Decision Trees - Decision Trees - 4

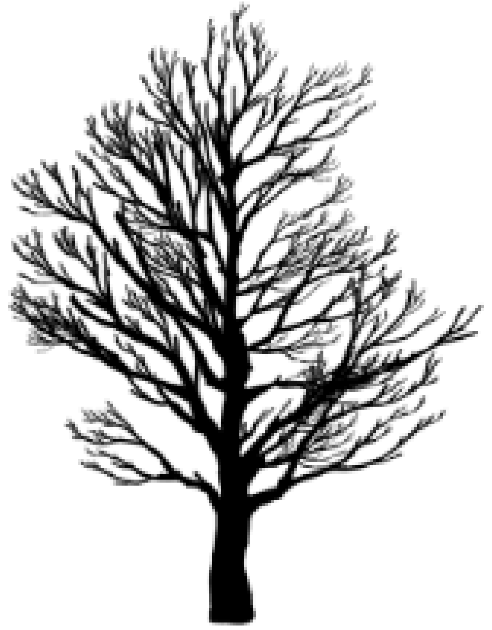
One should look for what is and not what he thinks should be. (Albert Einstein)

Module completion checklist

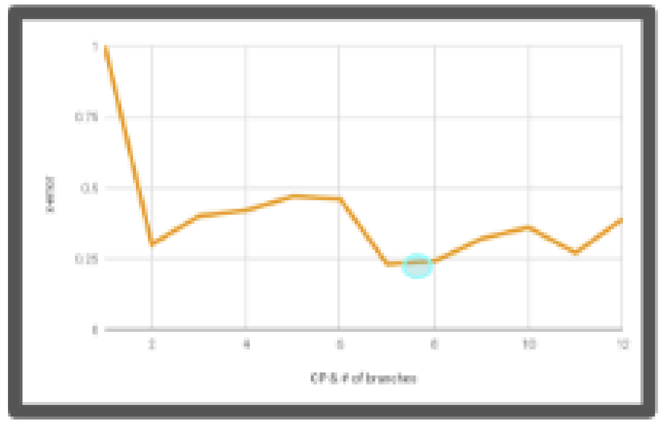
Objectives	Complete
Optimize the Decision Tree by tuning the hyperparameters	
Run the optimized model, predict, and evaluate the new model	

Decision Trees: process

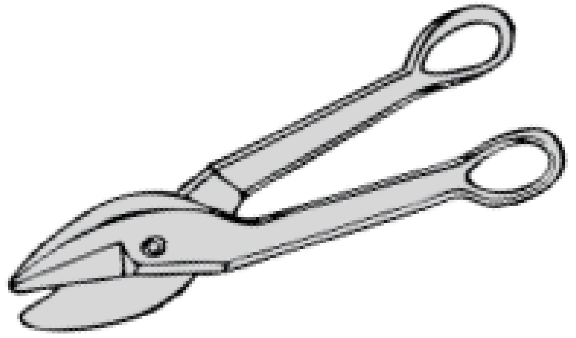
Step 1:
Grow tree on training data



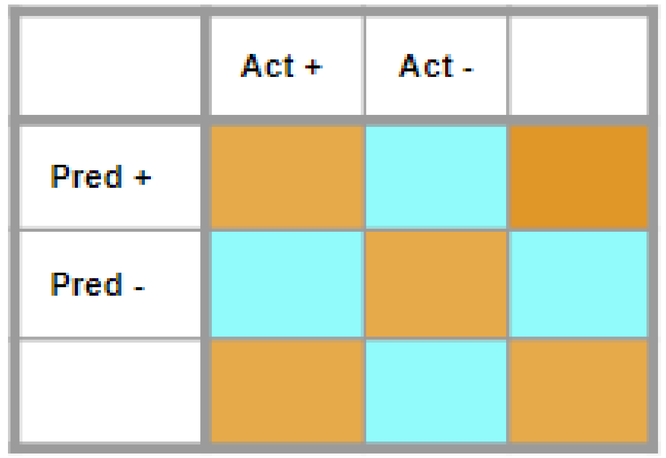
Step 2:
Examine Model output



Step 3:
Prune Tree



Step 4:
Check performance on test data



SO far we have grown our tree and made several choices to get to step 4. We must now continue to improve the tree through optimization.

Ways to optimize a Decision Tree

- Building a Decision Tree is affected by several parameters, as seen in the tree we built
- All the values of our original tree are set to the `tree` defaults within `sklearn`
- We are now going to optimize the tree **focusing on the four parameters we called out**
  - `max_depth = None`
  - `min_samples_split = 2`
  - `min_samples_leaf = 1`
  - `max_features = None`

Define an optimal number function

- Before we optimize individual parameters, let's build a function that will help us store the parameters we will be using in our `optimized_tree`
- The input is:
  - `values` : list of values for given parameter that we iterate through
  - `test_results` : predictions on test set for each parameter that we iterate over
- The output is:
  - `best_value` : the actual parameter value that performs the best and that we will use in our final optimized tree

```
# Define function that will determine the optimal number for each parameter.
def optimal_parameter(values,test_results):
    best_test_value = max(test_results)
    best_test_index = test_results.index(best_test_value)
    best_value = values[best_test_index]
    return(best_value)
```

Optimize: max depth

- `max_depth` indicates how deep the tree can be
- **The deeper the tree, the more splits it has and captures more information about the data**
- **But remember, there is a fine line between a well-fit model and an *overfit* model**
- In our original model, `max_depth = None`
- Now, we're going to fit a Decision Tree with depths ranging from 1 to 32 and plot the training and test accuracy

Optimize: max depth

```
# Max depth:
max_depths = range(1, 33)
train_results = []
test_results = []

for max_depth in max_depths:
    dt = DecisionTreeClassifier(max_depth=max_depth)
    dt.fit(X_train, y_train)

    train_pred = dt.predict(X_train)
    acc_train = accuracy_score(y_train, train_pred)
    train_results.append(acc_train)

    y_pred = dt.predict(X_test);
    acc_test = accuracy_score(y_test, y_pred)
    test_results.append(acc_test);
```

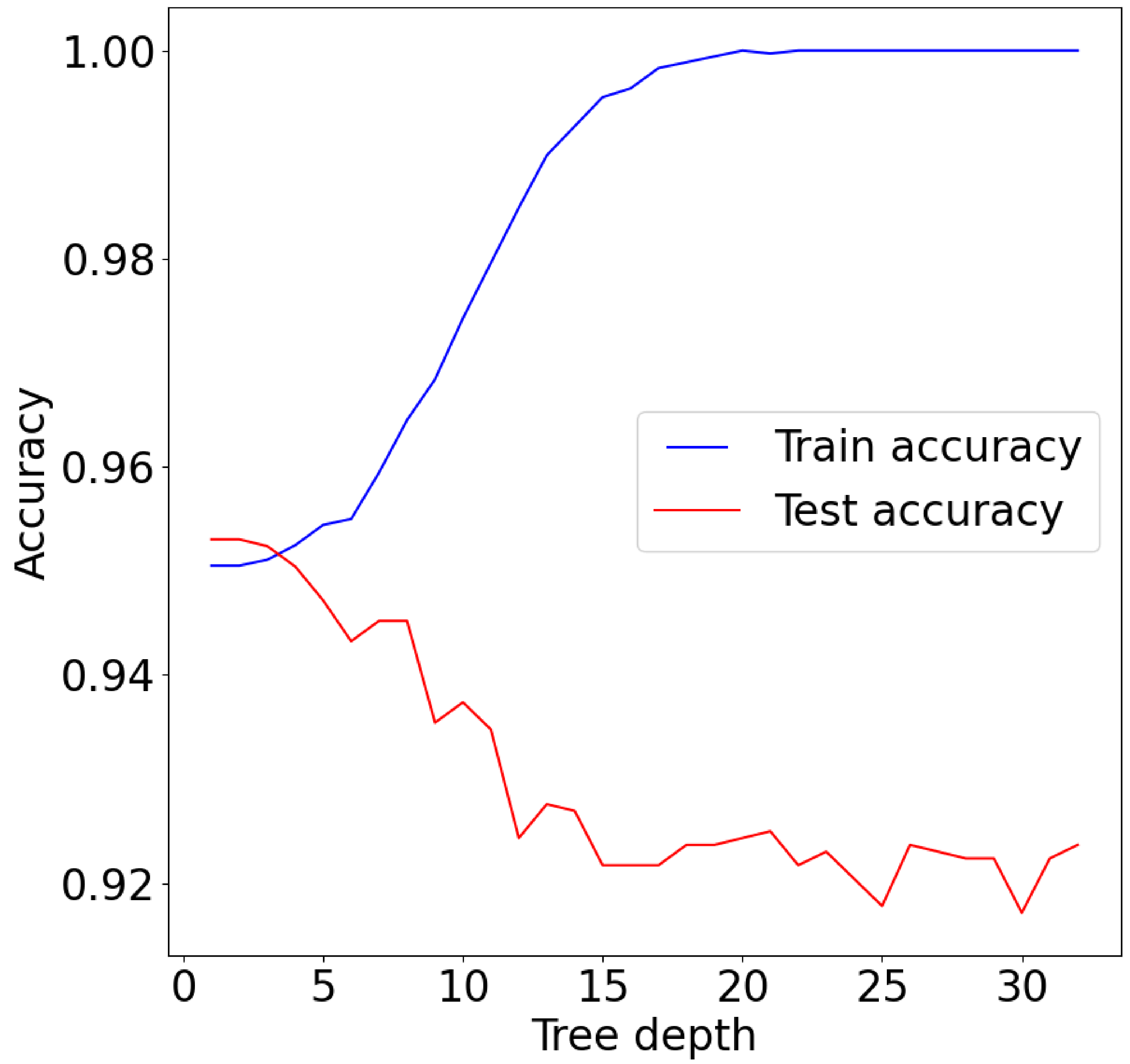
```
# Store optimal max_depth.
optimal_max_depth = optimal_parameter(max_depths, test_results);
```

Plot: max depth

- Let's plot the max depth `train_results` and `test_results`
- This will allow us to see when the model starts overfitting on train, as well as when the optimal test results are achieved
- **What observations can you make?**

```
# Plot max depth over 1 - 32.
line1, = plt.plot(max_depths, train_results, 'b',
label= "Train accuracy")
line2, = plt.plot(max_depths, test_results, 'r',
label= "Test accuracy")

plt.legend(handler_map={line1:
HandlerLine2D(numpoints = 2)})
plt.ylabel('Accuracy')
plt.xlabel('Tree depth')
plt.show()
```



Optimize: min samples split

- `min_samples_split` represents the minimum number of samples required to split an internal node
- **This varies between at least one sample at each node and all samples at each node**
- When we **increase this parameter, the tree becomes more constrained** as it has to consider more samples at each node
- We will vary the parameter from 10% to 100% of the samples

Optimize: min samples split

```
min_samples_splits = np.linspace(0.1, 1.0, 10, endpoint=True)
train_results = []
test_results = []

for min_samples_split in min_samples_splits:
    dt = DecisionTreeClassifier(min_samples_split=min_samples_split)
    dt.fit(X_train, y_train)
    train_pred = dt.predict(X_train)
    acc_train = accuracy_score(y_train, train_pred)
    # Add accuracy score to previous train results
    train_results.append(acc_train)
    y_pred = dt.predict(X_test)
    acc_test = accuracy_score(y_test, y_pred)
    # Add accuracy score to previous test results
    test_results.append(acc_test)
```

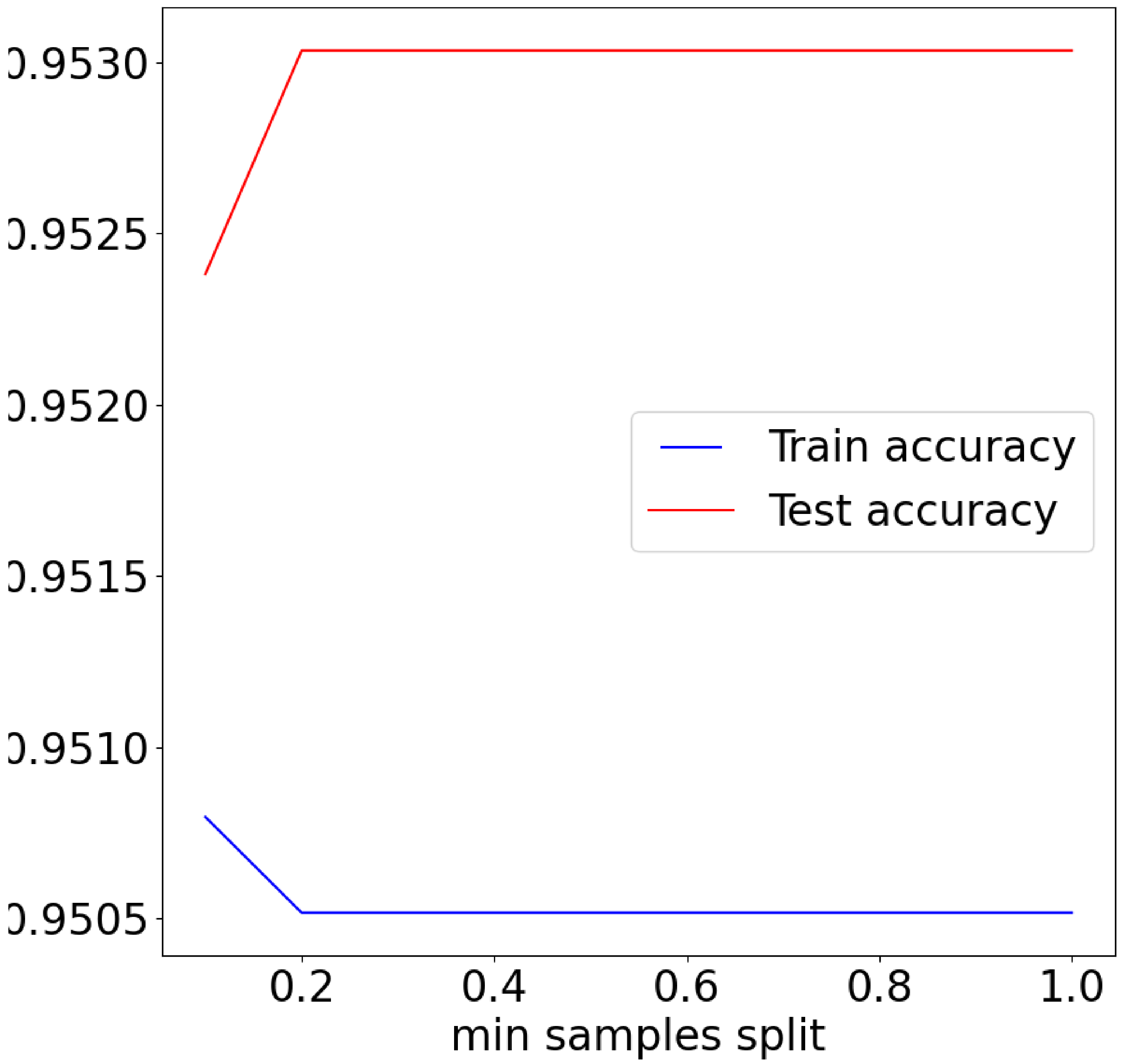
```
# Store optimal max_depth.
optimal_min_samples_split = optimal_parameter(min_samples_splits,test_results)
```

Plot: min samples split

- Let's plot the min samples split, `train_results` and `test_results`
- **What observations can you make?**

```
# Plot min_sample split.
line1, = plt.plot(min_samples_splits,
train_results, 'b', label = "Train accuracy")
line2, = plt.plot(min_samples_splits,
test_results, 'r', label = "Test accuracy")

plt.legend(handler_map={line1:
HandlerLine2D(numpoints=2)})
plt.ylabel('Accuracy')
plt.xlabel('min samples split')
plt.show()
```



Optimize: min samples leaf

- `min_samples_leaf` is the minimum number of samples required to be at a lead node
- This parameter is similar to `min_samples_split` except that **this parameter describes the minimum number of samples at the leafs - the base of the tree**

Optimize: min samples leaf

```python
# Min_samples_leaf:
min_samples_leafs = np.linspace(0.1, 0.5, 5, endpoint = True)
train_results = []
test_results = []

for min_samples_leaf in min_samples_leafs:
   dt = DecisionTreeClassifier(min_samples_leaf=min_samples_leaf)
   dt.fit(X_train, y_train)
   train_pred = dt.predict(X_train)
   acc_train = accuracy_score(y_train, train_pred)
   # Add accuracy score to previous train results
   train_results.append(acc_train)
   y_pred = dt.predict(X_test)
   acc_test = accuracy_score(y_test, y_pred)
   # Add accuracy score to previous test results
   test_results.append(acc_test)
```

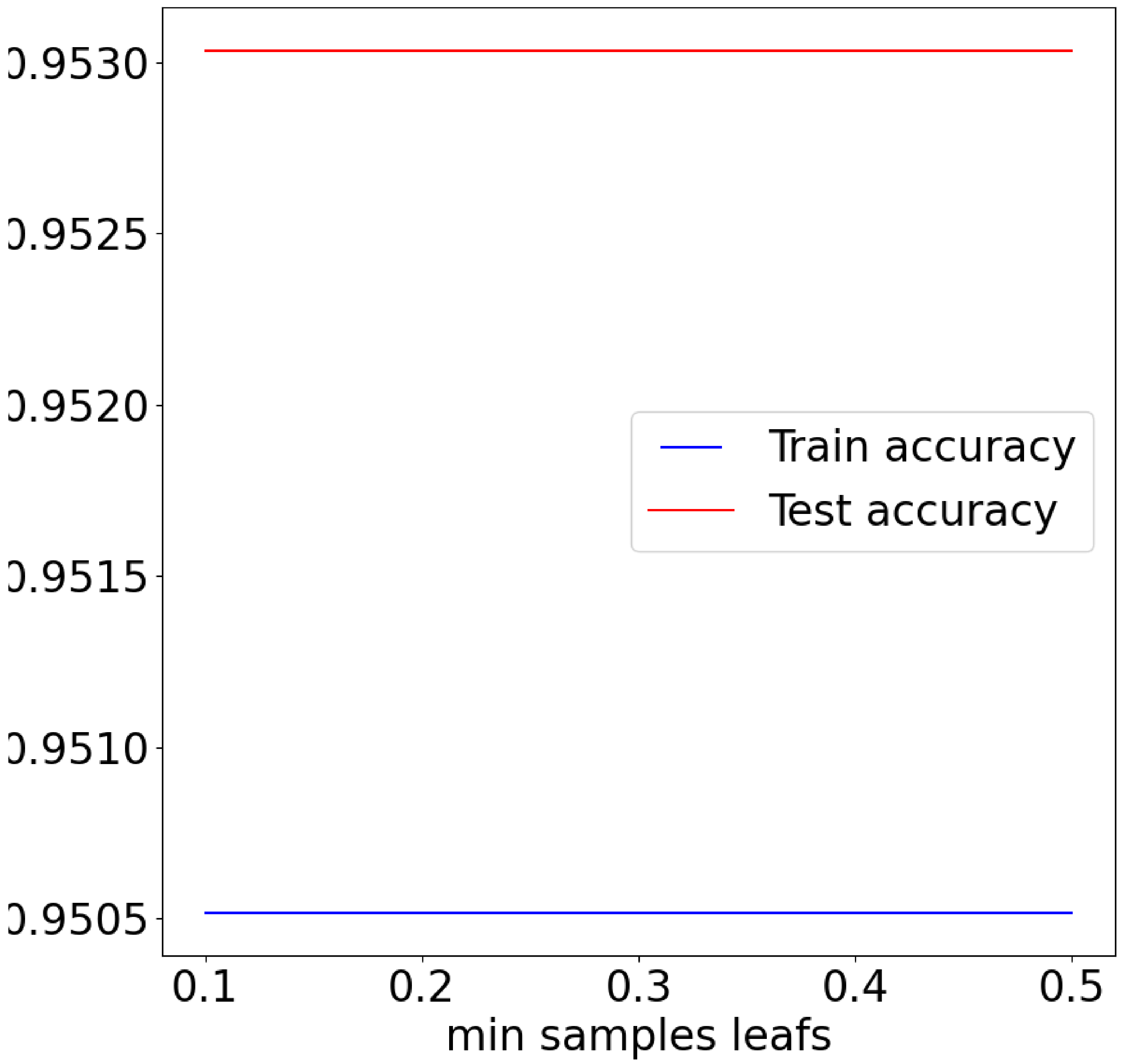
```python
optimal_min_samples_leafs = optimal_parameter(min_samples_leafs,test_results)
```

Plot: min samples leaf

- Let's plot the min samples leaf `train_results` and `test_results`
- **What observations can you make?**

```
# Plot min_sample split.
line1, = plt.plot(min_samples_leafs,
train_results, 'b', label= "Train accuracy")
line2, = plt.plot(min_samples_leafs,
test_results, 'r', label= "Test accuracy")

plt.legend(handler_map={line1:
HandlerLine2D(numpoints=2)})
plt.ylabel('Accuracy')
plt.xlabel('min samples leafs')
plt.show()
```



Optimize: max features

- `max_features` represents the number of features to consider when looking for the best split
- This parameter is set to `None` as its default value, meaning the tree will always look through all features
- This could sometimes cause overfitting and / or is computationally expensive when working with many variables

Optimize: max features

```
# Max_features:
max_features = list(range(1,X.shape[1]))
train_results = []
test_results = []

for max_feature in max_features:
   dt = DecisionTreeClassifier(max_features=max_feature)
   dt.fit(X_train, y_train)
   train_pred = dt.predict(X_train)
   acc_train = accuracy_score(y_train, train_pred)
   # Add accuracy score to previous train results
   train_results.append(acc_train)
   y_pred = dt.predict(X_test)
   acc_test = accuracy_score(y_test, y_pred)

   # Add accuracy score to previous test results
   test_results.append(acc_test)
```

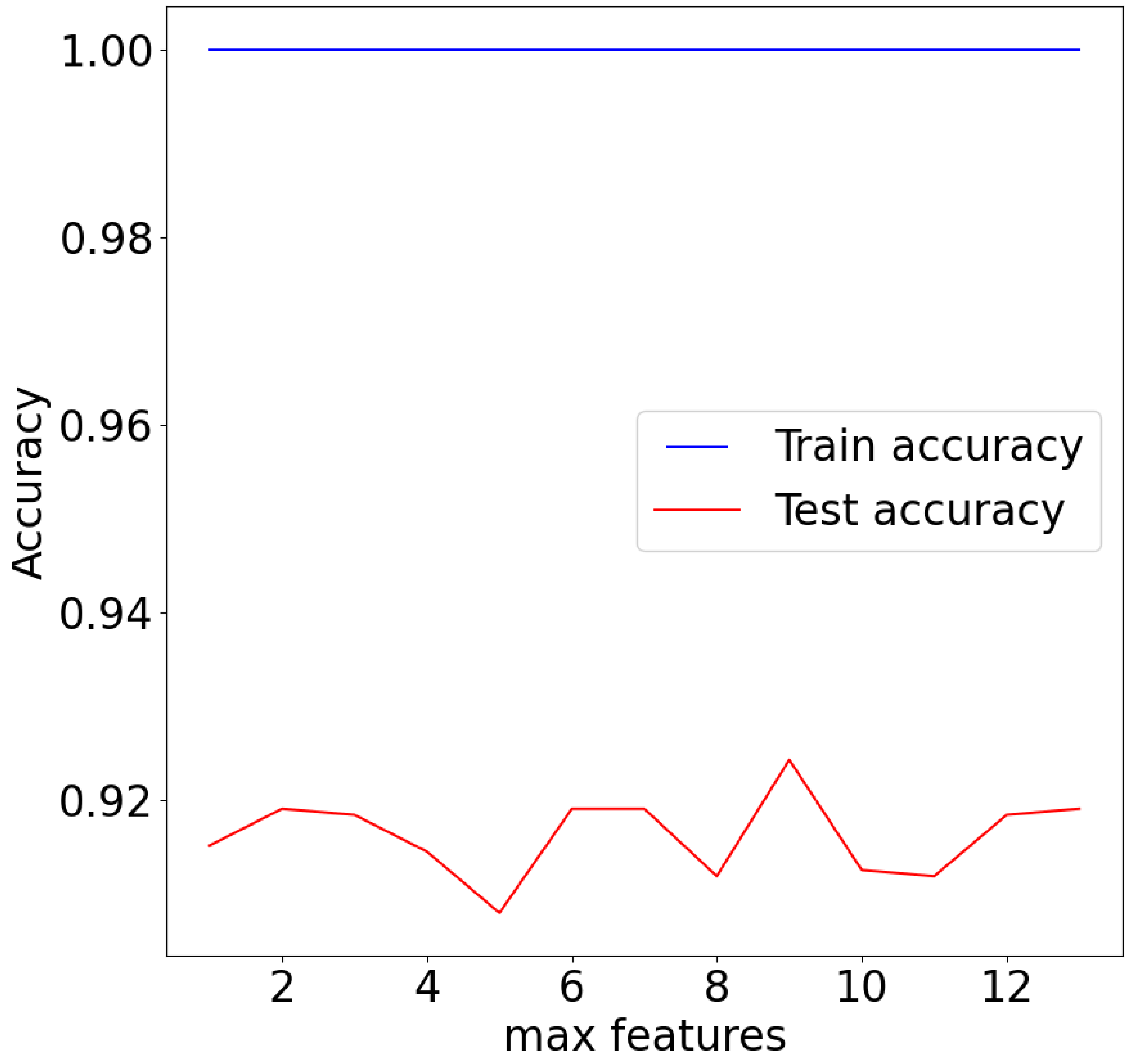
```
optimal_max_features = optimal_parameter(max_features,test_results)
```

Plot: max features

- Let's plot the max features, `train_results` and `test_results`
- **What observations can you make?**

```
# Plot min_sample split.
line1, = plt.plot(max_features, train_results, 'b', label=
"Train accuracy")
line2, = plt.plot(max_features, test_results, 'r', label=
"Test accuracy")

plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel('Accuracy')
plt.xlabel('max features')
plt.show()
```



- This is a case of overfitting - we can see max train accuracy for all values of max_features
- The search for the best split does not stop until at least one valid partition of the nodes is found, even if more than `max_features` features is inspected

Module completion checklist

Objectives	Complete
Optimize the Decision Tree by tuning the hyperparameters	✔
Run the optimized model, predict, and evaluate the new model	

Optimized model

- We have now walked through four parameters that will help us optimize our Decision Tree
- Remember that when we optimized each parameter, we saved the optimal parameter using our `optimal_parameter` function

```
print("The optimal max depth is:", optimal_max_depth)
```

```
The optimal max depth is: 1
```

```
print("The optimal min samples split is:", optimal_min_samples_split)
```

```
The optimal min samples split is: 0.2
```

```
print("The optimal min samples leaf is:", optimal_min_samples_leafs)
```

```
The optimal min samples leaf is: 0.1
```

```
print("The optimal max features is:", optimal_max_features)
```

```
The optimal max features is: 9
```

Build optimized model

- Now, we will run the optimized model on our `X_train`

```
# Set the seed.
np.random.seed(1)

# Implement the Decision Tree on X_train.
clf_optimized = tree.DecisionTreeClassifier(max_depth = optimal_max_depth,
                                            min_samples_split = optimal_min_samples_split,
                                            min_samples_leaf = optimal_min_samples_leafs,
                                            max_features = optimal_max_features)

# We can now see our optimized features where before they were just default:
print(clf_optimized)
```

```
DecisionTreeClassifier(max_depth=1, max_features=9, min_samples_leaf=0.1,
                       min_samples_split=0.2)
```

```
clf_optimized_fit = clf_optimized.fit(X_train, y_train)
```

Predict with optimized model

- Finally, let's predict on `X_test` and calculate our accuracy score
- **How is our optimized model doing?**
- **What other metrics can you also look at?**

```
# Predict on X_test.
y_predict_optimized = clf_optimized_fit.predict(X_test)

# Get the accuracy score.
acc_score_tree_optimized = accuracy_score(y_test, y_predict_optimized)

print(acc_score_tree_optimized)
```

```
0.9530332681017613
```

Train accuracy

```
# Compute accuracy using training data.
acc_train_tree_optimized = clf_optimized_fit.score(X_train,
                                                   y_train)

print ("Train Accuracy:", acc_train_tree_optimized)
```

```
Train Accuracy: 0.9505171931786414
```

Predict and save results

- Now we know some of the parameters that help us optimize our Decision Tree
- **What is another way you could optimize the tree, instead of searching each parameter separately?**

Predict and save results

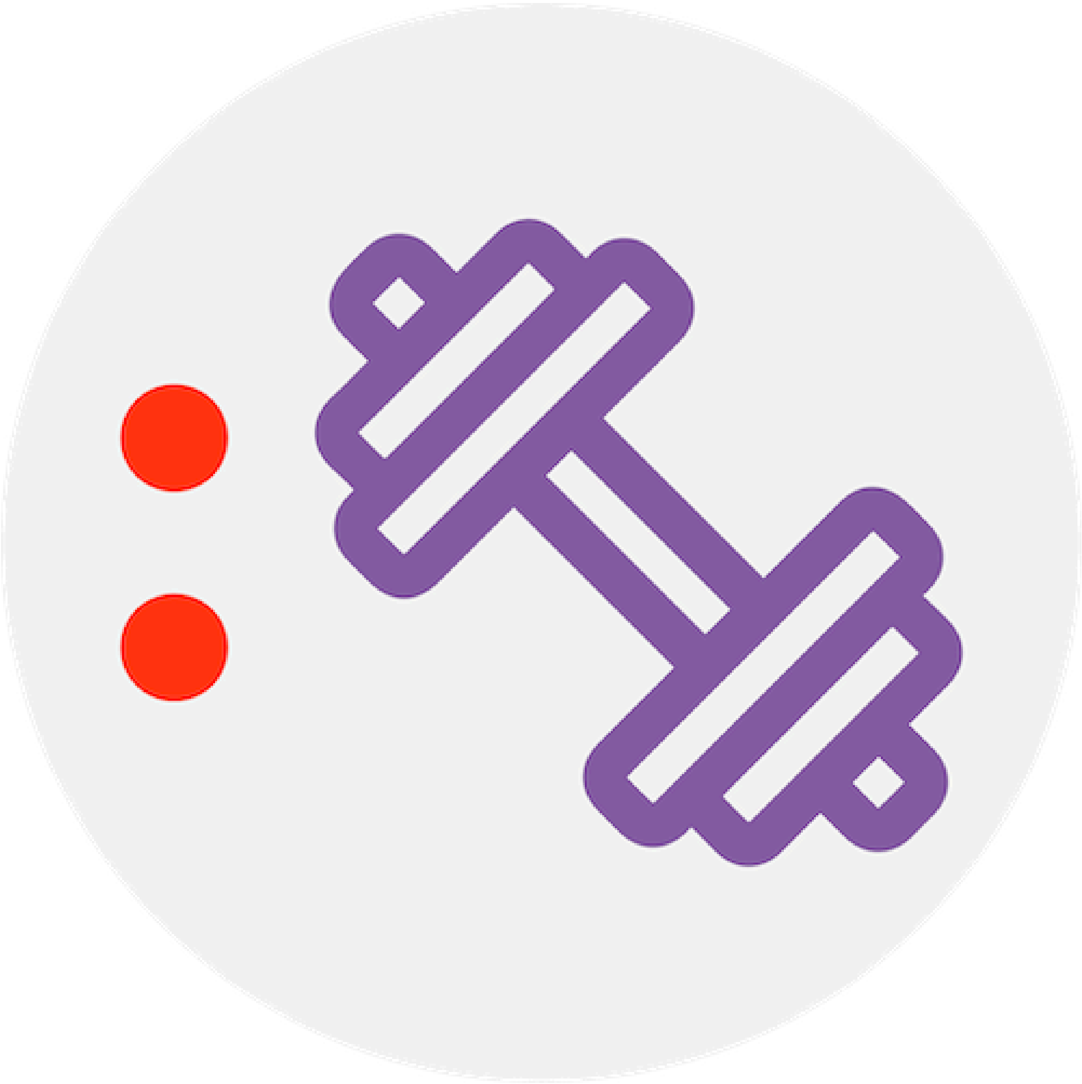
- This other method is `GridSearchCV`. Although it's helpful, it can be computationally expensive
- Going through each of the four parameters helps you understand your tree a little better
- **Let's see our optimized tree accuracy score in our `model_final` dataset**

```
# Add the optimized model to our dataframe.
model_final_tree = {'metrics' : "accuracy" ,
                    'values' : round(acc_score_tree_optimized,4),
                    'model':'tree_all_variables_optimized' }
print(model_final_tree)
```

```
{'metrics': 'accuracy', 'values': 0.953, 'model':
'tree_all_variables_optimized'}
```

Knowledge check

Exercise



You are now ready to try Tasks 8-12 in the Exercise for this topic

Module completion checklist

Objectives	Complete
Optimize the Decision Tree by tuning the hyperparameters	✔
Run the optimized model, predict, and evaluate the new model	✔

Decision Trees: Topic summary

In this part of the course, we have covered:

- Decision Trees use cases and the theory behind them
- Data transformation necessary for Decision Trees
- Implementation of Decision Trees on a dataset
- Model performance evaluation and tuning

Congratulations on completing this module!