Activity_Build a random forest model

December 30, 2023

1 Activity: 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`

[2]:
```
# Import `numpy`, `pandas`, `pickle`, and `sklearn`.
import numpy as np
import pandas as pd
import pickle as pickle

# Import the relevant functions from `sklearn.ensemble`, `sklearn.
↪model_selection`, and `sklearn.metrics`.
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import PredefinedSplit
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
```

```
from sklearn.metrics import recall_score
from sklearn.metrics import 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.

[37]:
```
# 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.

[5]:
```
# Display first 10 rows.

air_data.head(10)
```

[5]:

	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	354	0	0
5	satisfied	Loyal Customer	30	Personal Travel	Eco	1894	0	0
6	satisfied	Loyal Customer	66	Personal Travel	Eco	227	0	0
7	satisfied	Loyal Customer	10	Personal Travel	Eco	1812	0	0
8	satisfied	Loyal Customer	56	Personal Travel	Business			
9	satisfied	Loyal Customer	22	Personal Travel	Eco			

```
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.

[7]:
```
# Display variable names and types.

air_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 129880 entries, 0 to 129879
Data columns (total 22 columns):
 #   Column                            Non-Null Count   Dtype
---  ------                            --------------   -----
 0   satisfaction                      129880 non-null  object
 1   Customer Type                     129880 non-null  object
 2   Age                               129880 non-null  int64
 3   Type of Travel                    129880 non-null  object
 4   Class                             129880 non-null  object
 5   Flight Distance                   129880 non-null  int64
 6   Seat comfort                      129880 non-null  int64
 7   Departure/Arrival time convenient 129880 non-null  int64
 8   Food and drink                    129880 non-null  int64
 9   Gate location                     129880 non-null  int64
 10  Inflight wifi service             129880 non-null  int64
 11  Inflight entertainment            129880 non-null  int64
 12  Online support                    129880 non-null  int64
 13  Ease of Online booking            129880 non-null  int64
 14  On-board service                  129880 non-null  int64
 15  Leg room service                  129880 non-null  int64
 16  Baggage handling                  129880 non-null  int64
 17  Checkin service                   129880 non-null  int64
 18  Cleanliness                       129880 non-null  int64
 19  Online boarding                   129880 non-null  int64
 20  Departure Delay in Minutes        129880 non-null  int64
 21  Arrival Delay in Minutes          129487 non-null  float64
dtypes: float64(1), int64(17), object(4)
memory usage: 21.8+ MB
```

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 4 objects and the rest are numerical.

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

[4]:
```
# Identify the number of rows and the number of columns.

air_data.shape
```

[4]:
```
(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.

[38]:
```
# 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.

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

[38]:
```
393
```

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

There are 393 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`.

[39]:
```
# Drop missing values.
# Save the DataFrame in variable `air_data_subset`.

air_data_subset = air_data.dropna(axis=1)
```

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.

[9]:
```
# Display the first 10 rows.

air_data_subset.head(10)
```

[9]:

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

	Ease of Online booking	On-board service	Leg room service	\
0	3	3	0	
1	3	4	4	
2	2	3	3	
3	1	1	0	
4	2	2	0	
5	2	5	4	

```
6                      5                 5                 0
7                      2                 3                 3
8                      4                 4                 0
9                      2                 2                 4

   Baggage handling  Checkin service  Cleanliness  Online boarding  \
0                 3                5            3                2
1                 4                2            3                2
2                 4                4            4                2
3                 1                4            1                3
4                 2                4            2                5
5                 5                5            4                2
6                 5                5            5                3
7                 4                5            4                2
8                 1                5            4                4
9                 5                3            4                2

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

[10 rows x 21 columns]
```

Confirm that it does not contain any missing values.

[40]:
```
# Count of missing values.

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

[40]:
```
0
```

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.

[41]:
```
# Convert categorical features to one-hot encoded features.

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's is necessary because building decision trees strictly only accepts numerical data.

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

[42]:
```
# Display the first 10 rows.

air_data_subset_dummies.head(10)
```

[42]:

	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  …

   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

   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 25 columns]
```

Then, check the variables of air_data_subset_dummies.

[43]:
```
# Display variables.

air_data_subset_dummies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 129880 entries, 0 to 129879
Data columns (total 25 columns):
 #   Column                                Non-Null Count   Dtype
---  ------                                --------------   -----
 0   satisfaction                          129880 non-null  object
 1   Age                                   129880 non-null  int64
 2   Flight Distance                       129880 non-null  int64
 3   Seat comfort                          129880 non-null  int64
 4   Departure/Arrival time convenient     129880 non-null  int64
 5   Food and drink                        129880 non-null  int64
 6   Gate location                         129880 non-null  int64
 7   Inflight wifi service                 129880 non-null  int64
 8   Inflight entertainment                129880 non-null  int64
 9   Online support                        129880 non-null  int64
 10  Ease of Online booking                129880 non-null  int64
 11  On-board service                      129880 non-null  int64
 12  Leg room service                      129880 non-null  int64
 13  Baggage handling                      129880 non-null  int64
 14  Checkin service                       129880 non-null  int64
 15  Cleanliness                           129880 non-null  int64
 16  Online boarding                       129880 non-null  int64
 17  Departure Delay in Minutes            129880 non-null  int64
 18  Customer Type_Loyal Customer          129880 non-null  uint8
 19  Customer Type_disloyal Customer       129880 non-null  uint8
 20  Type of Travel_Business travel        129880 non-null  uint8
 21  Type of Travel_Personal Travel        129880 non-null  uint8
 22  Class_Business                        129880 non-null  uint8
 23  Class_Eco                             129880 non-null  uint8
 24  Class_Eco Plus                        129880 non-null  uint8
dtypes: int64(17), object(1), uint8(7)
memory usage: 18.7+ MB
```

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

All variables are now numerical within the entire set.

1.4 Step 3: Model building

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

[44]:
```
# Separate the dataset into labels (y) and features (X).

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.

[63]:
```
# Separate into train, validate, test sets.
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.

[64]:
```
# Determine set of hyperparameters.
```

```
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.

[65]:
```
# Create list of split indices.

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.

[66]:
```
# Instantiate model.

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.

[67]:
```
# Search over specified parameters.

```

```
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.

[68]:
```
%%time
# Fit the model.

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:   32.1s finished

CPU times: user 7.22 s, sys: 35.1 ms, total: 7.25 s
Wall time: 39 s
```

[68]:
```
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.

[69]:
```
# Obtain optimal parameters.

rf_val.best_params_
```

[69]:
```
{'max_depth': 50,
 'max_features': 'sqrt',
 'max_samples': 0.9,
 'min_samples_leaf': 1,
 'min_samples_split': 0.001,
 'n_estimators': 100}
```

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.

[70]:
```
# Use optimal parameters on GridSearchCV.

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.

[71]:
```
# Fit the optimal model.

rf_opt.fit(X_train, y_train)
```

[71]:
```
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.

[72]:
```
# Predict on test set.

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.

[73]:
```
# Get precision score.

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.

[74]:
```
# Get recall score.

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.

[76]:
```
# Get accuracy score.

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

```
The accuracy score is 0.943
```

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.

[77]:
```
# Get F1 score.

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 0.948
```

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?

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: It cuts down workload, the scripts themselves are shoter and simpler to write.

cons: It can be easily over fit and opens possibilities for biased results.

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?

True Positives, True Negatives, False Positives, 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 Precision: TP/TP+FP Recall: Sensitivity, TP/TP+FN F1 Score: Is the average of precision and recall with both FP an FN

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

[78]:
```
# Precision score on test data set.

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.
```

[79]:
```
# Recall score on test data set.

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.
```

[80]:
```
# Accuracy score on test data set.

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: 0.943 for the test set,
which means of all cases in test set, 94.3% are predicted true positive or true
negative.
```

[81]:
```
# F1 score on test data set.

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: 0.948 for the test set,
which means the test set's harmonic mean is 94.8%.
```

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

This model performs well all 4 scores are over 90%.

1.5.3 Evaluate the model

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

[82]:
```
# Create table of results.

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
```

[82]:

	Model	F1	Recall	Precision	Accuracy
0	Tuned Decision Tree	0.945422	0.935863	0.955197	0.940864
1	Tuned Random Forest	0.947809	0.945154	0.950479	0.942778

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?

This model scores a lot better than the previous model I built in the last Notebook. This is a very clear better option between both models.

1.6 Considerations

What are the key takeaways from this lab? Consider important steps when building a model, most effective approaches and tools, and overall results.

Building a full random forest with multiple trees can be much better with overall modeling when the proper amount of data is provided. This method returns very promising results more so than a single tree.

What summary would you provide to stakeholders?

This model predicts with 94% accuracy, this would be a very great option to predict customer satisfaction for future flyers and even returning flyers.

1.6.1 References

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

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