params, cv=custom_split, refit='f1', n_jobs = -1,␣
↪verbose = 1)
```

Hint 1

Use `GridSearchCV()`, passing in `rf` and `cv_params` and specifying `cv` as `custom_split`. Additional arguments that you can specify include: `refit='f1', n_jobs = -1, verbose = 1`.

Now, fit your model.

[19]:
```
%%time

# Fit the model.

### YOUR CODE HERE ###


rf_val.fit(X_train, y_train)
```

```
Fitting 1 folds for each of 32 candidates, totalling 32 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  32 out of  32 | elapsed:   40.8s finished

CPU times: user 4.99 s, sys: 87.7 ms, total: 5.08 s
Wall time: 45.5 s
```

[19]:
```
GridSearchCV(cv=PredefinedSplit(test_fold=array([-1, -1, …, -1, -1])),
             error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weig…
                                              n_estimators=100, n_jobs=None,
                                              oob_score=False, random_state=0,
                                              verbose=0, warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [10, 50], 'max_features': ['sqrt'],
```

```
                    'max_samples': [0.5, 0.9],
                    'min_samples_leaf': [0.5, 1],
                    'min_samples_split': [0.001, 0.01],
                    'n_estimators': [50, 100]},
             pre_dispatch='2*n_jobs', refit='f1', return_train_score=False,
             scoring=None, verbose=1)
```

Hint 1

Use the `fit()` method to train the GridSearchCV model on `X_train` and `y_train`.

Hint 2

Add the magic function `%%time` to keep track of the amount of time it takes to fit the model and display this information once execution has completed. Remember that this code must be the first line in the cell.

Finally, obtain the optimal parameters.

[20]:
```
# Obtain optimal parameters.

### YOUR CODE HERE ###

rf_val.best_params_
```

[20]:
```
{'max_depth': 50,
 'max_features': 'sqrt',
 'max_samples': 0.9,
 'min_samples_leaf': 1,
 'min_samples_split': 0.001,
 'n_estimators': 50}
```

Hint 1

Use the `best_params_` attribute to obtain the optimal values for the hyperparameters from the GridSearchCV model.

1.5 Step 4: Results and evaluation

Use the selected model to predict on your test data. Use the optimal parameters found via GridSearchCV.

[21]:
```
# Use optimal parameters on GridSearchCV.

### YOUR CODE HERE ###

rf_opt = RandomForestClassifier(n_estimators = 50, max_depth = 50,
                                min_samples_leaf = 1, min_samples_split = 0.001,
                                max_features="sqrt", max_samples = 0.9,␣
 ↪random_state = 0)
```

Hint 1

Use `RandomForestClassifier()`, specifying the `random_state` argument for reproducible results and passing in the optimal hyperparameters found in the previous step. To distinguish this from the previous random forest model, consider naming this variable `rf_opt`.

Once again, fit the optimal model.

[22]:
```
# Fit the optimal model.

### YOUR CODE HERE ###

rf_opt.fit(X_train, y_train)
```

[22]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=50, max_features='sqrt',
                       max_leaf_nodes=None, max_samples=0.9,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=0.001,
                       min_weight_fraction_leaf=0.0, n_estimators=50,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

Hint 1

Use the `fit()` method to train `rf_opt` on `X_train` and `y_train`.

And predict on the test set using the optimal model.

[23]:
```
# Predict on test set.

### YOUR CODE HERE ###

y_pred = rf_opt.predict(X_test)
```

Hint 1

You can call the `predict()` function to make predictions on `X_test` using `rf_opt`. Save the predictions now (for example, as `y_pred`), to use them later for comparing to the true labels.

1.5.1 Obtain performance scores

First, get your precision score.

[24]:
```
# Get precision score.

### YOUR CODE HERE ###

pc_test = precision_score(y_test, y_pred, pos_label = "satisfied")
print("The precision score is {pc:.3f}".format(pc = pc_test))
```

```
The precision score is 0.950
```

Hint 1

You can call the `precision_score()` function from `sklearn.metrics`, passing in `y_test` and `y_pred` and specifying the `pos_label` argument as `"satisfied"`.

Then, collect the recall score.

[25]:
```
# Get recall score.

### YOUR CODE HERE ###

rc_test = recall_score(y_test, y_pred, pos_label = "satisfied")
print("The recall score is {rc:.3f}".format(rc = rc_test))
```

```
The recall score is 0.945
```

Hint 1

You can call the `recall_score()` function from `sklearn.metrics`, passing in `y_test` and `y_pred` and specifying the `pos_label` argument as `"satisfied"`.

Next, obtain your accuracy score.

[26]:
```
# Get accuracy score.

### YOUR CODE HERE ###

ac_test = accuracy_score(y_test, y_pred)
print("The accuracy score is {ac:.3f}".format(ac = ac_test))
```

```
The accuracy score is 0.942
```

Hint 1

You can call the `accuracy_score()` function from `sklearn.metrics`, passing in `y_test` and `y_pred` and specifying the `pos_label` argument as `"satisfied"`.

Finally, collect your F1-score.

[27]:
```
# Get F1 score.

### YOUR CODE HERE ###

f1_test = f1_score(y_test, y_pred, pos_label = "satisfied")
print("The F1 score is {f1:.3f}".format(f1 = f1_test))
```

```
The F1 score is 0.947
```

Hint 1

You can call the `f1_score()` function from `sklearn.metrics`, passing in `y_test` and `y_pred` and specifying the `pos_label` argument as `"satisfied"`.

Question: How is the F1-score calculated?

F1 scores are calculated using the following formula:

F1 = 2 * (precision * recall) / (precision + recall)

Question: What are the pros and cons of performing the model selection using test data instead of a separate validation dataset?

Pros: * The coding workload is reduced. * The scripts for data splitting are shorter. * It's only necessary to evaluate test dataset performance once, instead of two evaluations (validate and test).

Cons: * If a model is evaluated using samples that were also used to build or fine-tune that model, it likely will provide a biased evaluation. * A potential overfitting issue could happen when fitting the model's scores on the test data.

1.5.2 Evaluate the model

Now that you have results, evaluate the model.

Question: What are the four basic parameters for evaluating the performance of a classification model?

1. True positives (TP): These are correctly predicted positive values, which means the value of actual and predicted classes are positive.
2. True negatives (TN): These are correctly predicted negative values, which means the value of the actual and predicted classes are negative.
3. False positives (FP): This occurs when the value of the actual class is negative and the value of the predicted class is positive.
4. False negatives (FN): This occurs when the value of the actual class is positive and the value of the predicted class in negative.

Reminder: When fitting and tuning classification modeld, data professioals aim to minimize false positives and false negatives.

Question: What do the four scores demonstrate about your model, and how do you calculate them?

- Accuracy (TP+TN/TP+FP+FN+TN): The ratio of correctly predicted observations to total observations.
- Precision (TP/TP+FP): The ratio of correctly predicted positive observations to total predicted positive observations.
- Recall (Sensitivity, TP/TP+FN): The ratio of correctly predicted positive observations to all observations in actual class.
- F1 score: The harmonic average of precision and recall, which takes into account both false positives and false negatives.

Calculate the scores: precision score, recall score, accuracy score, F1 score.

[28]:
```
# Precision score on test data set.

### YOUR CODE HERE ###

print("\nThe precision score is: {pc:.3f}".format(pc = pc_test), "for the test␣
 ↪set,", "\nwhich means of all positive predictions,", "{pc_pct:.1f}%␣
 ↪prediction are true positive.".format(pc_pct = pc_test * 100))
```

```
The precision score is: 0.950 for the test set,
which means of all positive predictions, 95.0% prediction are true positive.
```

[29]:
```
# Recall score on test data set.

### YOUR CODE HERE ###

print("\nThe recall score is: {rc:.3f}".format(rc = rc_test), "for the test␣
 ↪set,", "\nwhich means of which means of all real positive cases in test␣
 ↪set,", "{rc_pct:.1f}% are  predicted positive.".format(rc_pct = rc_test *␣
 ↪100))
```

```
The recall score is: 0.945 for the test set,
which means of which means of all real positive cases in test set, 94.5% are
predicted positive.
```

[30]:
```
# Accuracy score on test data set.

### YOUR CODE HERE ###

print("\nThe accuracy score is: {ac:.3f}".format(ac = ac_test), "for the test␣
 ↪set,", "\nwhich means of all cases in test set,", "{ac_pct:.1f}% are␣
 ↪predicted true positive or true negative.".format(ac_pct = ac_test * 100))
```

```
The accuracy score is: 0.942 for the test set,
which means of all cases in test set, 94.2% are predicted true positive or true
negative.
```

[31]:
```
# F1 score on test data set.

### YOUR CODE HERE ###

print("\nThe F1 score is: {f1:.3f}".format(f1 = f1_test), "for the test set,",␣
 ↪"\nwhich means the test set's harmonic mean is {f1_pct:.1f}%.".format(f1_pct␣
 ↪= f1_test * 100))
```

```
The F1 score is: 0.947 for the test set,
which means the test set's harmonic mean is 94.7%.
```

Question: How does this model perform based on the four scores?

The model performs well according to all 4 performance metrics. The model's precision score is slightly better than the 3 other metrics.

1.5.3 Evaluate the model

Finally, create a table of results that you can use to evaluate the performace of your model.

[32]:
```
# Create table of results.

### YOUR CODE HERE ###
table = pd.DataFrame({'Model': ["Tuned Decision Tree", "Tuned Random Forest"],
                      'F1':  [0.945422, f1_test],
                      'Recall': [0.935863, rc_test],
                      'Precision': [0.955197, pc_test],
                      'Accuracy': [0.940864, ac_test]
                     }
                    )
table
```

[32]:

	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

Hint 1

Build a table to compare the performance of the models. Create a DataFrame using the `pd.DataFrame()` function.

Question: How does the random forest model compare to the decision tree model you built in the previous lab?

The tuned random forest has higher scores overall, so it is the better model. Particularly, it shows a better F1 score than the decision tree model, which indicates that the random forest model may do better at classification when taking into account false positives and false negatives.

1.6 Considerations

What are the key takeaways from this lab? - Data exploring, cleaning, and encoding are necessary for model building. - A separate validation set is typically used for tuning a model, rather than using the test set. This also helps avoid the evaluation becoming biased. - F1 scores are usually more useful than accuracy scores. If the cost of false positives and false negatives are very different, it's better to use the F1 score and combine the information from precision and recall. * The random forest model yields a more effective performance than a decision tree model.

What summary would you provide to stakeholders? * The random forest model predicted satisfaction with more than 94.2% accuracy. The precision is over 95% and the recall is approximately 94.5%. * The random forest model outperformed the tuned decision tree with the best hyperparameters in most of the four scores. This indicates that the random forest model may perform better. * Because stakeholders were interested in learning about the factors that are most important to customer satisfaction, this would be shared based on the tuned random forest. * In addition, you would provide details about the precision, recall, accuracy, and F1 scores to support your findings.

1.6.1 References

What is the Difference Between Test and Validation Datasets?, Jason Brownlee

Decision Trees and Random Forests Neil Liberman

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