Generalization & Overfitting

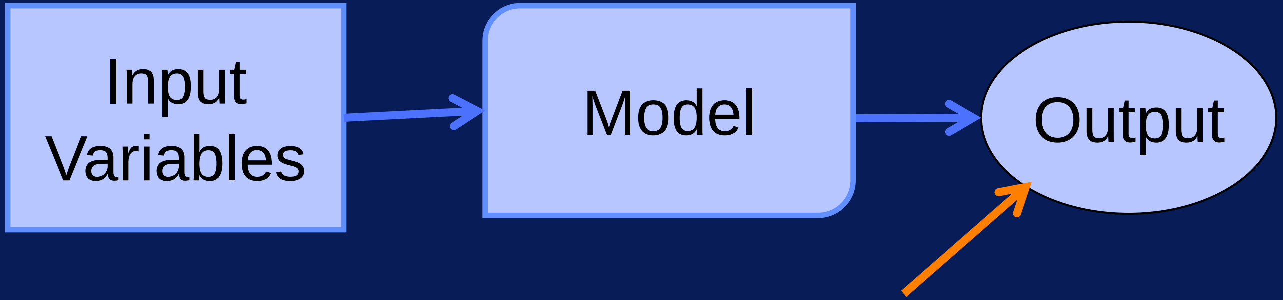
After this video you will be able to..

- Define what generalization is
- Describe how overfitting is related to generalization
- Explain why overfitting should be avoided

Errors in Classification

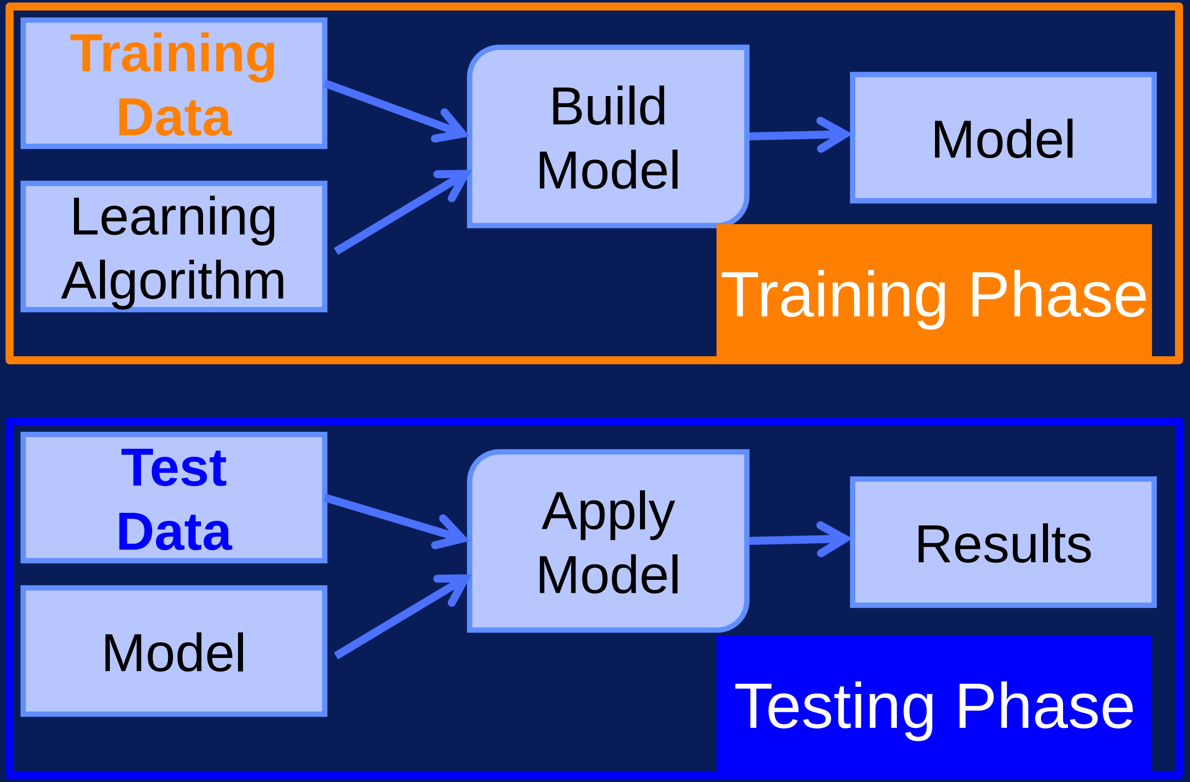
- Success: Output = Target
- Error: Output != Target
- Error rate = Error = Misclassification Error
  - # errors / # samples = % error

true label



predicted label

Training vs. Testing Phases



Errors in Classification

Error on Training Data

Training Error

Error on Test Data

Test Error

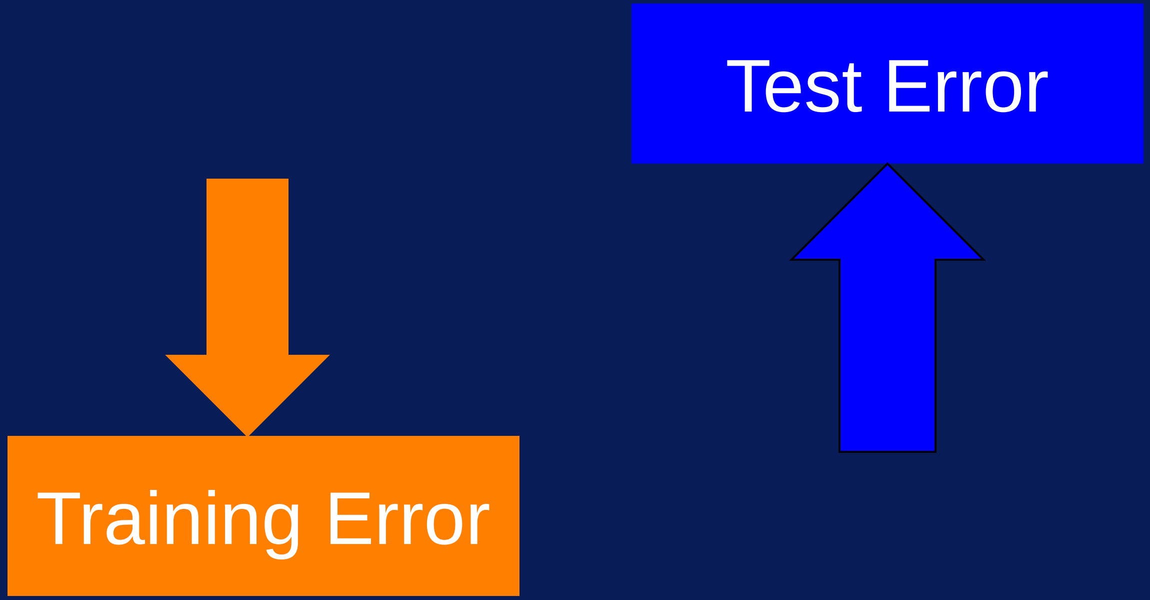
Test error indicates how well model will perform on new data!

Generalization

Performs well on new data → *Good Generalization*

Test Error = Generalization Error

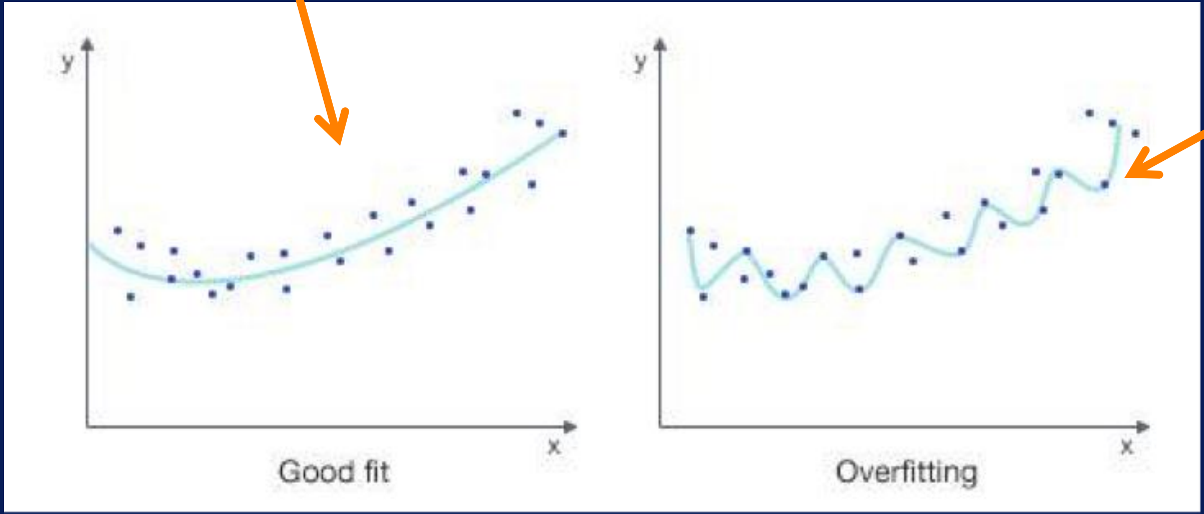
Overfitting



Overfitting

Model is fitting to structure of data

Model is fitting to noise in data

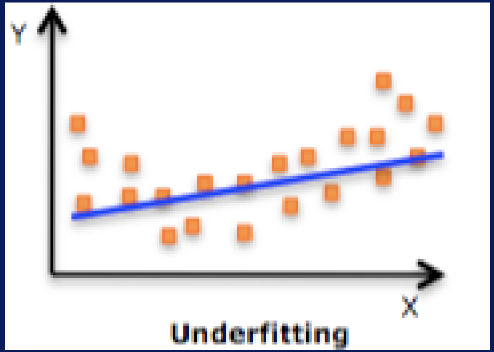


Source: http://blog.fliptop.com/blog/2015/03/02/bias-variance-and-overfitting-machine-learning-overview/

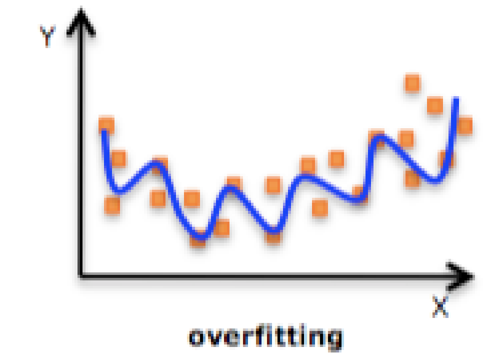
Overfitting & Generalization

Overfitting ➡ Poor Generalization

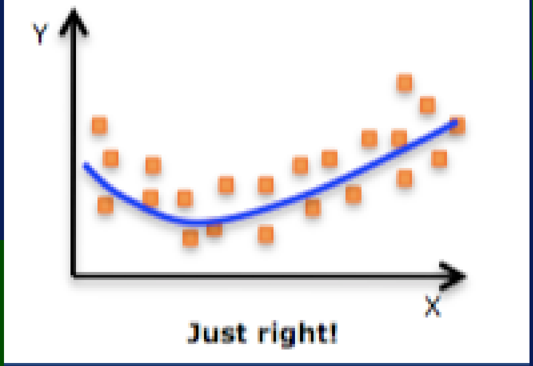
Overfitting & Underfitting



Underfitting

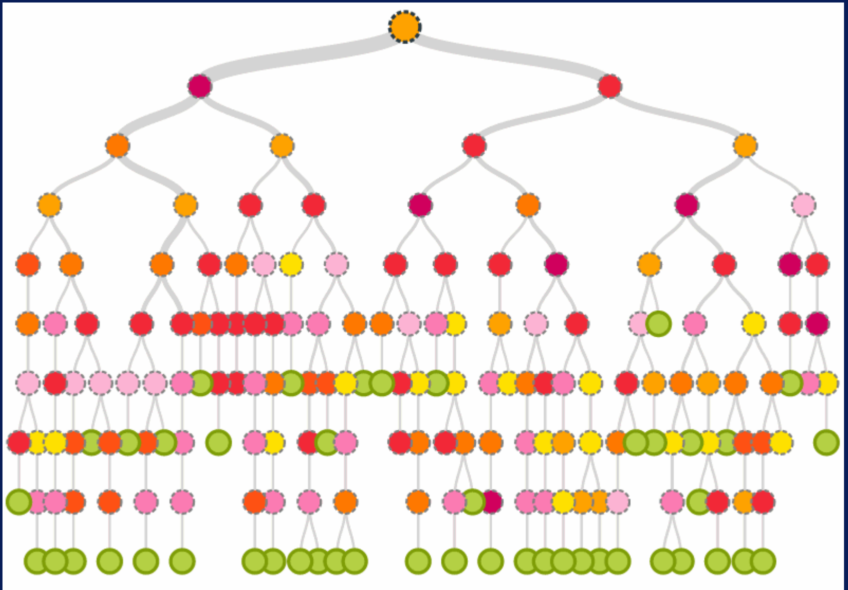


Overfitting

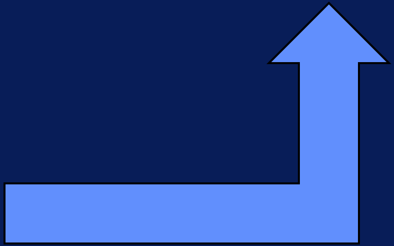


Just Right

What Causes Overfitting?



Overly complex model



Overfitting

Generalization & Overfitting

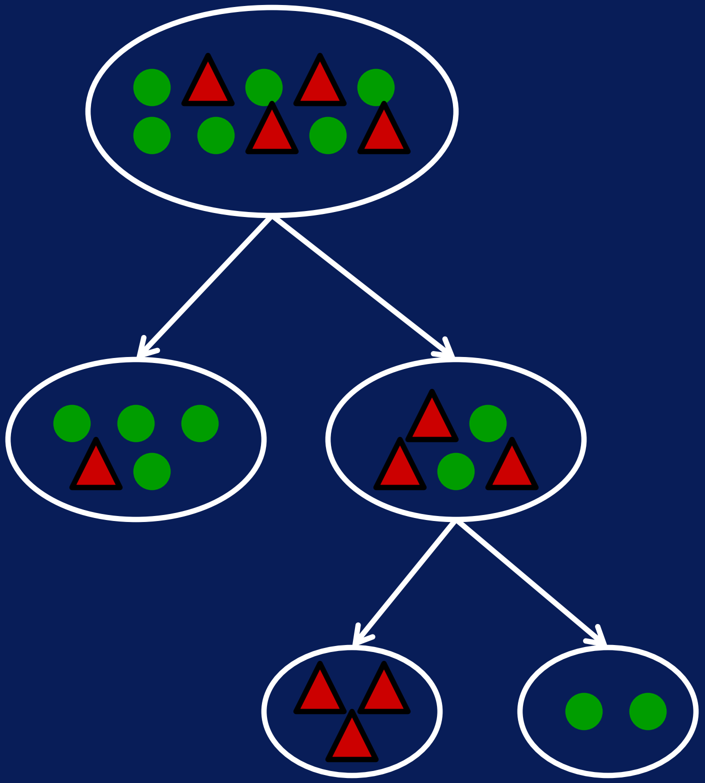
~~Overfitting~~ → Good Generalization

Overfitting in Decision Trees

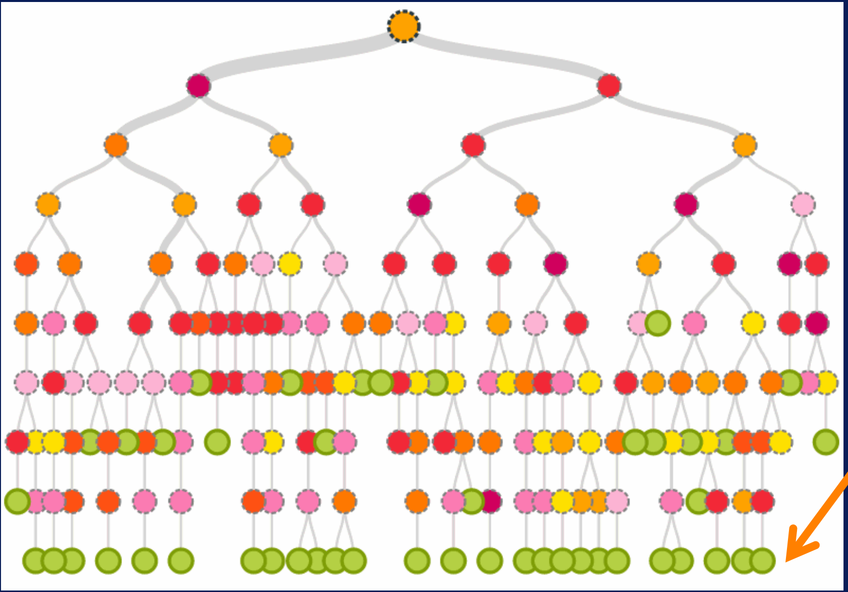
After this video you will be able to..

- Discuss overfitting in the context of decision tree models
- Explain how overfitting is addressed in decision tree induction
- Define pre-pruning and post-pruning

Decision Tree Induction



Overfitting in Decision Tree



Source: http://piepdx.org/blog/2013/12/10/which-one-is-is

If nodes are fitting to noise in training data, model will not generalize well

Avoiding Overfitting in Decision Tree

Pre-Pruning	Post-Pruning
Stop growing tree before fully grown	Grow tree to max size, then prune

Control number of nodes to limit complexity of tree

Pre-Pruning

- Restrictive stopping conditions for growing tree:
  - Stop if number of records < some threshold
  - Stop if improvement in impurity measure < some threshold

Pre-Pruning
Stop growing tree before fully grown

Post-Pruning

- Pruning
  - Remove nodes from bottom up
  - Replace subtree with leaf node if generalization error improves or does not change

Post-Pruning
Grow tree to max size, then prune

Overfitting in Decision Tree

Pre-Pruning	Post-Pruning
Stop growing tree before fully grown	Grow tree to max size, then prune

- Post-pruning used more often
- But is more computational expensive

Tree Pruning to Avoid Overfitting

Pre-Pruning	Post-Pruning
Stop growing tree before fully grown	Grow tree to max size, then prune

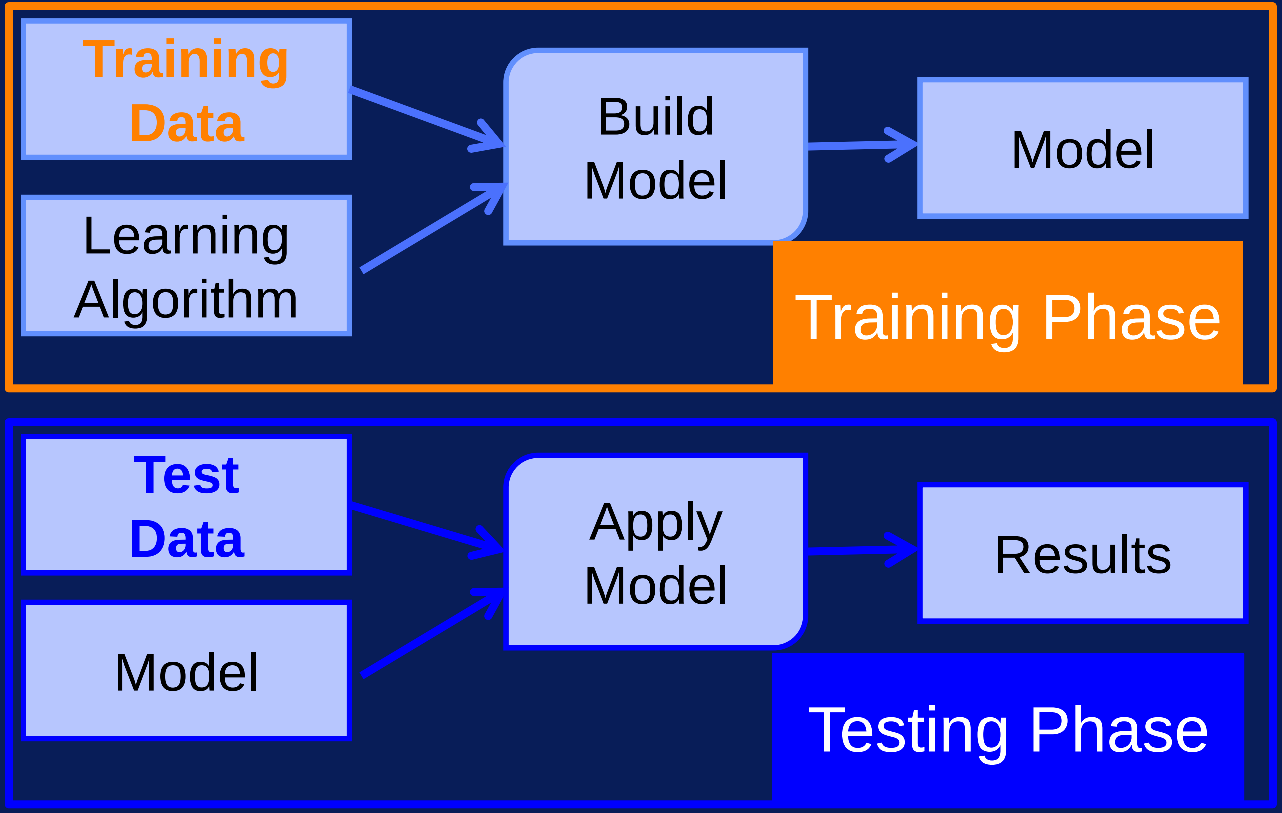
Control number of nodes to limit complexity of tree

Using a Validation Set

After this video you will be able to..

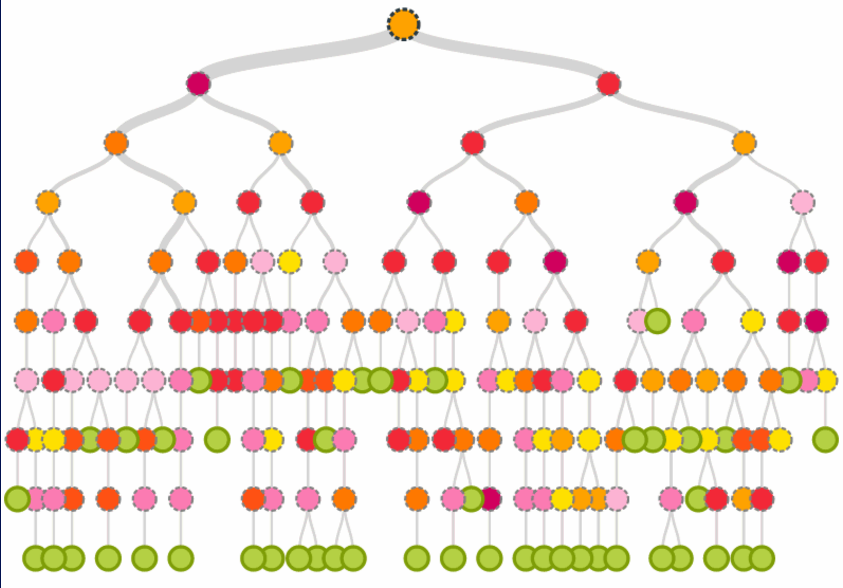
- Describe how a validation set can be used to avoid overfitting
- Articulate how training, validation, and test sets are used
- List three ways that validation can be performed

Training vs. Testing Phases



Avoiding Overfitting

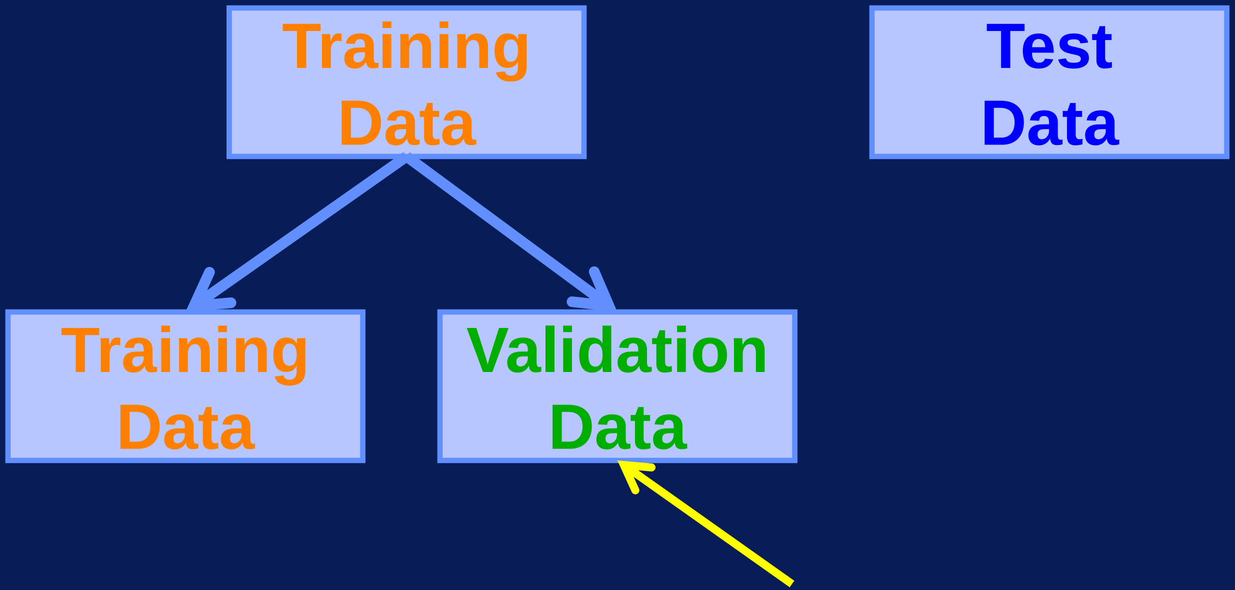
Overly complex model



Overfitting

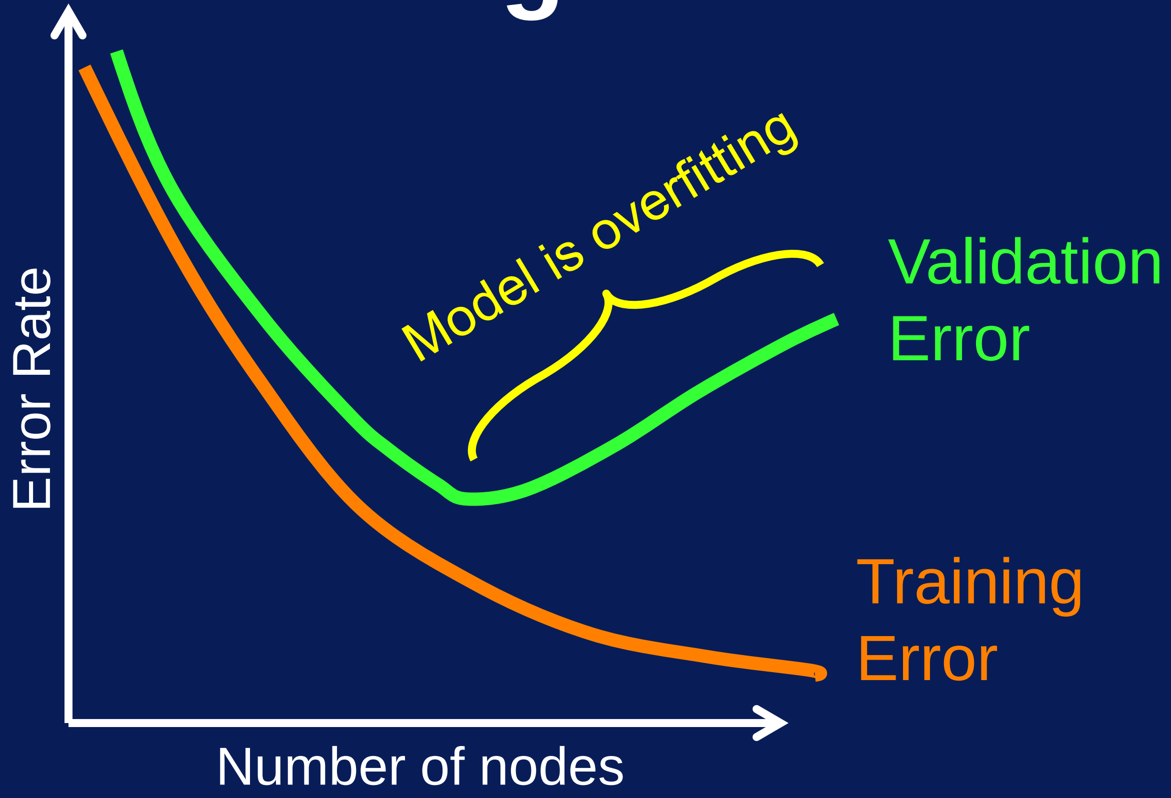
When to stop training before model gets too complex?

Validation Set

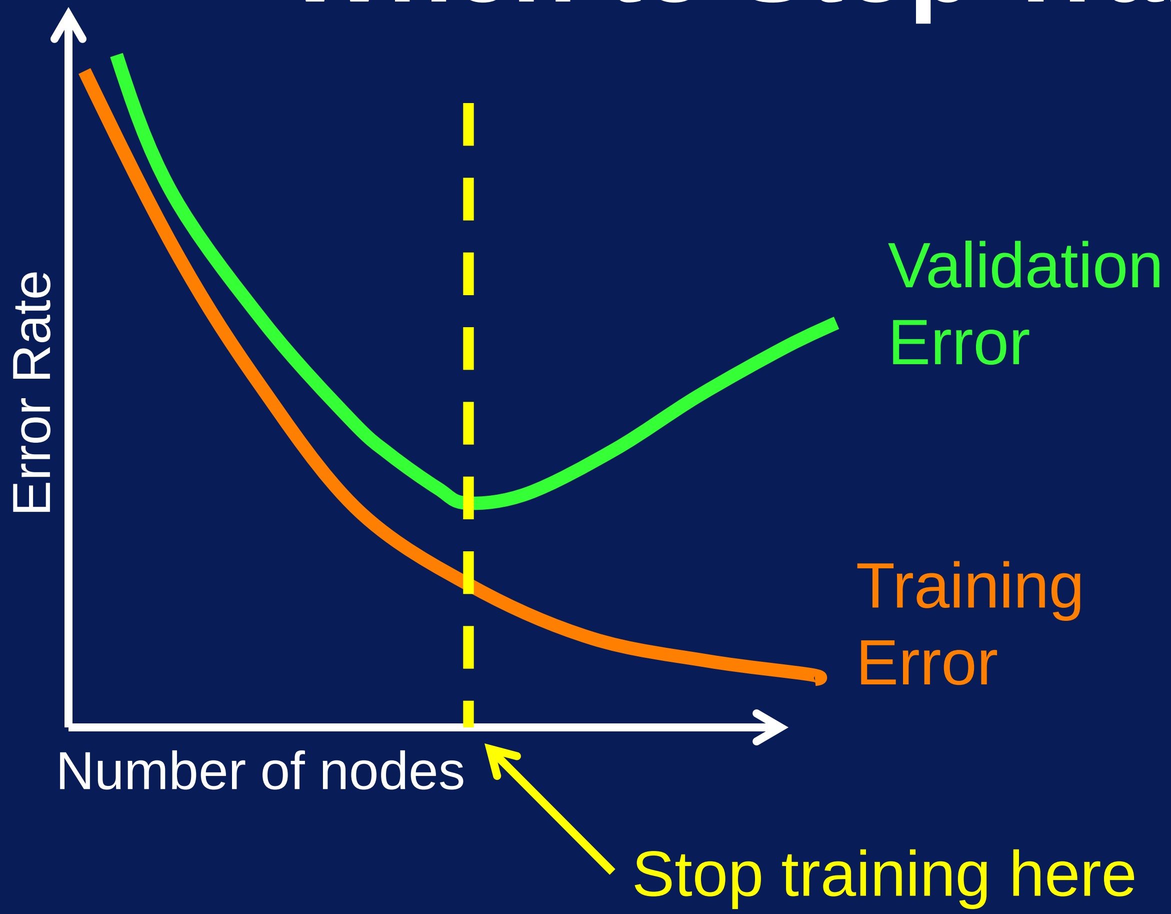


Used to determine when to stop training to avoid overfitting

Training & Validation Errors



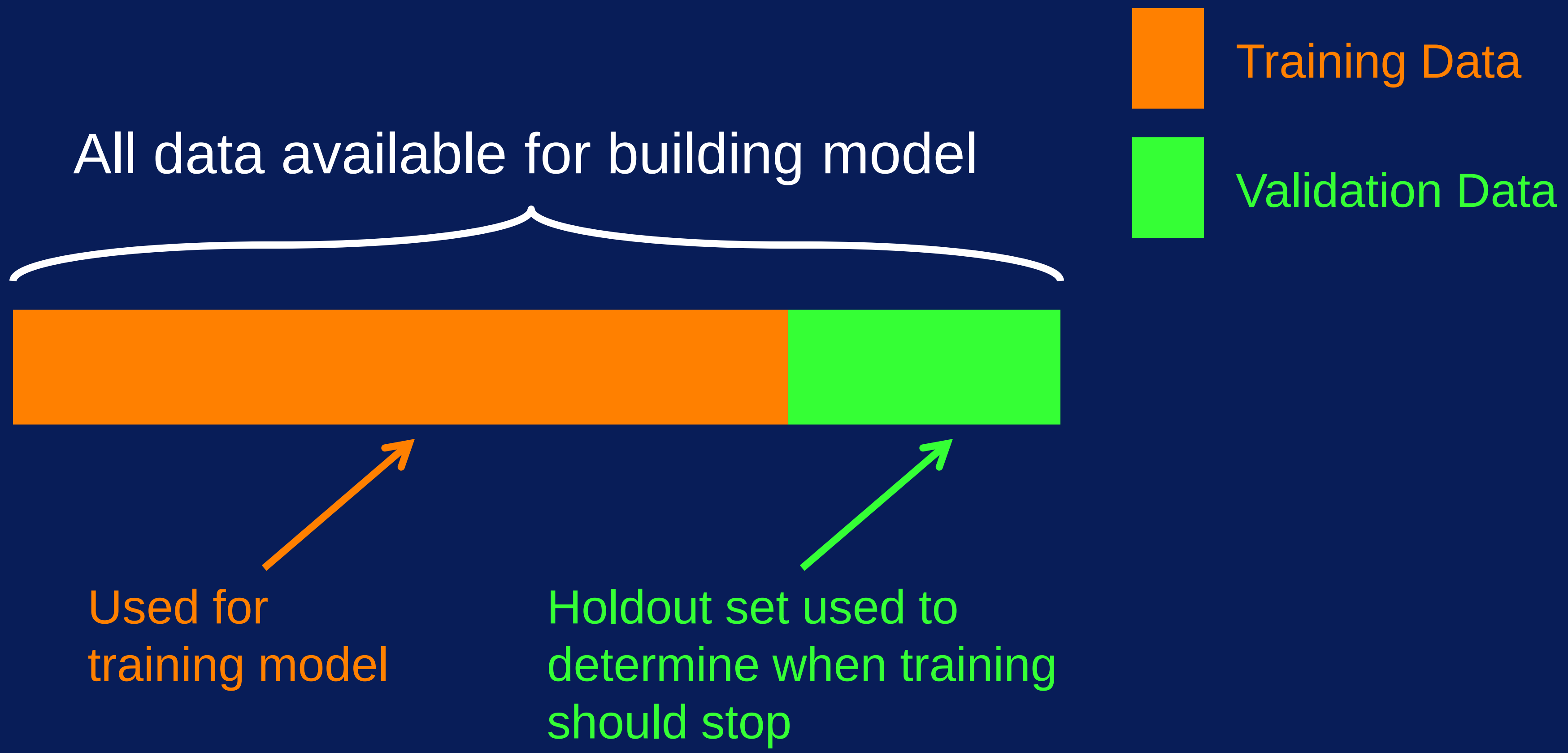
When to Stop Training



Ways to Create & Use Validation Set

- Holdout method
- Random subsampling
- K-fold cross-validation
- Leave-one-out cross-validation

Holdout Method



Repeated Holdout

- Repeating holdout method several times
- Randomly select different hold out set each iteration
- Average validation errors over all repetitions

K-Fold Cross-Validation

Leave-One-Out Cross-Validation

Uses of Validation Set

Validation Data

- Uses:
  - Address overfitting
  - Estimate generalization performance

Datasets

Training Data

Adjust model parameters

Validation Data

Determine when to stop training (avoid overfitting)

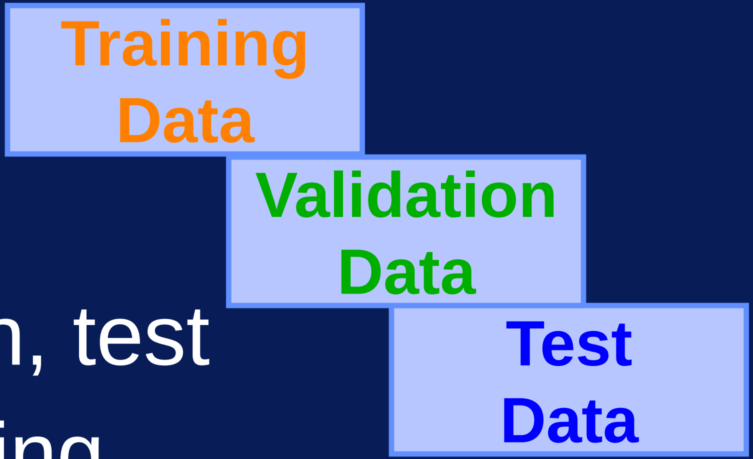
Estimate generalization performance

Test Data

Evaluate performance on new data

Cannot be used in <u>any</u> way in model creation!

Validation Set Summary



- Datasets: training, validation, test
- Validation set: avoid overfitting, estimate generalization
- Using validation: holdout, repeated holdout, cross-validation (k-fold, leave-one-out)

Metrics to Evaluate Model Performance

After this video you will be able to..

- Discuss how performance metrics can be used to evaluate models
- Name three model evaluation metrics
- Explain why accuracy may be misleading

Classification

Is this animal a mammal?

Yes

No

Class Labels

Types of Classification Errors

Is this animal a mammal?

Yes No

Class Labels

True Label	Predicted Label	Error Type
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)

Accuracy Rate

True	Predicted	Error
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)

$$\text{Accuracy Rate} = \frac{\#\ \text{correct predictions}}{\#\ \text{total predictions}}$$

$$= \frac{TP + TN}{TP + TN + FP + FN}$$

Error Rate

True	Predicted	Error
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)

$$\text{Error Rate} = \frac{\text{\# incorrect predictions}}{\text{\# total predictions}}$$

$$= \frac{\text{FN+ TP}}{\text{TP + TN + FP + FN}}$$

$$= 1 - \text{Accurate Rate}$$

True Label	Predicted Label
Yes	No
No	No
No	No
Yes	Yes
Yes	Yes
No	No
Yes	No
Yes	Yes
No	No
No	Yes

True	Predicted	Error
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)

Classification Example

True Label	Predicted Label
Yes	No
No	No
No	No
Yes	Yes
Yes	Yes
No	No
Yes	No
Yes	Yes
No	No
No	Yes

True	Predicted	Error
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)

TP= 3

Classification Example

True Label	Predicted Label
Yes	No
No	No
No	No
Yes	Yes
Yes	Yes
No	No
Yes	No
Yes	Yes
No	No
No	Yes

TN = 4

True	Predicted	Error
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)

Classification Example

Accuracy Rate

True	Predicted	Error
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)

$$\text{Accuracy Rate} = \frac{\#\text{ correct predictions}}{\#\text{ total predictions}}$$

$$= \frac{TP + TN}{TP + TN + FP + FN}$$

$$= (3 + 4) / 10 = 7 / 10 = 0.7$$

Error Rate

True	Predicted	Error
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)

$$\text{Error Rate} = \frac{\text{\# incorrect predictions}}{\text{\# total predictions}}$$

$$= 1 - \text{Accuracy Rate}$$

$$= 1 - 0.7 = 0.3$$

Limitation with Accuracy

Is this tumor cancerous?

most are negative examples

very few positive examples

Class Imbalance Problem

Limitation with Accuracy

Is this tumor cancerous?

- Say 3% of samples are cancer
- If model <u>always</u> predicts non-cancer
  - Accuracy = 97%
  - But no cancer cases detected!

Precision & Recall

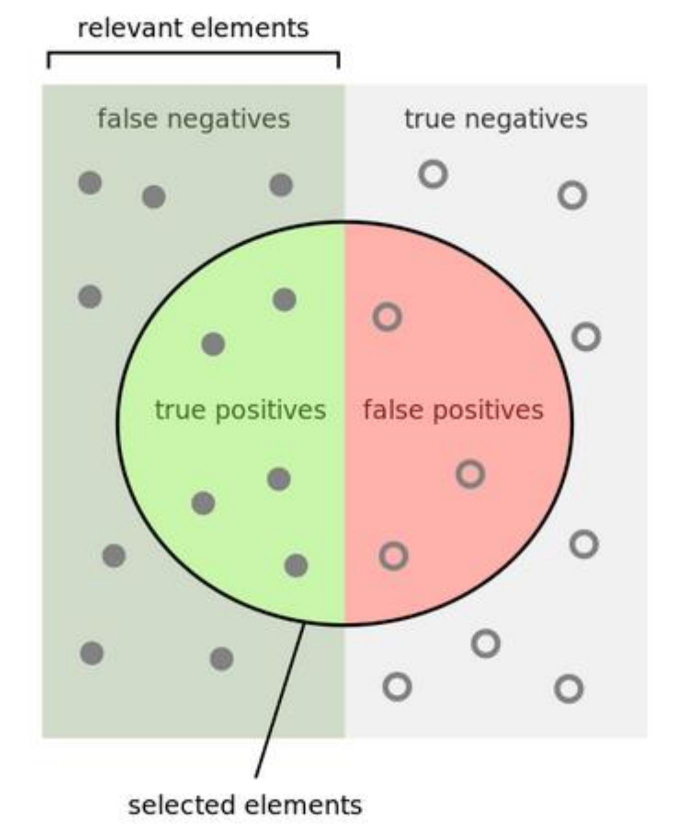
True	Predicted	Error
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)

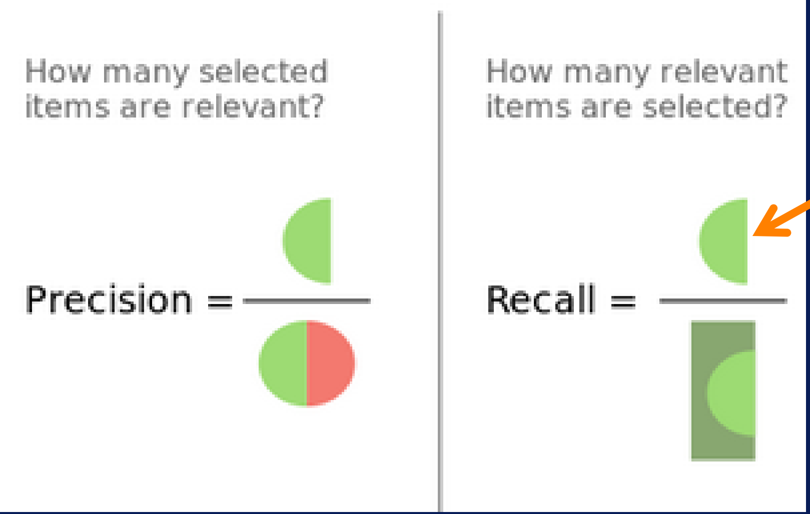
$$\text{Precision} = \frac{TP}{TP + FP}$$

All samples with Predicted = Yes

$$\text{Recall} = \frac{TP}{TP + FN}$$

All samples with True = Yes

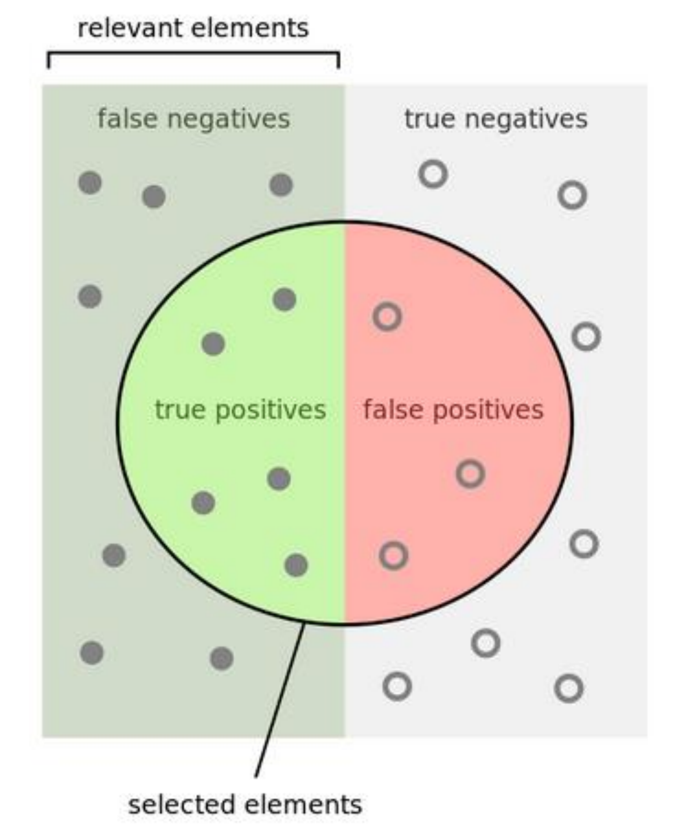


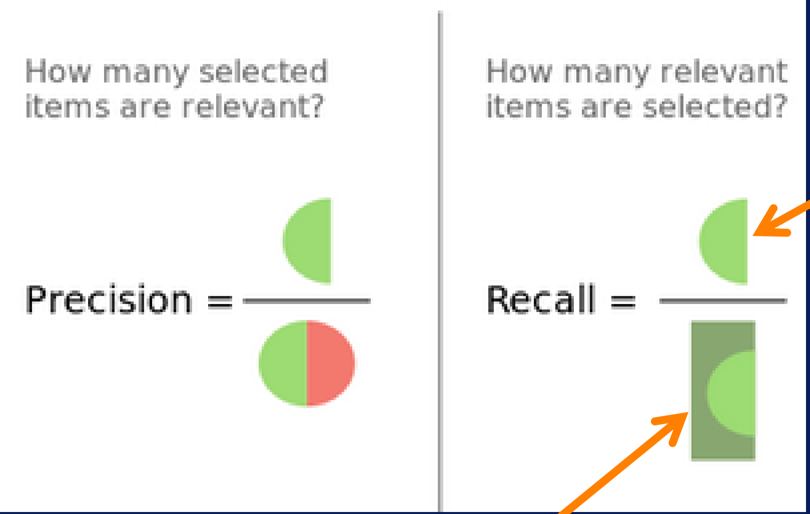


Samples correctly predicted as Positive

Recall

Source: https://en.wikipedia.org/wiki/Precision_and_recall



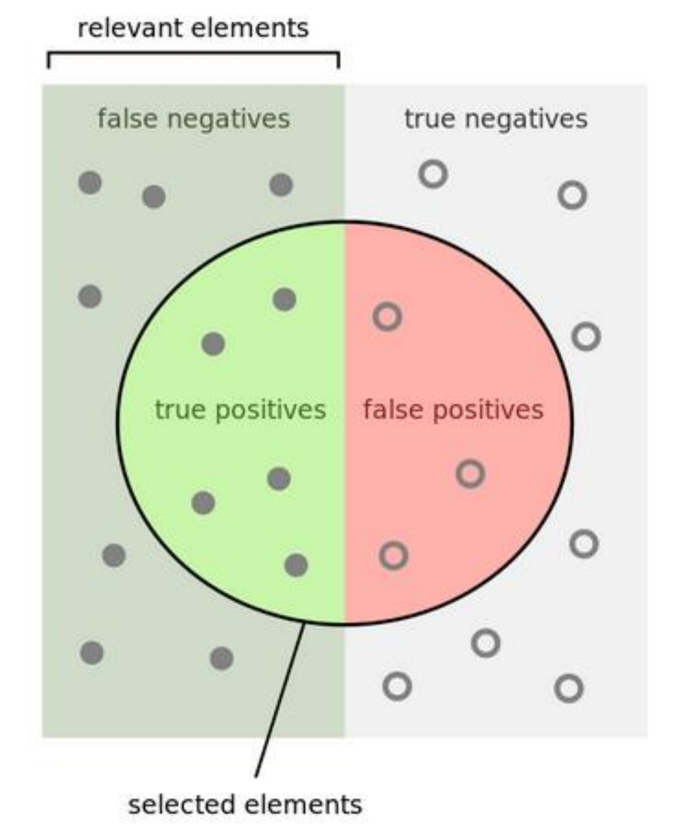


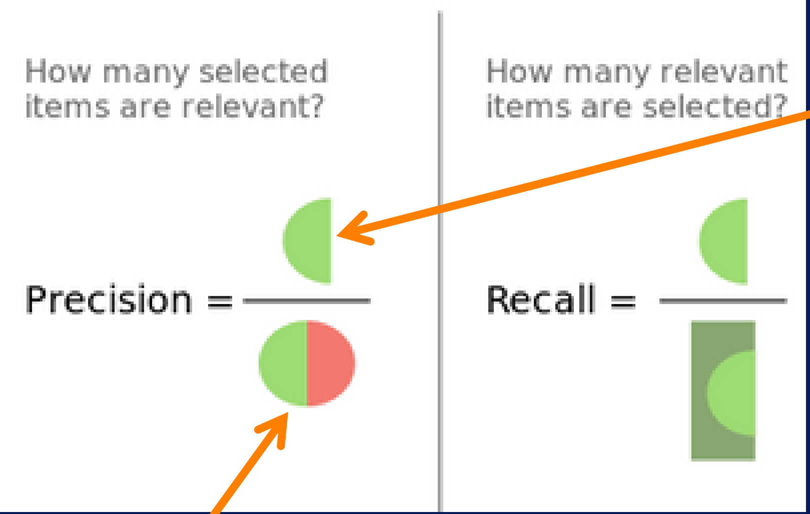
Samples correctly predicted as Positive

Samples actually Positive

Recall

Source: https://en.wikipedia.org/wiki/Precision_and_recall





Samples correctly predicted as Positive

Samples predicted as Positive

Recall

Source: https://en.wikipedia.org/wiki/Precision_and_recall

Precision & Recall

True	Predicted	Error
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{\text{Positive samples correctly predicted}}{\text{All samples predicted as Positive}}$$

Measure of exactness

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{\text{Positive samples correctly predicted}}{\text{All samples with true label Positive}}$$

Measure of completeness

Precision & Recall

Precision + **Recall**

- Use together
- Goal: Maximize both

F-Measure

Precision + **Recall**

$$F_1 = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

- F_1 : evenly weighted
- F_2 : weights Recall more
- $F_{0.5}$: weights Precision more

Evaluation Metrics

True	Predicted	Error
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)

Accuracy Rate

Error Rate

Precision & Recall

F_1-Measure

Confusion Matrix

After this video you will be able to..

- Describe how a confusion matrix can be used to evaluate a classifier
- Interpret the confusion matrix of a model
- Relate accuracy to values in a confusion matrix

Classification

Is this animal a mammal?

Yes

No

Class Labels

Types of Classification Errors

Is this animal a mammal?

Yes No

Class Labels

True Label	Predicted Label	Error Type
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)

Confusion Matrix

Is this animal a mammal?

Yes *No*

Class Labels

		Predicted Class Label	
True Class Label		**Yes**	**No**
	Yes	True Positive (TP)	False Negative (FN)
	No	False Positive (FP)	True Negative (TN)

True Label	Predicted Label
Yes	No
No	No
No	No
Yes	Yes
Yes	Yes
No	No
Yes	No
Yes	Yes
No	No
No	Yes

	Predicted Class Label		
True Class Label		Yes	No
	Yes	TP	FN
	No	FP	TN

Confusion Matrix

True Label	Predicted Label
Yes	No
No	No
No	No
Yes	Yes
Yes	Yes
No	No
Yes	No
Yes	Yes
No	No
No	Yes

TP

		Predicted Class Label	
True Class Label		Yes	No
	Yes	TP = 3	
	No		

Confusion Matrix

True Label	Predicted Label
Yes	No
No	No
No	No
Yes	Yes
Yes	Yes
No	No
Yes	No
Yes	Yes
No	No
No	Yes

TN

		Predicted Class Label	
True Class Label		Yes	No
	Yes	TP = 3	
	No		TN = 4

Confusion Matrix

True Label	Predicted Label
Yes	No
No	No
No	No
Yes	Yes
Yes	Yes
No	No
Yes	No
Yes	Yes
No	No
No	Yes

FN

		Predicted Class Label	
True Class Label		Yes	No
	Yes	TP = 3	FN = 2
	No		TN = 4

Confusion Matrix

True Label	Predicted Label
Yes	No
No	No
No	No
Yes	Yes
Yes	Yes
No	No
Yes	No
Yes	Yes
No	No
No	Yes

FP

	Predicted Class Label		
True Class Label		Yes	No
	Yes	TP = 3	FN = 2
	No	FP = 1	TN = 4

Confusion Matrix

True Label	Predicted Label
Yes	No
No	No
No	No
Yes	Yes
Yes	Yes
No	No
Yes	No
Yes	Yes
No	No
No	Yes

		Predicted Class Label	
True Class Label		Yes	No
	Yes	TP = 3	FN = 2
	No	FP = 1	TN = 4

Confusion Matrix

True Label	Predicted Label
Yes	No
No	No
No	No
Yes	Yes
Yes	Yes
No	No
Yes	No
Yes	Yes
No	No
No	Yes

	Predicted Class Label		
True Class Label		Yes	No
	Yes	TP = 3	FN = 2
	No	FP = 1	TN = 4

Correct Predictions :
7 out of 10 = 0.7

True Label	Predicted Label
Yes	No
No	No
No	No
Yes	Yes
Yes	Yes
No	No
Yes	No
Yes	Yes
No	No
No	Yes

		Predicted Class Label	
True Class Label		Yes	No
	Yes	TP = 3	FN = 2
	No	FP = 1	TN = 4

Incorrect Predictions : 3 out of 10 = 0.3

Confusion Matrix & Accuracy Rate

		Predicted Class Label	
True Class Label		Yes	No
	Yes	TP = 3	FN = 2
	No	FP = 1	TN = 4

$$\text{Accuracy Rate} = \frac{\#\ \text{correct predictions}}{\#\ \text{total predictions}}$$

$$= \frac{TP + TN}{TP + TN + FP + FN}$$

$$= (3 + 4) / 10 = 7 / 10 = 0.7$$

Confusion Matrix & Error Rate

		Predicted Class Label	
True Class Label		Yes	No
	Yes	TP = 3	FN = 2
	No	FP = 1	TN = 4

$$\text{Error Rate} = \frac{\#\ \text{incorrect predictions}}{\#\ \text{total predictions}}$$

$$= 1 - \text{Accuracy Rate}$$

$$= 1 - 0.7 = 0.3$$

Misclassifications in Confusion Matrix

	Predicted Class Label		
True Class Label		Yes	No
	Yes	TP = 3	FN = 2
	No	FP = 1	TN = 4

High value means classifying Positive class is problematic

High value means classifying Negative class is problematic

Confusion Matrix

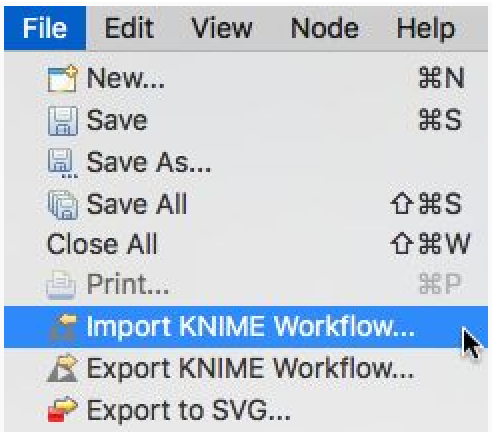
	Predicted Class Label		
True Class Label		Yes	No
	Yes	TP	FN
	No	FP	TN

Completed KNIME Workflows

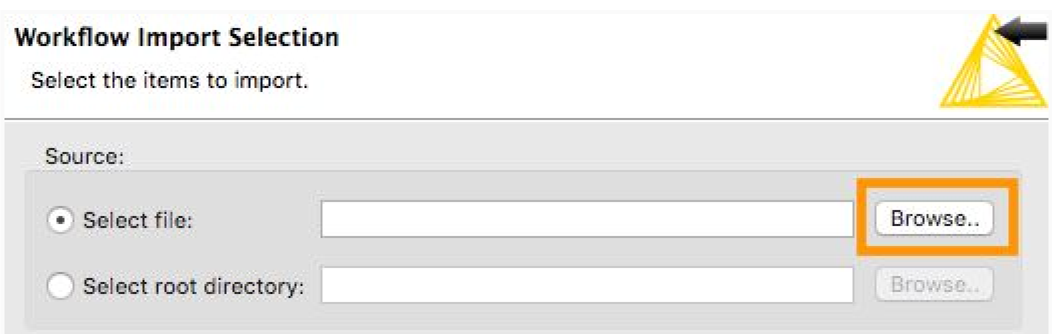
The completed KNIME workflows for this course are available here:

KNIME-workflows.zip

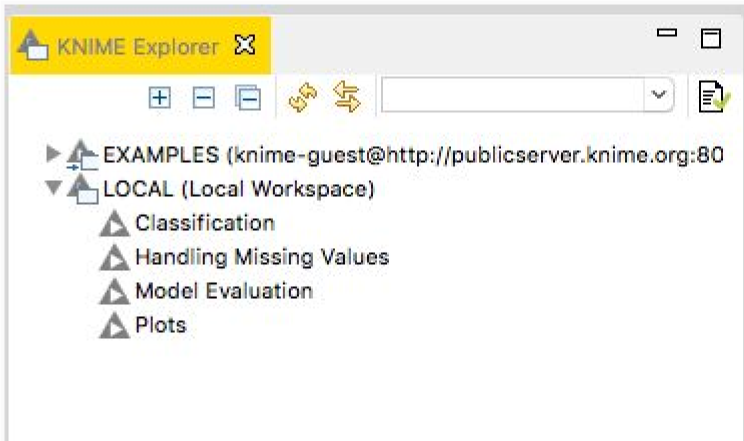
To load these in KNIME, select *File -> Import KNIME Workflow*:



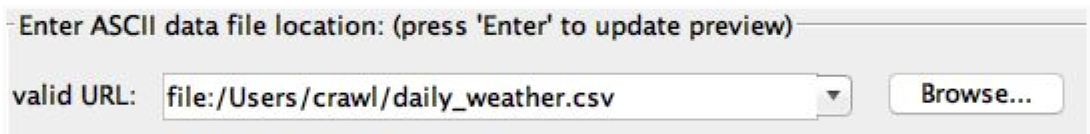
Next, click on the *Browse* button and select the workflow to import:



Finally, click on *Finish.* The workflow will be added to your *LOCAL* Workspace, and you can load it by double-clicking on the name:



Finally, double-click on the *File Reader* Node and change the *valid URL* parameter to the location of *daily_weather.csv* on your computer:



Comparing Classification Results for KNIME and Spark

You may have noticed that the classification results from KNIME and Spark MLlib decision trees are not exactly the same. This document describes the differences in the setup that can lead to this disparity.

Random Seed for Partitioning

In the step to partition data into training and test sets, random sampling is used in both KNIME and Spark MLlib. However, note that different random seed values are used. A value of 12345 is used for KNIME, and 13234 is used for Spark. Different seed values will generate different random number sequences. Since the selection of which samples go into the training vs. the test partition is based on these random number sequences, the contents of the training and test datasets will be different with different seed values. Consequently, different training and test sets will change the performance results of the classifier. Cross validation is a common approach to address this variability in performance with a single partitioning of the data.

Partitioning into Training and Test Sets

You also may have noticed that the sizes of the training and test sets are different between KNIME and Spark. This is due to the different sampling methods that they use to partition data.

Spark uses a sampling method that does not generate a fixed sample size. So if we specify that the test size should be 20% of the available data, the resulting test size is only approximately 20%, not necessarily exactly.

KNIME uses a different way to randomly select samples to be placed in either training and test set. Due to the different sampling techniques, the contents of the training and test sets are different for KNIME and Spark MLlib.

Decision Tree

KNIME and Spark MLlib use different methods to constrain the complexity of the decision tree model. In both, the minimum number of samples in a node can be specified as a stopping criterion for growing the tree. If the number of samples reaches this threshold, the node is not split.

In Spark, you can also specify the maximum depth of the tree as another stopping criterion (the maxDepth parameter). KNIME does not have this option in its Decision Tree Learner node.

KNIME does have an option for pruning the tree, however. The default setting in the Decision Tree Learner node uses ‘Reduced error pruning’. This prunes the tree by replacing a node with the class of the majority of the samples in that node, if doing so does not decrease the accuracy of the classifier. Spark’s DecisionTreeClassifier method does not have an option to prune the tree.

Performance Results

As we can see, there are several differences in the setup for a decision tree classifier in KNIME and Spark MLlib. These differences result in different trained models, and consequently, different classification performance numbers.

Evaluation of Decision Tree in KNIME

Learning Objectives

At the end of this activity, you will be able to perform the following operations in KNIME:

1. Create and interpret a confusion matrix for a decision tree
2. Determine the accuracy rate of a decision tree model
3. Use highlighting to analyze classification errors

Problem Description

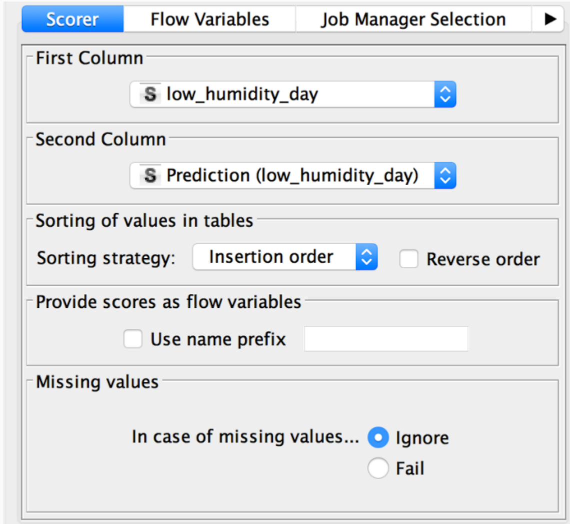
With the decision tree classifier built, we now need to evaluate its performance.

Steps

Generate a Confusion Matrix and Determine Accuracy Rate

A confusion matrix shows the type of errors and correct classifications that a classifier makes. It can be generated using a **Scorer** node.

1. Open the Decision Tree Workflow that you created from the Classification Hands-On reading.
2. Connect a **Scorer** node to the existing **Decision Tree Predictor**.
3. The Scorer Configure Dialog should look like this by default. Click OK.



Execute and view the **Scorer** node. It shows the confusion matrix, along with the accuracy of the prediction. Here you should see an accuracy rate of 80.282% if you followed all the hands-on instructions.

low_humidity_day \ Prediction (low_humidity_day)	humidity_low	humidity_not_low
humidity_low	76	24
humidity_not_low	18	95

Correct classified: 171

Wrong classified: 42

Accuracy: 80.282 %

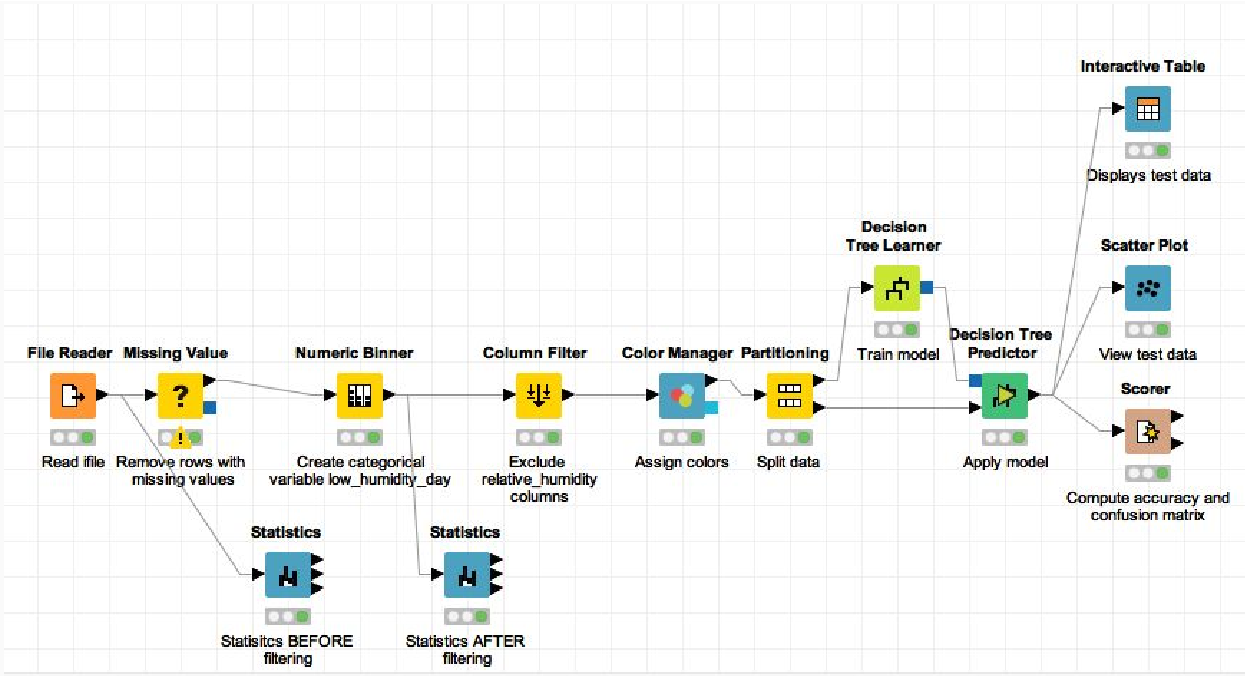
Error: 19.718 %

Cohen's kappa (κ) 0.603

From the confusion matrix, we see the following:

- There are 213 samples in the test data set (the sum of all the values in the confusion matrix)
- 76 humidity_low samples with were correctly classified
- 95 humidity_not_low samples were correctly classified
- The accuracy rate is (76 + 95) / 213 = 171 / 213 = 80.282%
- 24 humidity_low samples were incorrectly classified as humidity_not_low
- 18 humidity_not_low samples were incorrectly classified as humidity_low
- The error rate is (24 + 18) / 213 = 42 / 213 = 19.718%

Use Highlighting and Scatter Plot to Analyze Classification Errors



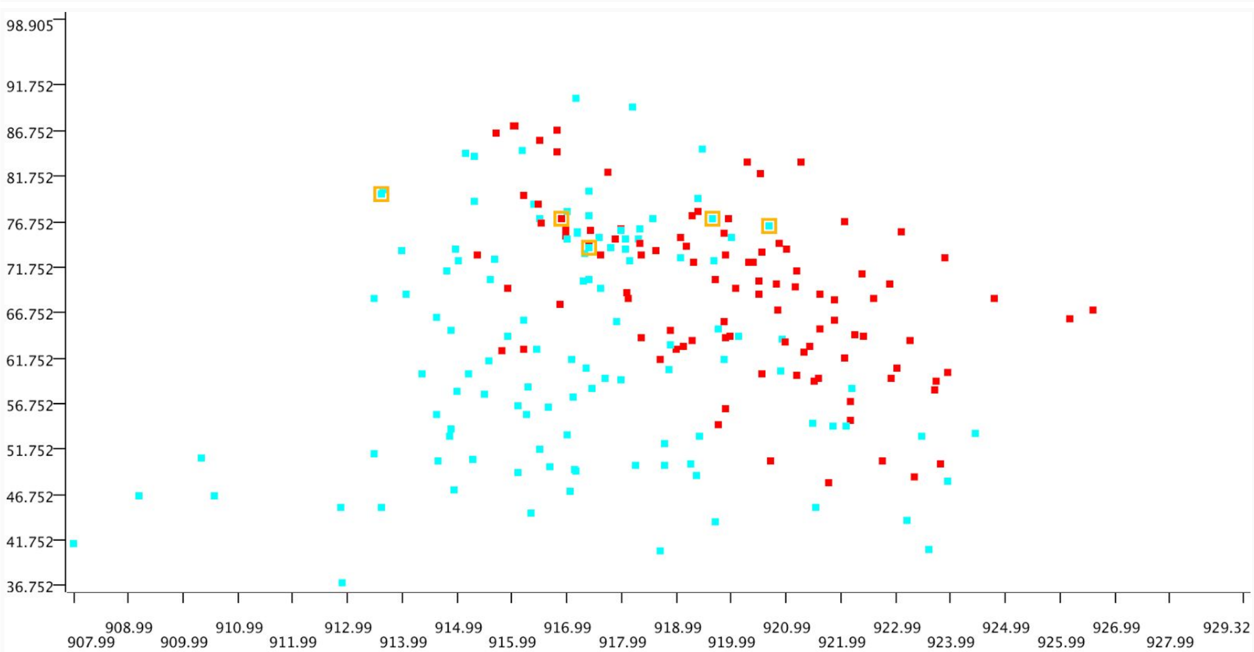
A good way to enhance analysis of incorrect predictions is to visualize them. This can be accomplished using a feature called **hiliting**, and viewing the data in a **Scatter Plot** node.

1. Connect an **Interactive Table** node to the **Decision Tree Predictor**.
2. Execute and view this **Interactive Table** to see the input values for each sample (row), along with the ACTUAL/TRUE low_humidity_day value and the PREDICTED low_humidity_day value. The red and blue squares next to the Row ID color-codes the actual/true label (low or not). You can use this table to analyze samples whose true value differs from the predicted value (incorrect prediction).
3. Connect a **Scatter Plot** node to the **Decision Tree Predictor**.
4. Execute and view the **Scatter Plot** node, and place the window side-by-side with the **Interactive Table** window.
5. Go through the the table looking for rows with predictions that are different from the true value.
6. When you find such a row, click anywhere on that row. At the top of the window click **Hilite > Hilite Selected**. This will make that row yellow in the table and in the Scatter Plot. It may be easiest to use the up and down arrow keys to navigate the rows of the table. In this example, we are just going to highlight the first 5 misclassifications.
7. Do this for any row with a misclassification. This allows you to pinpoint the misclassified samples and analyze them further. Analyzing the misclassified samples can bring insight into how to improve model performance. For example, if many samples with

avg_temperature_9am between 60 and 70 degrees are misclassified, this suggests that more samples with these values for avg_temperature_9am are needed to train the model.

File Hilite Navigation View Output

Row ID	D air_pr...	D air_te...	D avg_w...	D avg_w...	D max_...	D max_...	D rain_a...	D rain_d...	S low_humidi...	S Prediction (...
10	919.65	77.036	70.6	3.825	85.5	4.765	0	0	humidity_not_low	humidity_low
15	922.383	70.865	36.174	1.847	58.429	2.529	0	0	humidity_low	humidity_low
17	916.915	77.019	234.539	2.275	229.474	2.907	0	0	humidity_low	humidity_not_low
31	916.28	55.544	174.6	2.595	191.7	3.378	0	0	humidity_not_low	humidity_not_low
33	918.37	63.914	53.7	14.451	72.1	17.045	0	0	humidity_low	humidity_low
38	914.66	50.36	177.6	8.523	186.3	10.021	0	20	humidity_not_low	humidity_not_low
43	914.9	64.688	174.6	5.659	184.5	6.867	0	0	humidity_not_low	humidity_not_low
46	917.9	65.66	183.7	6.062	188.5	6.509	0.021	319	humidity_not_low	humidity_not_low
47	916.05	87.188	210	1.566	109.3	2.595	0	0	humidity_low	humidity_low
50	921.9	65.876	199	4.496	207.2	5.324	0	0	humidity_low	humidity_low
51	918.58	76.982	150.7	1.32	182	1.946	0	0	humidity_not_low	humidity_not_low
63	917.02	53.258	180.2	3.132	216.6	4.452	0	0	humidity_not_low	humidity_not_low
65	921.29	83.228	162.6	2.662	194	3.557	0	0	humidity_low	humidity_low
66	922.06	61.754	56.2	10.424	77	13.422	0	0	humidity_low	humidity_low
70	917.763	82.064	110.258	2.235	129.78	3.206	0	0	humidity_low	humidity_low
76	922.07	76.742	165.736	3.771	205.67	6.004	0	0	humidity_low	humidity_low
79	915.57	61.34	174.3	4.899	180.6	5.503	0	0	humidity_not_low	humidity_not_low
81	917.46	58.298	182.3	5.906	190.3	6.935	0	0	humidity_not_low	humidity_not_low
84	920.93	63.824	196.4	7.65	204	8.567	0	0	humidity_not_low	humidity_not_low
88	915.914	69.389	46.736	10.977	64.645	14.298	0	9.397	humidity_low	humidity_low
91	915.32	83.84	219.9	1.521	245	2.058	0	0	humidity_not_low	humidity_not_low
96	921.874	68.094	70.909	2.147	96.095	2.891	0	0	humidity_low	humidity_low
98	916.52	51.638	190.8	7.404	198	8.389	0.37	2,170	humidity_not_low	humidity_not_low
103	921.5	54.536	220	5.011	234.9	6.442	0	0	humidity_not_low	humidity_not_low
105	917.4	73.868	181.7	4.138	189.8	4.765	0.05	460	humidity_not_low	humidity_low
106	914.95	47.21	183.9	7.628	191.2	8.612	20.02	11,650	humidity_not_low	humidity_not_low
112	916.45	62.654	193.7	3.534	208.8	4.429	0	0	humidity_not_low	humidity_not_low
120	920.7	76.172	190.9	6.263	199.6	7.27	0	0	humidity_not_low	humidity_low
122	918.32	74.732	239.5	2.058	273.4	3.02	0	0	humidity_not_low	humidity_not_low
124	913.61	79.628	180.5	3.266	187.9	3.758	0	0	humidity_not_low	humidity_low



Save Your Workflow

Save your workflow using <control>-s on Windows or <command>-s on Mac, or selecting File>Save or File>Save As.

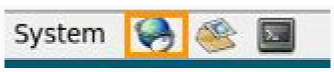
Evaluation of Decision Tree in Spark

By the end of this activity, you will be able to perform the following in Spark:

1. Determine the accuracy of a classifier model
2. Display the confusion matrix for a classifier model

In this activity, you will be programming in a Jupyter Python Notebook. If you have not already started the Jupyter Notebook server, see the instructions in the Reading *Instructions for Starting Jupyter*.

Step 1. **Open Jupyter Python Notebook.** Open a web browser by clicking on the web browser icon at the top of the toolbar:



Navigate to *localhost:8889/tree/Downloads/big-data-4*:

Open the model evaluation notebook by clicking on *model-evaluation.ipynb:*

Step 2. **Load predictions.** Execute the first cell to load the classes used in this activity:

```
In [1]: from pyspark.sql import SQLContext
        from pyspark.ml.evaluation import MulticlassClassificationEvaluator
        from pyspark.mllib.evaluation import MulticlassMetrics
```

Execute the next cell to load the predictions CSV file that we created at the end of the Week 3 Hands-On *Classification in Spark* into a DataFrame:

```
In [2]: sqlContext = SQLContext(sc)
        predictions = sqlContext.read.load('file:///home/cloudera/Downloads/big-data-4/predictions.csv',
                                  format='com.databricks.spark.csv',
                                  header='true',inferSchema='true')
```

Step 3. **Compute accuracy.** Let's create an instance of *MulticlassClassificationEvaluator* to determine the accuracy of the predictions:

```
In [3]: evaluator = MulticlassClassificationEvaluator(
            labelCol="label", predictionCol="prediction", metricName="precision")
```

The first two arguments specify the names of the label and prediction columns, and the third argument specifies that we want the overall precision.

We can compute the accuracy by calling *evaluate()*:

```
In [4]: accuracy = evaluator.evaluate(predictions)
        print("Accuracy = %g " % (accuracy))

        Accuracy = 0.809524
```

Step 4. **Display confusion matrix.** The *MulticlassMetrics* class can be used to generate a confusion matrix of our classifier model. However, unlike *MulticlassClassificationEvaluator*, *MulticlassMetrics* works with RDDs of numbers and not DataFrames, so we need to convert our *predictions* DataFrame into an RDD.

If we use the *rdd* attribute of *predictions,* we see this is an *RDD* of *Rows:*

```
In [5]: predictions.rdd.take(2)

Out[5]: [Row(prediction=1.0, label=1.0), Row(prediction=1.0, label=1.0)]
```

Instead, we can map the RDD to *tuple* to get an RDD of numbers:

```
In [6]: predictions.rdd.map(tuple).take(2)

Out[6]: [(1.0, 1.0), (1.0, 1.0)]
```

Let's create an instance of *MulticlassMetrics* with this RDD:

```
In [7]: metrics = MulticlassMetrics(predictions.rdd.map(tuple))
```

NOTE: the above command can take longer to execute than most Spark commands when first run in the notebook.

The *confusionMatrix()* function returns a Spark *Matrix*, which we can convert to a Python Numpy array, and transpose to view:

```
In [8]: metrics.confusionMatrix().toArray().transpose()

Out[8]: array([[ 87.,  26.],
               [ 14.,  83.]])
```