Decision Trees - Decision Trees - 3

One should look for what is and not what he thinks should be. (Albert Einstein)

Quick review

Type the answer in the chat box.

- Which of the following is NOT true about Decision Trees?
  i. Decision Trees are used for classification and regression analysis
  ii. Decision Trees are a useful method for mitigating bias
  iii. Decision Tree models cope well with heterogeneous data, missing data, and nonlinear effects
  iv. Decision Tree algorithm is a supervised machine learning method

Quick review

The correct answer is **2**

- Which of the following is NOT true about decision trees?
  i. Decision Trees are are used for classification and regression analysis
  ii. **Decision Trees are a useful method for mitigating bias**
  iii. Decision Tree models cope well with heterogeneous data, missing data, and nonlinear effects
  iv. Decision Tree algorithm is a supervised machine learning method

Module completion checklist

Objectives	Complete
Evaluate the model and store final results	
Implement Decision Tree on the entire dataset and evaluate its results	

Evaluate the model

- We can use the following metrics to measure how well our simple decision tree is doing
  - Accuracy score
  - Confusion matrix
  - AUC score
  - ROC

Evaluate the model (cont'd)

- Let's calculate the confusion matrix:

```
# Confusion matrix for first model.
cm_tree = confusion_matrix(y_test,y_predict)
```

- We used a confusion matrix to calculate accuracy, misclassification rate, true positive rate, false positive rate, and specificity
- We won't go through all of the metrics right now, but let's **calculate accuracy because it's a metric used frequently to compare classification models**

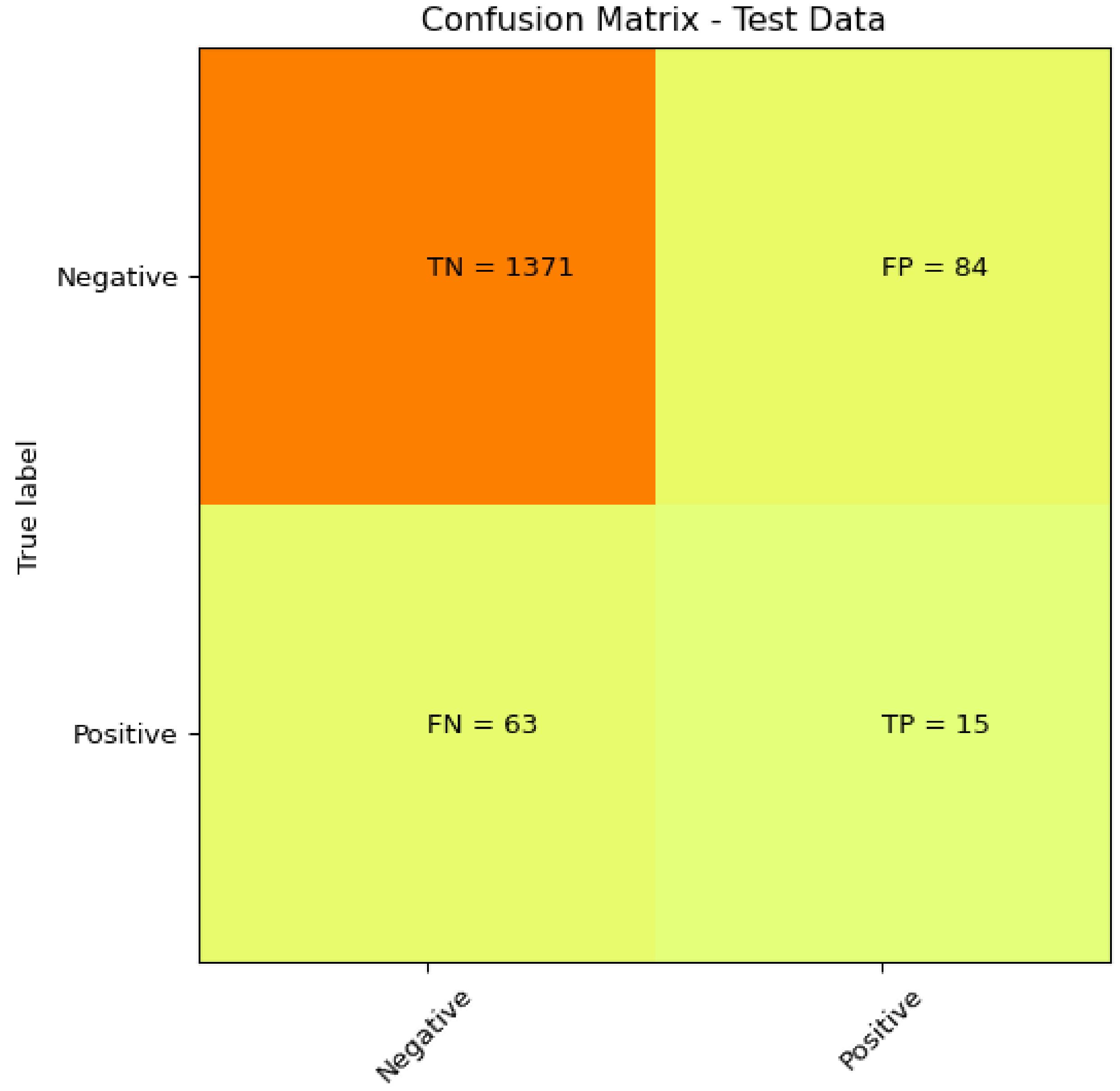
```
# Accuracy score.
acc_score = accuracy_score(y_test, y_predict)
print(acc_score)
```

```
0.9041095890410958
```

Plot confusion matrix

- Let's plot our confusion matrix

```
plt.clf()
plt.imshow(cm_tree, interpolation='nearest',
cmap=plt.cm.Wistia)
classNames = ['Negative','Positive']
plt.title('Confusion Matrix - Test Data')
plt.ylabel('True label')
plt.xlabel('Predicted label')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j,i, str(s[i][j]) + " = " +
str(cm_tree[i][j]))
plt.show()
```



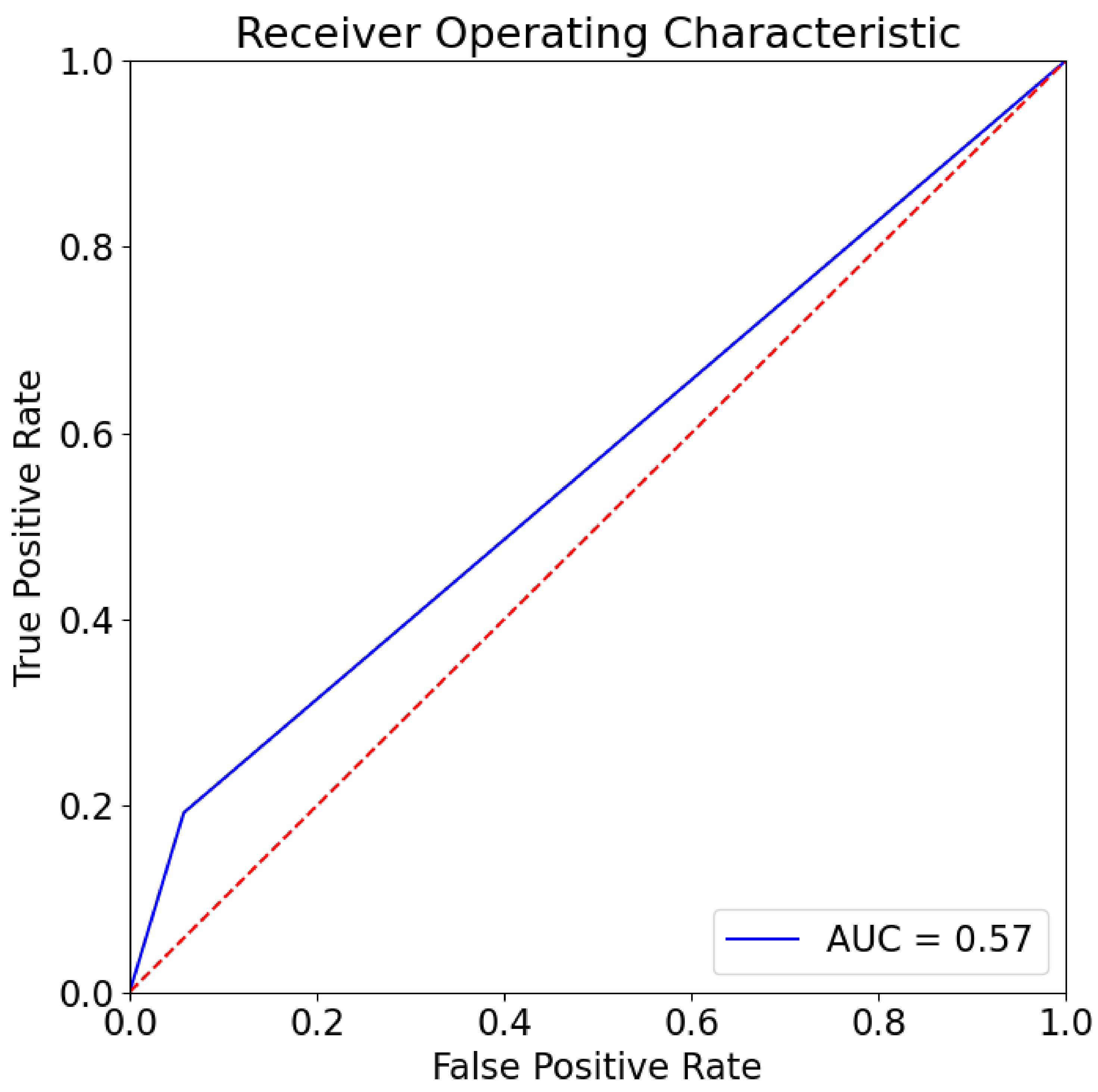
Plot ROC and calculate AUC

- Finally, let's plot our ROC curve and calculate AUC

```
# Calculate metrics for ROC (fpr, tpr) and
calculate AUC.
fpr, tpr, threshold = metrics.roc_curve(y_test,
y_predict)
roc_auc = metrics.auc(fpr, tpr)

# Plot ROC.
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' %
roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

- Our AUC score is `0.58`, which indicates that the performance of our model is not that great



Module completion checklist

Objectives	Complete
Evaluate the model and store final results	✔
Implement Decision Tree on the entire dataset and evaluate its results	

Decision Tree: build

- Let's build our Decision Tree and use all default parameters for now as our baseline model

```
# Set up logistic regression model.
clf = tree.DecisionTreeClassifier()
print(clf)
```

```
DecisionTreeClassifier()
```

- We can see that the default model contains many parameters, including:
  - `max_depth = None`
  - `min_samples_split = 2`
  - `min_samples_leaf = 1`
  - `max_features = None`

Decision Tree: fit

- Let's fit the Decision Tree with `X_train` and `y_train`
- We will run the model on our training data and predict on test data

```
# Fit the model.
clf_fit = clf.fit(X_train, y_train)
```

Decision Tree: predict

- We will predict on the test data using our trained model
- The result is a **vector of the predictions**

```
# Predict on X_test.
y_predict = clf_fit.predict(X_test)
print(y_predict[:20])
```

```
[False False False False False False False False False False False False
 False False False False False False False False]
```

Decision Tree: accuracy score

- Let's calculate the accuracy score of our tree and save it to our `model_final` dataframe

```
# Compute test model accuracy score.
tree_accuracy_score = metrics.accuracy_score(y_test, y_predict)
print("Accuracy on test data: ", tree_accuracy_score)
```

```
Accuracy on test data:  0.9047619047619048
```

- **Is this result accurate?**
- The high accuracy could be due to a multitude of reasons:
  - the classifier could be overfitting the dataset
  - the tree could be biased to classes which have a majority in the dataset
  - the train set and test set could be very similar

Decision Tree: train accuracy

- Let's find out the accuracy on the training data to be sure

```
# Compute accuracy using training data.
acc_train_tree = clf_fit.score(X_train,
                                y_train)
print ("Train Accuracy:", acc_train_tree)
```

```
Train Accuracy: 1.0
```

Decision Tree: accuracy

- Save the accuracy score to our `model_final` if you want to use it later.

```
# Save this model to use later if needed
model_final_tree = {'metrics' : "accuracy" ,
                                 'values' : round(tree_accuracy_score,4),
                                 'model':'tree_all_variables' }
print(model_final_tree)
```

```
{'metrics': 'accuracy', 'values': 0.9048, 'model': 'tree_all_variables'}
```

- Let’s run a **10-fold cross-validation** to see if the results are accurate

Introducing cross-validation

- Before applying any machine learning algorithms on the data, we usually need to split the data into a **train set** and a **test set**
- But now, we are doing this **multiple times**
- We have a new **test set** for each fold `n`
- The rest of the data is the **train set**

Why do we use cross-validation?

- Cross-validation is helpful in multiple ways:
  - It tunes our model better by running it multiple times on our data (instead of just once on the train set and once on the test set)
  - You get assurance that your model has most of the patterns from the data correct and it's not picking up too much on the noise
  - It finds optimal parameters for your model because it runs multiple times

Cross-validation: train and test

Train

- This is the data that you train your model on
- Use a larger portion of the data to train so that the model gets a large enough sample of the population
- Usually about **70%** of your dataset
- **When there is not a large population to begin with, cross-validation techniques can be implemented**

Test

- This is the data that you test your model on
- Use a smaller portion to test your trained model on
- Usually about **30%** of your dataset
- **When cross-validation is implemented, small test sets will be held out multiple times**

Cross-validation: n-fold

Here is how cross-validation works:

1. Split the dataset into several subsets (“n” number of subsets) of equal size
2. **Use each subset as the test dataset** and **use the rest of the data as the training dataset**
3. Repeat the process for every subset you create

Test

Train

Data	x	y	z
1	...	...	...
2	...	...	...
3	...	...	...
4	...	...	...
5	...	...	...
6	...	...	...

Data	x	y	z
1	...	...	...
2	...	...	...
3	...	...	...
4	...	...	...
5	...	...	...
6	...	...	...

Data	x	y	z
1	...	...	...
2	...	...	...
3	...	...	...
4	...	...	...
5	...	...	...
6	...	...	...

Cross-validation

- The input is an estimator, X, y and the number of folds for cross-validation
- It returns an **array of scores of the estimator for each run of the cross-validation**

sklearn.model_selection.cross_val_score

`sklearn.model_selection.` **`cross_val_score`** (*estimator, X, y=None, groups=None, scoring=None, cv='warn', n_jobs=None, verbose=0, fit_params=None, pre_dispatch='2*n_jobs', error_score='raise-deprecating'*) [source]

Evaluate a score by cross-validation

Read more in the User Guide.

Parameters:

estimator : *estimator object implementing 'fit'*
The object to use to fit the data.

X : *array-like*
The data to fit. Can be for example a list, or an array.

y : *array-like, optional, default: None*
The target variable to try to predict in the case of supervised learning.

groups : *array-like, with shape (n_samples,), optional*
Group labels for the samples used while splitting the dataset into train/test set.

scoring : *string, callable or None, optional, default: None*
A string (see model evaluation documentation) or a scorer callable object / function with signature `scorer(estimator, X, y)` which should return only a single value.

Similar to `cross_validate` but only a single metric is permitted.

If None, the estimator's default scorer (if available) is used.

cv : *int, cross-validation generator or an iterable, optional*
Determines the cross-validation splitting strategy. Possible inputs for cv are:

- None, to use the default 3-fold cross validation,
- integer, to specify the number of folds in a `(Stratified)KFold`,
- CV splitter,
- An iterable yielding (train, test) splits as arrays of indices.

Cross-validation scores

```
clf = tree.DecisionTreeClassifier()
cv_scores = cross_val_score(clf, X, y, cv = 10)
```

```
# Print each cv score (accuracy) and average them.
print(cv_scores)
```

```
[0.91389432 0.90019569 0.90410959 0.90998043 0.90998043 0.90802348
 0.90410959 0.90998043 0.91389432 0.91976517]
```

```
print("cv_scores mean:{}".format(np.mean(cv_scores)))
```

```
cv_scores mean:0.9093933463796479
```

```
mean = np.mean(cv_scores)
print("Optimal cv score is:", round(mean, 4))
```

```
Optimal cv score is: 0.9094
```

- There's a big difference in the vanilla model results and cross-validation results
- We can now try pruning and optimizing the tree by finding optimal parameters for our model

Knowledge check

Module completion checklist

Objectives	Complete
Evaluate the model and store final results	✔
Implement decision tree on the entire dataset and evaluate its results	✔

Congratulations on completing this module!

You are now ready to try Task 7 in the Exercise for this topic