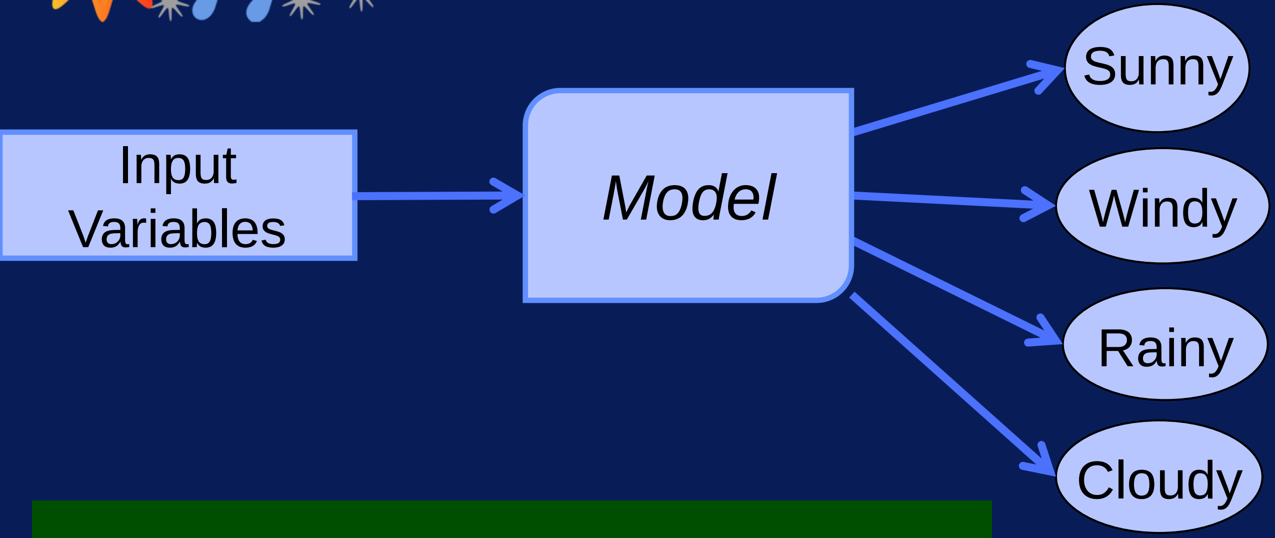
Classification Overview

After this video you will be able to..

- Define what classification is
- Discuss whether classification is supervised or unsupervised
- Describe how binomial classification differs from multinomial classification

Classification

Target variable is categorical



Goal:
Given input variables, predict category

Data for Classification

Input Variables

Target Variable

Temperature	Humidity	Wind Speed	Weather
79	48	2.7	Sunny
60	80	3.8	Rainy
68	45	17.9	Windy
57	77	4.2	Cloudy

Classification is Supervised

Target

Label

Output

Class Variable

Class

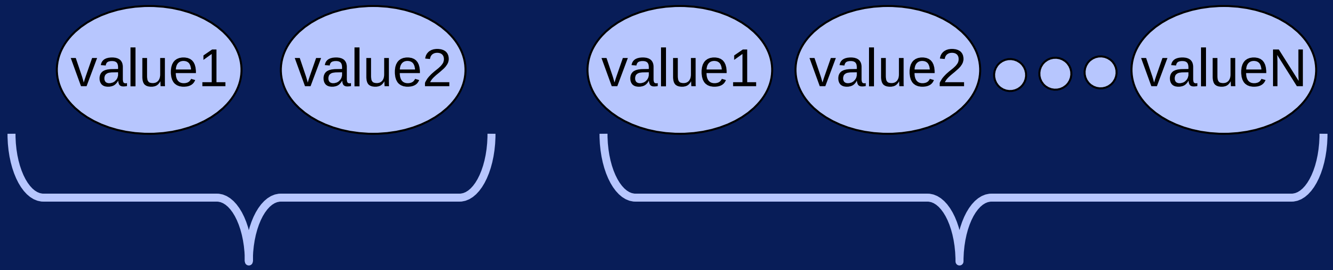
Category

Temperature	Humidity	Wind Speed	Weather
79	48	2.7	Sunny
60	80	3.8	Rainy
68	45	17.9	Windy
57	77	4.2	Cloudy

Types of Classification

Binary Classification

Multi-class Classification



Target has two values

Target has > 2 values

Classification Examples

Binary Classification

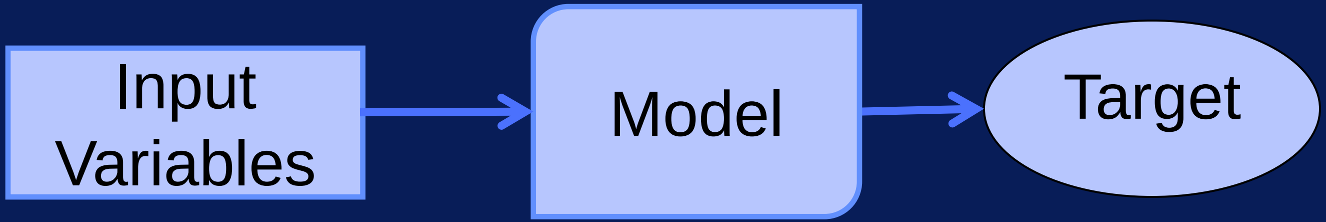
- Will it rain tomorrow or not?
- Is this transaction legitimate or fraudulent

Multi-Class Classification

- What type of product will this customer buy?
- Is this tweet positive, negative, or neutral

Classification Main Points

- Predict category from input variables
- Classification is a supervised task
- Target variable is categorical



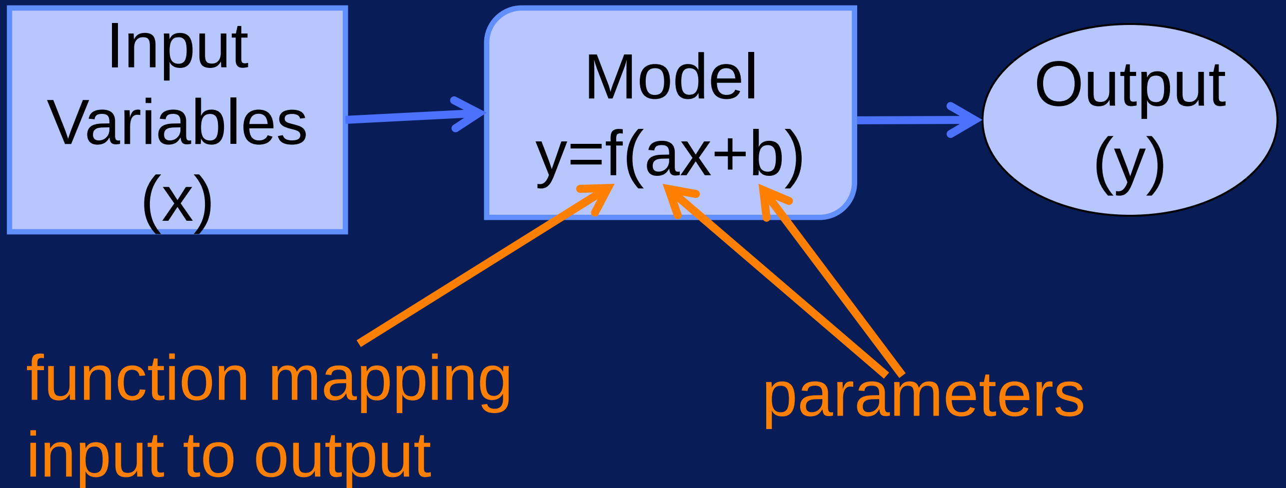
Building and Applying a Classification Model

After this video you will be able to..

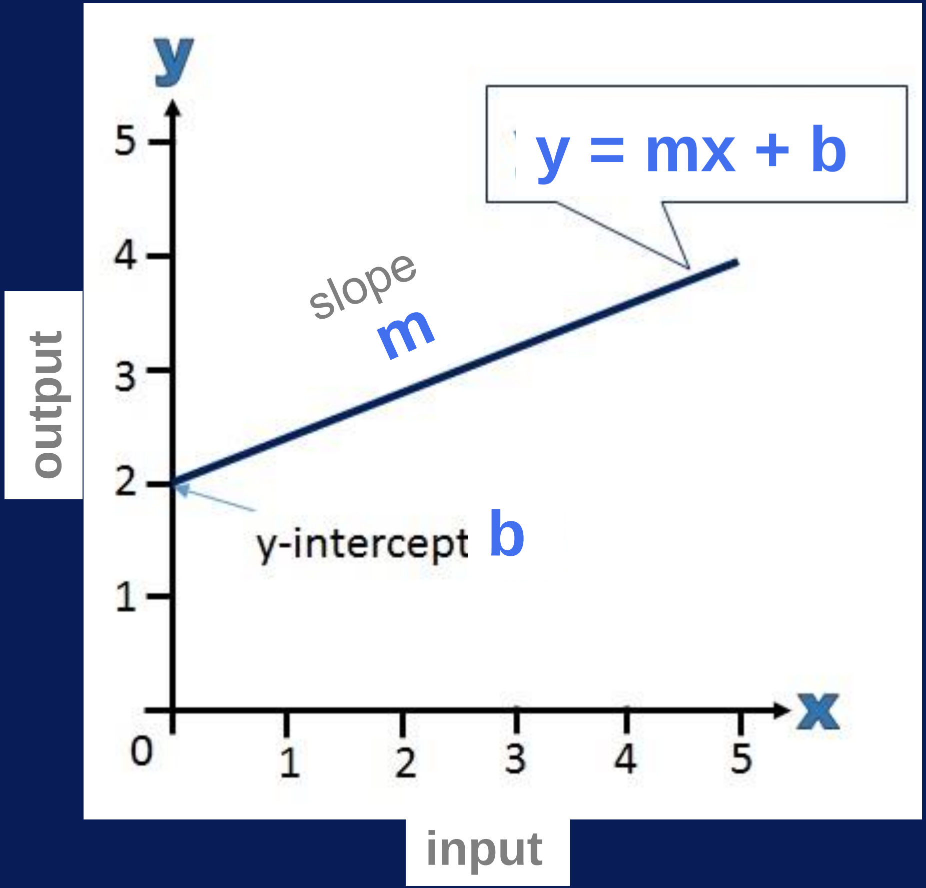
- Discuss what building a classification model means
- Explain the difference between building and applying a model
- Summarize why the parameters of a model need to be adjusted

What is a Machine Learning Model?

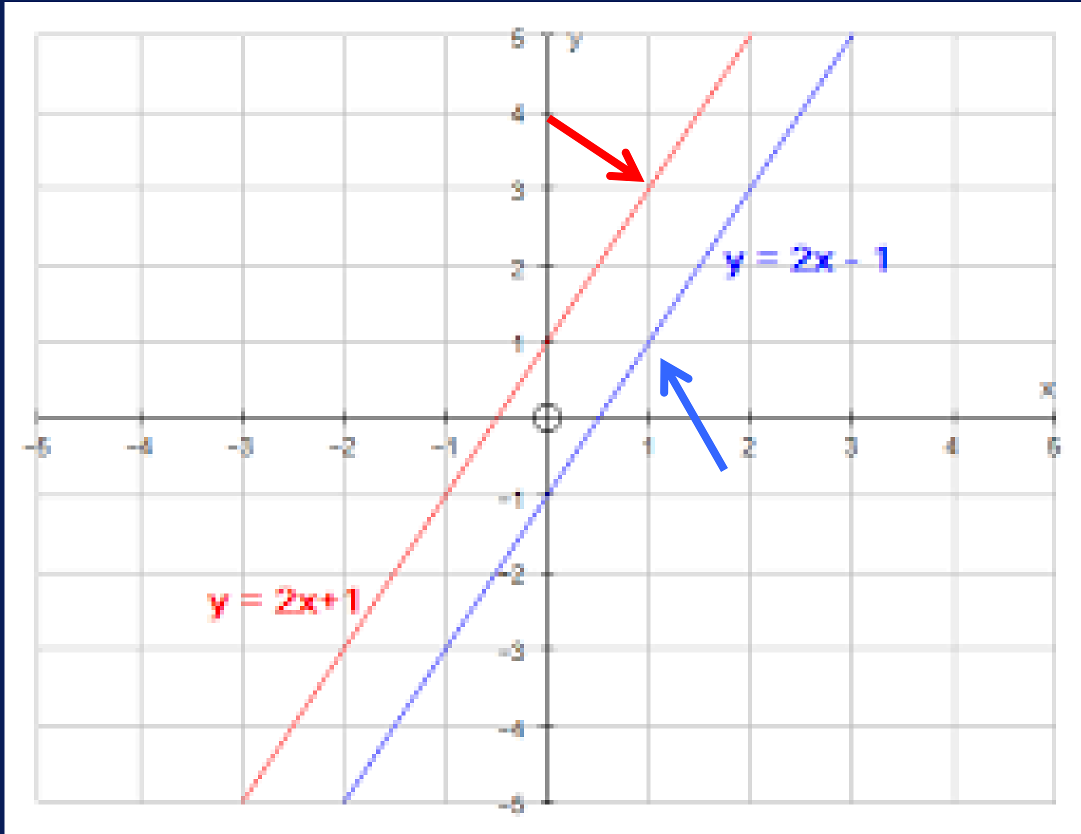
- A mathematical model with parameters that map input to output



Example of Model



Adjusting Model Parameters

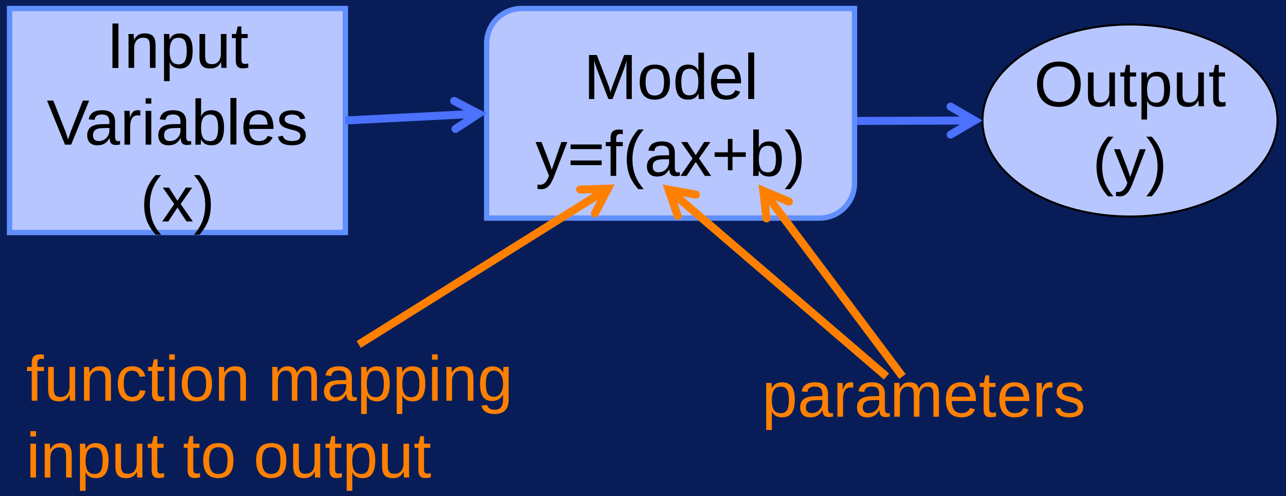


slope m = 2
y-intercept b = -1
x=1 => y=2*1-1=1

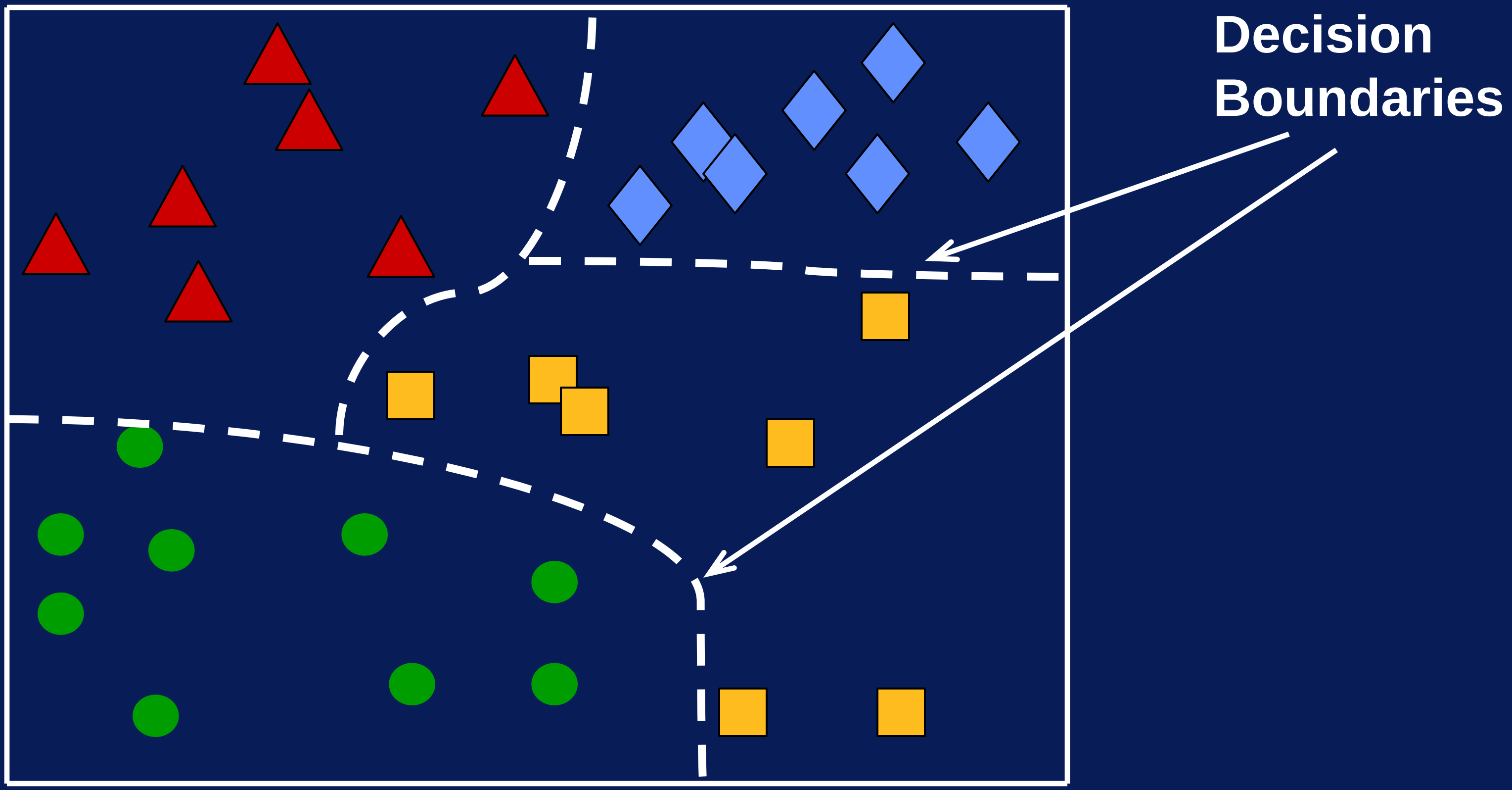
slope m = 2
y-intercept b = +1
x=1 => y=2*1+1= 3

Building Machine Learning Model

Model parameters are adjusted during model training to change input-output mapping.



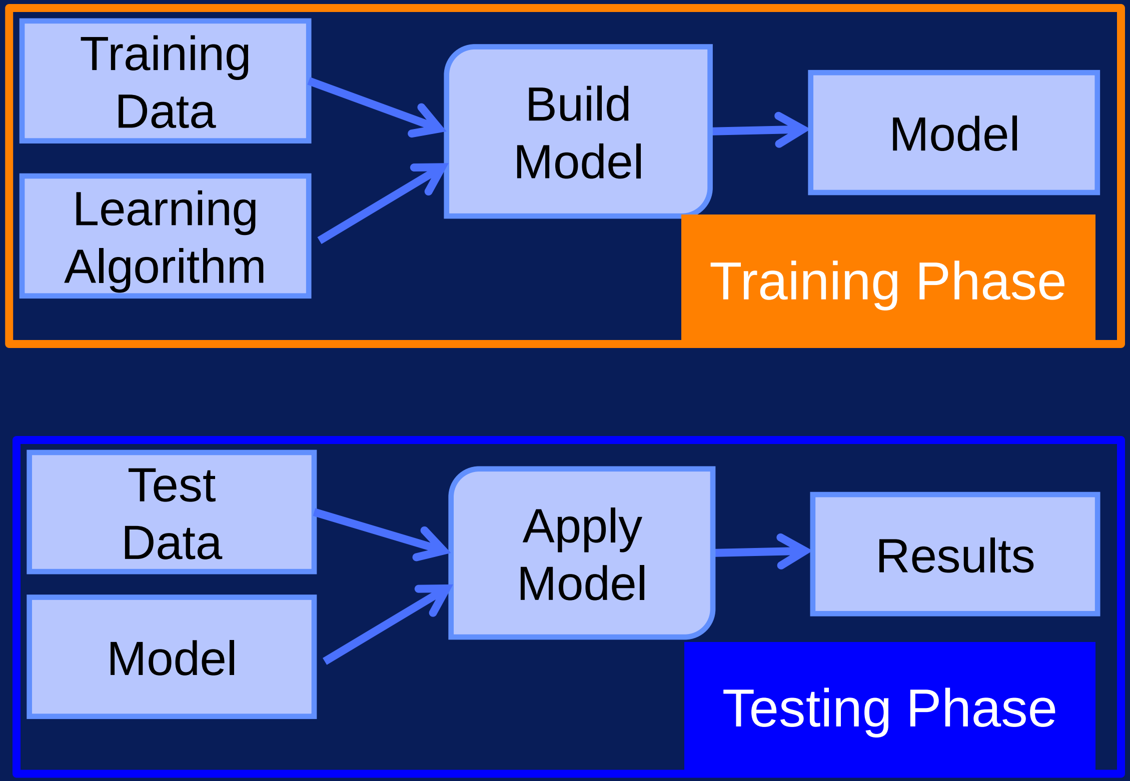
Building Classification Model



Building vs. Applying Model

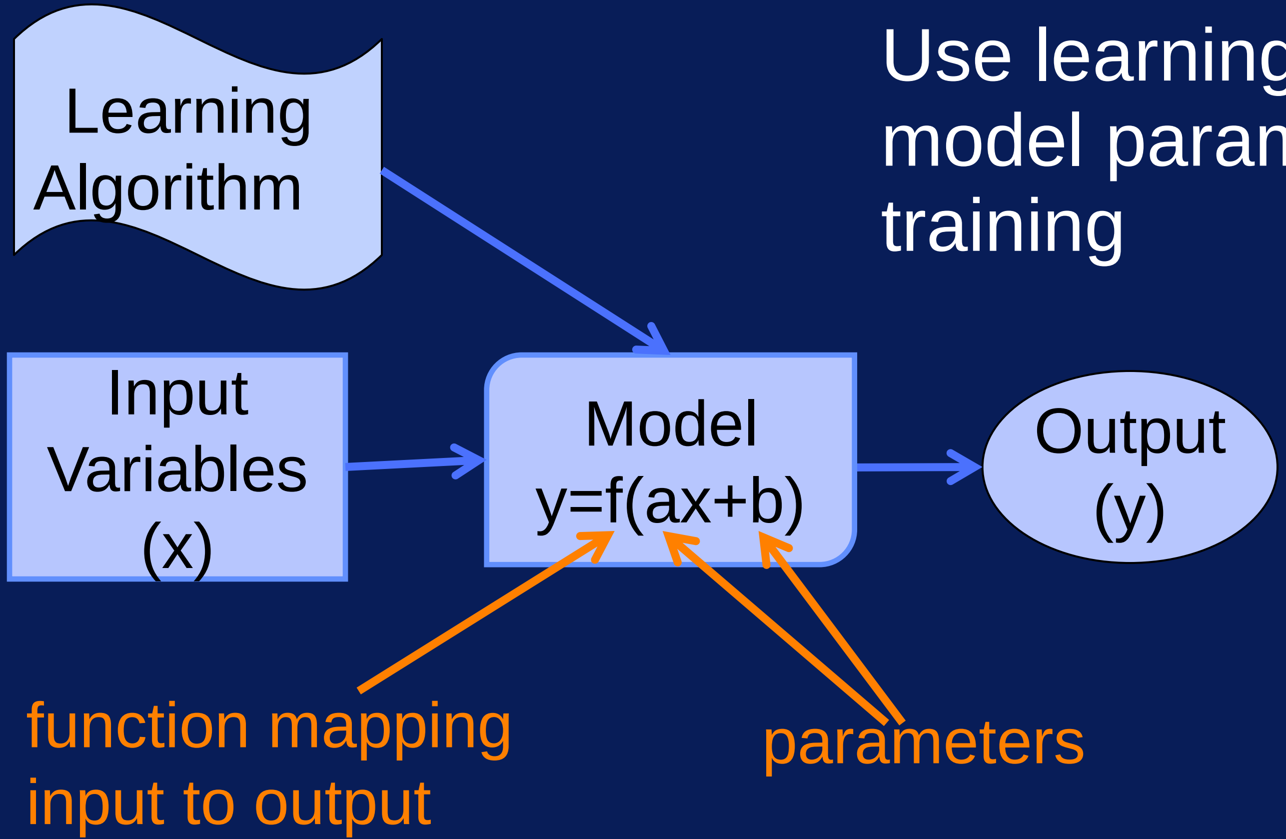
- Training Phase
  - Adjust model parameters
  - Use training data
- Testing Phase
  - Apply learned model
  - Use new data

Building vs. Applying Model



Building a Classification Model

Use learning algorithm to model parameters during training



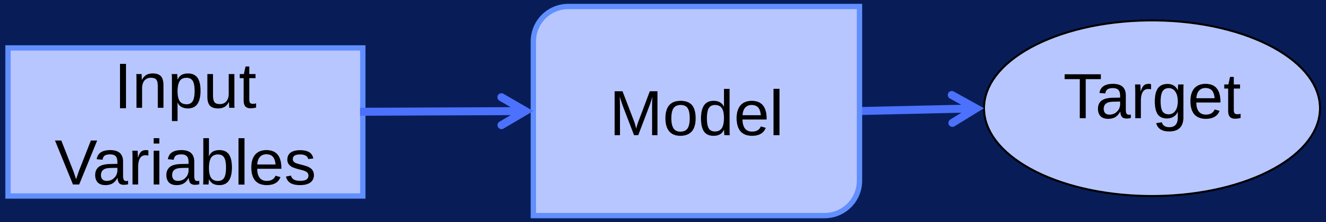
Classification Algorithms Overview

After this video you will be able to..

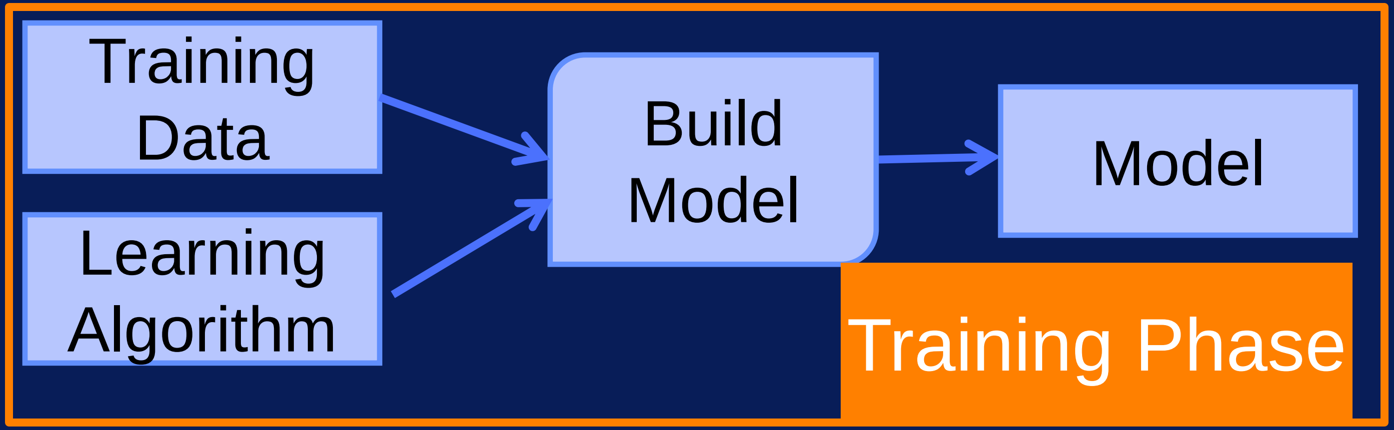
- Describe the goal of a classification algorithm
- Name some common algorithms for classification

Classification

- **Task:** Predict category from input variables
- **Goal:** Match model outputs to targets (desired outputs)



Learning Algorithm



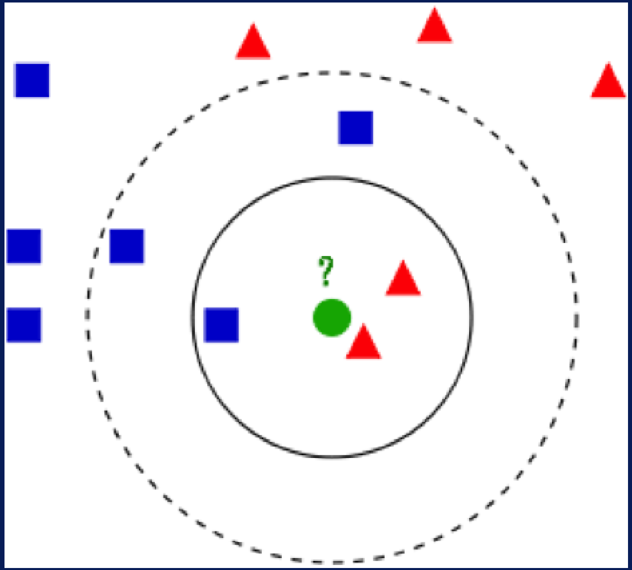
- Learning algorithm used to adjust model's parameters

Classification Algorithms

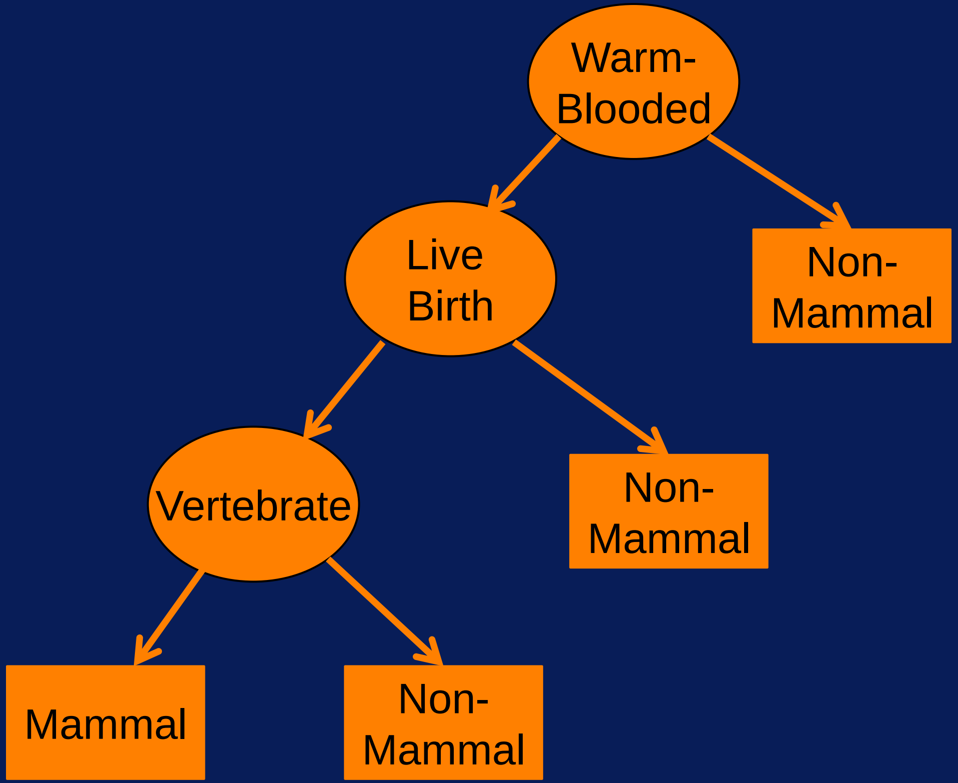
- Common basic classification algorithms
  - kNN
  - Decision tree
  - Naïve bayes

kNN Overview

- Classify sample by looking at its closest neighbors



Decision Tree Overview



- Tree captures multiple classification decision paths

Naïve Bayes Overview

- Probabilistic approach to classification

Bayes' Theorem:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

Classification Algorithms

k Nearest Neighbors

Decision Tree

Naïve Bayes

Many others ...

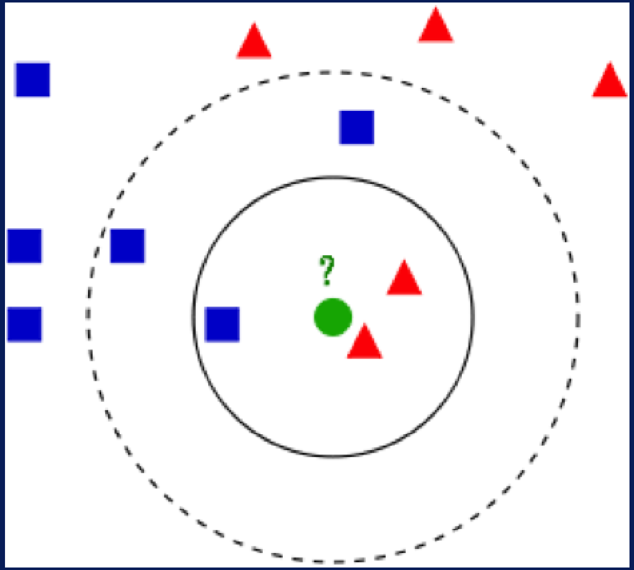
k Nearest Neighbors

After this video you will be able to..

- Describe how kNN is used for classification
- Discuss the assumption behind kNN
- Explain what the ‘k’ stands for in kNN

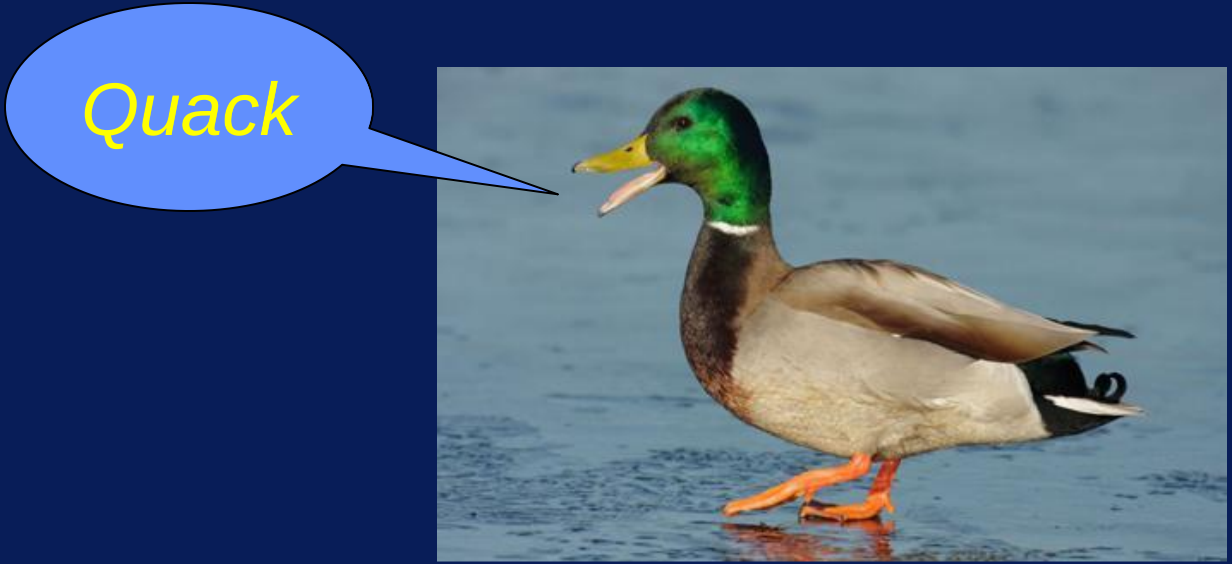
kNN

- Simple classification technique
- Label sample based on its neighbors



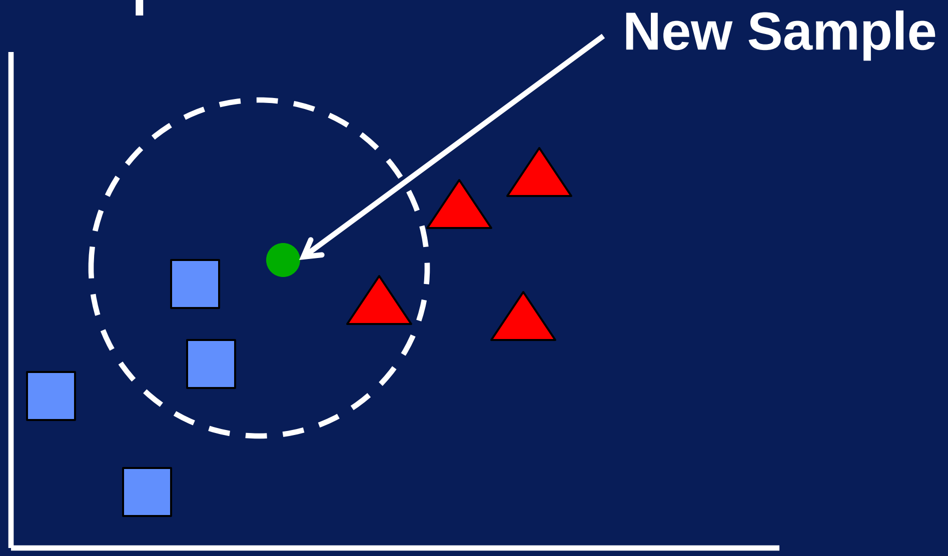
kNN Assumption

- Duck test



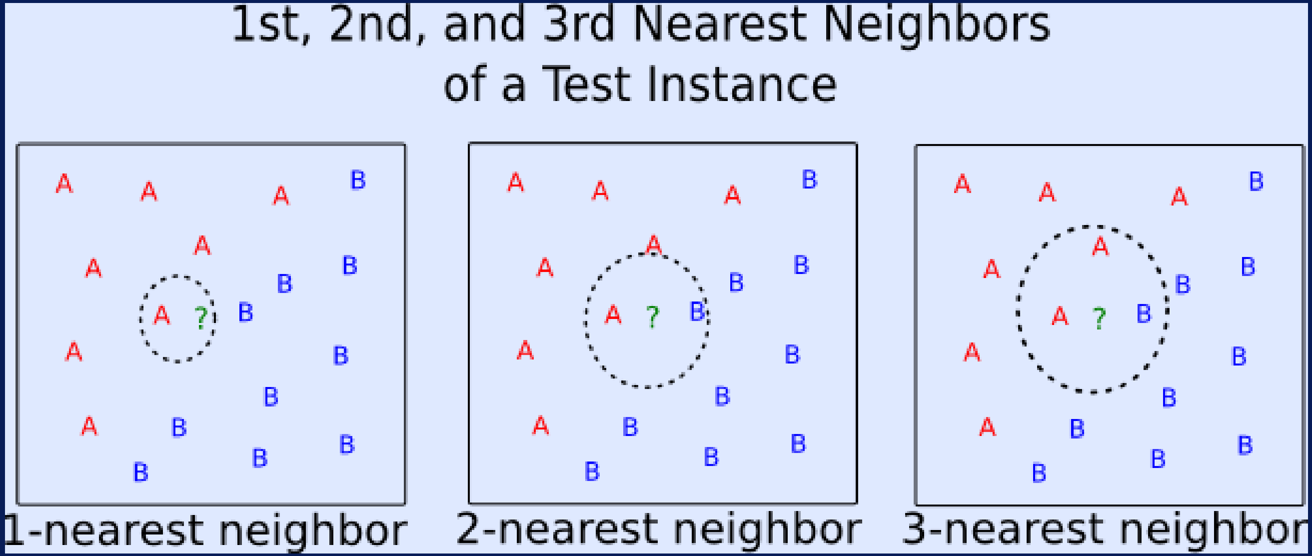
How kNN Works

- Use labels of neighboring samples to determine label for new point



What is k?

- Value of k determines number of closest neighbors to consider

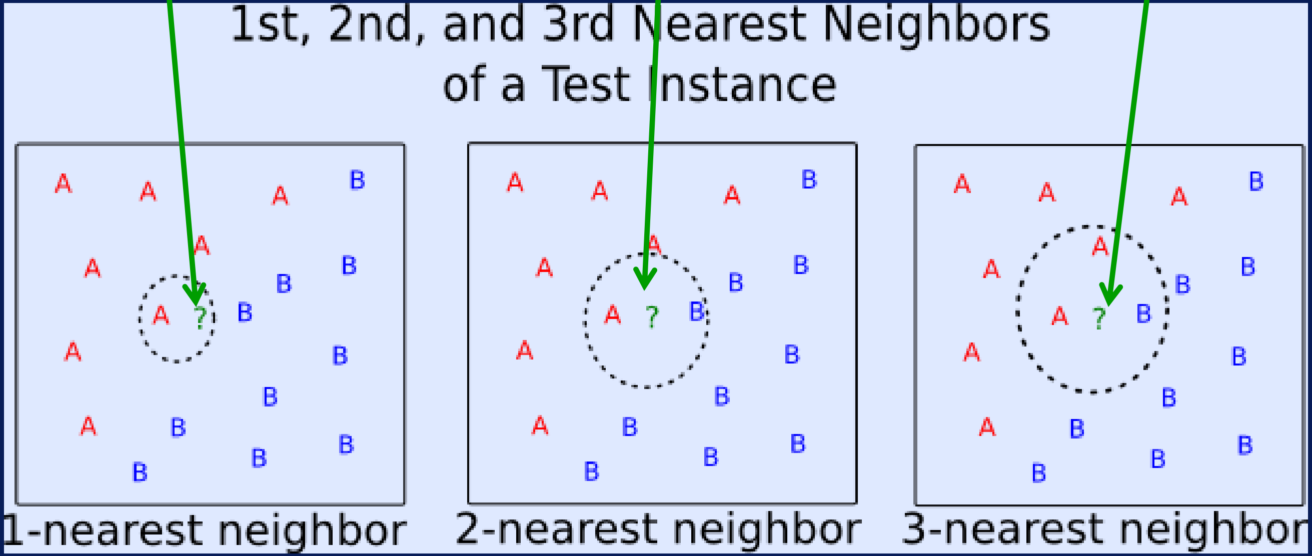


Using k Nearest Neighbors

Label='A' (from neighbor)

Label='B' (random tiebreaker)

Label='A' (majority vote)



Distance Measure

- Need measure to determine "closeness"

kNN Classification

- No separate training phase
- Can generate complex decision boundaries
- Can be slow
  - Distance between new sample and all samples must be computed to classify new sample

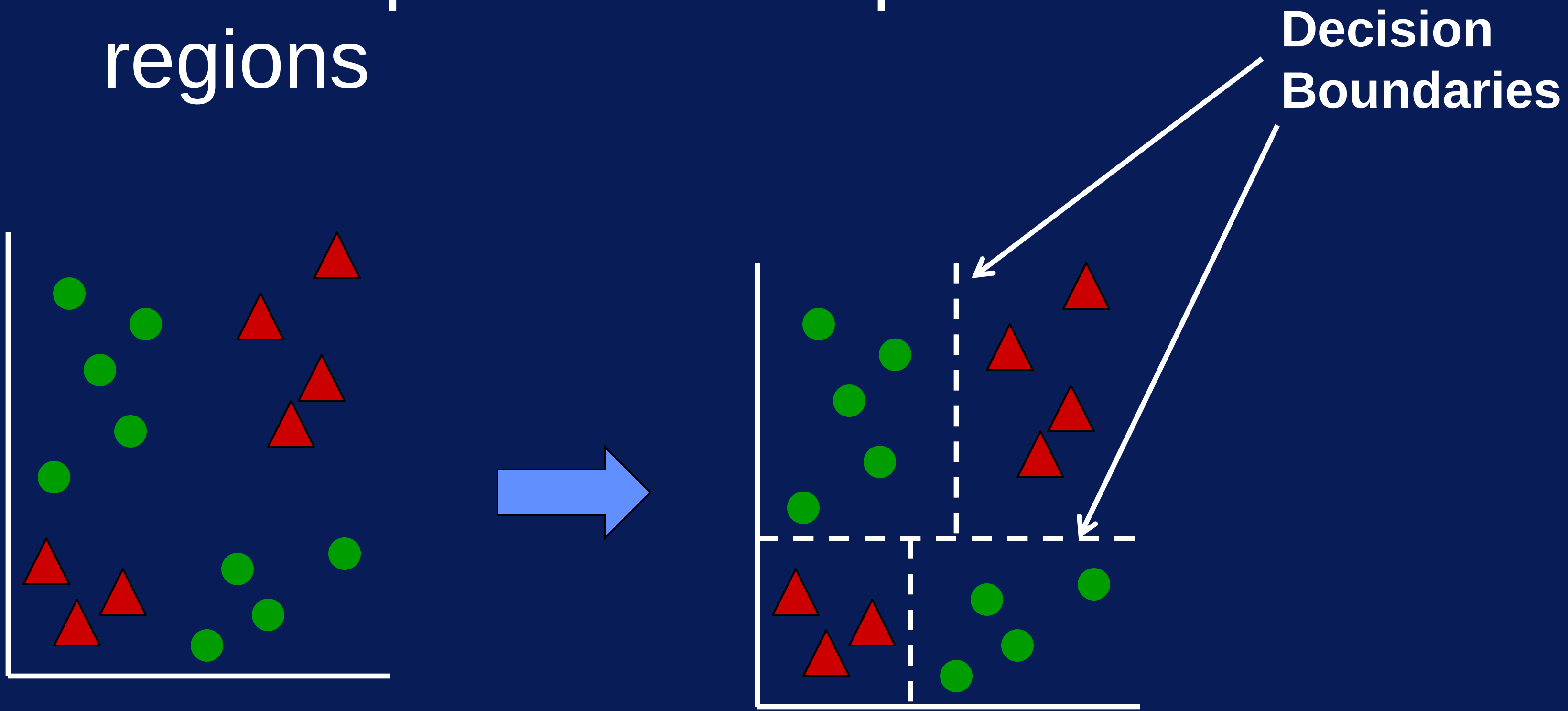
Decision Tree

After this video you will be able to..

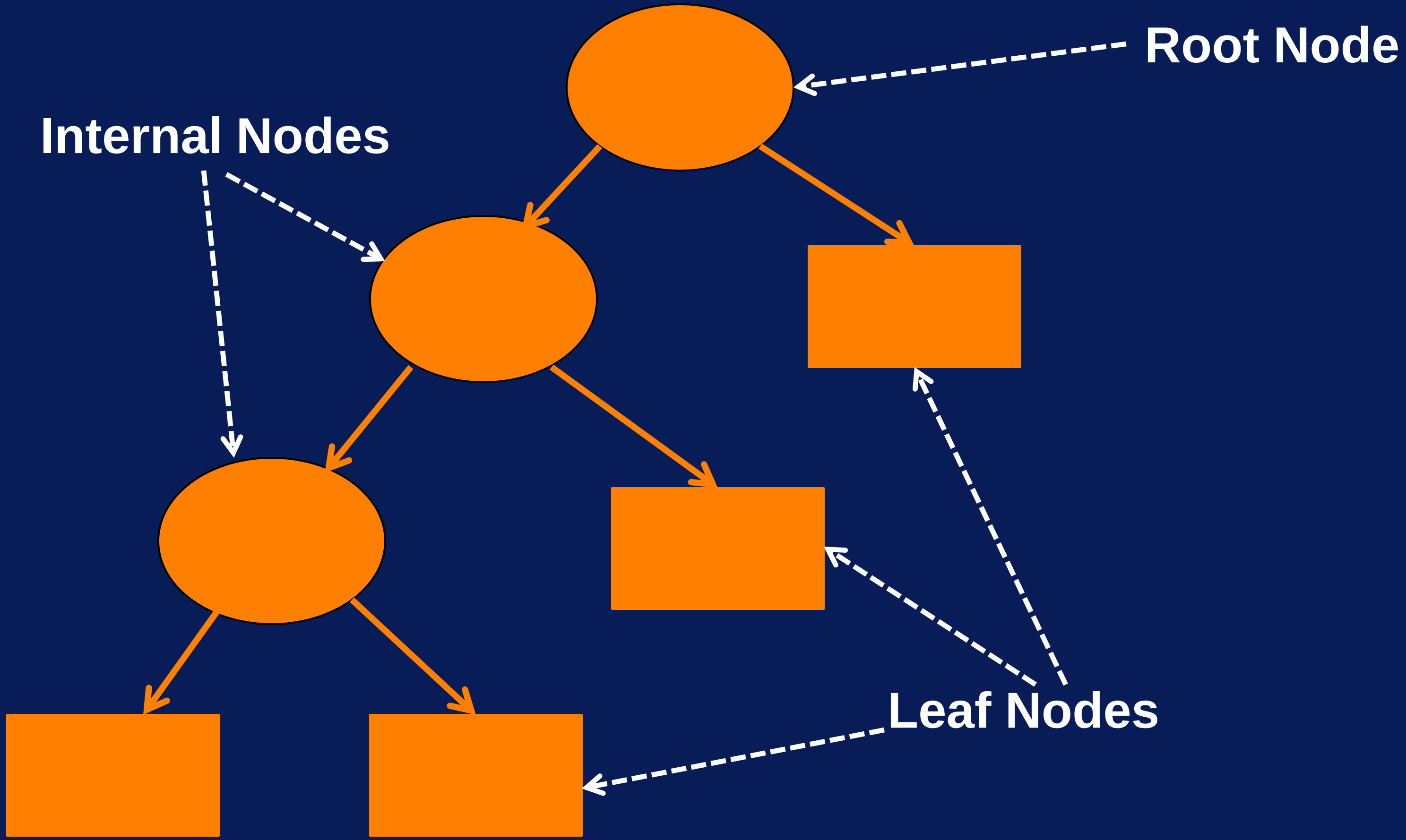
- Explain how a decision tree is used for classification
- Describe the process of constructing a decision tree for classification
- Interpret how a decision tree comes up with a classification decision

Decision Tree Overview

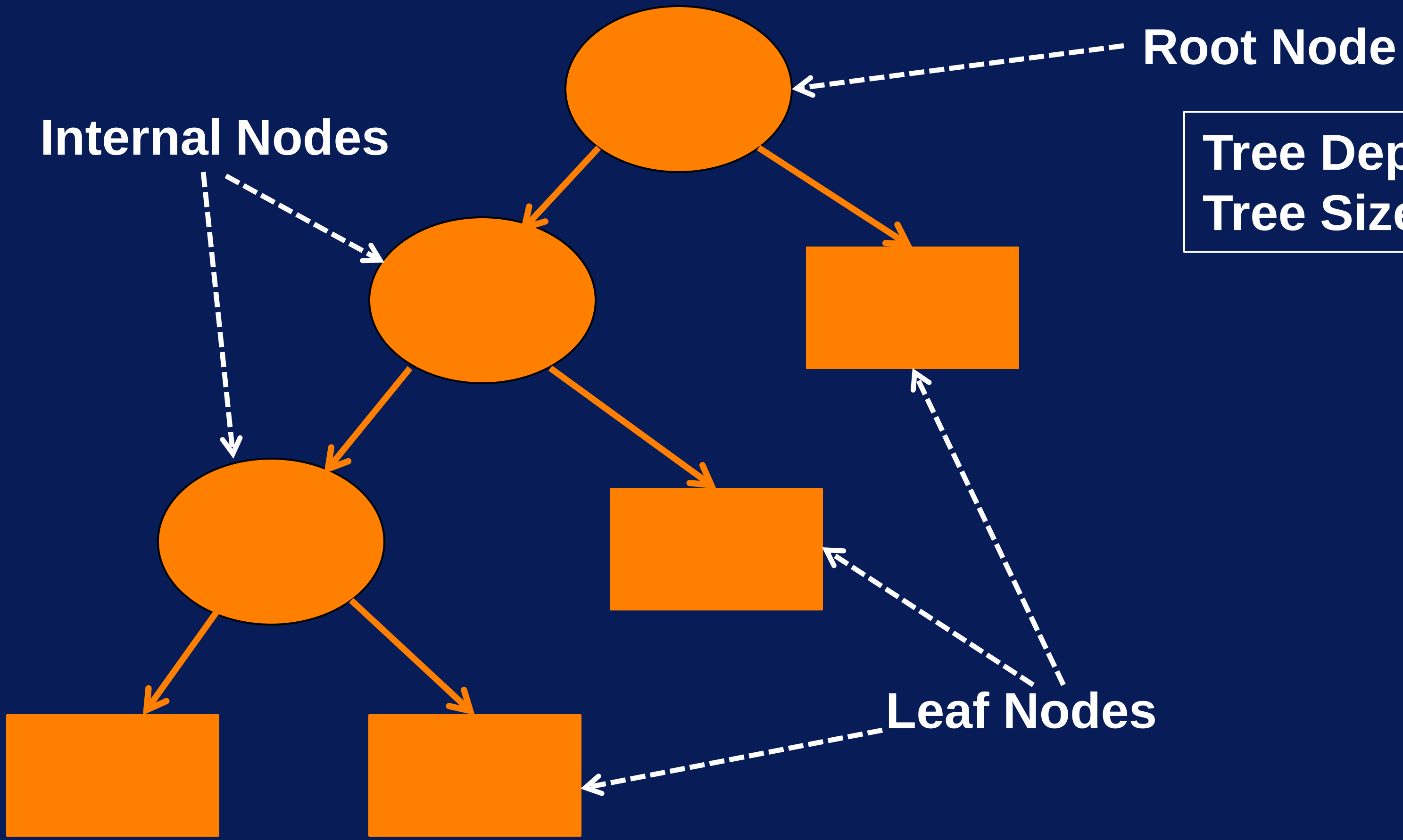
- Idea: Split data into "pure" regions



Classification Using Decision Tree



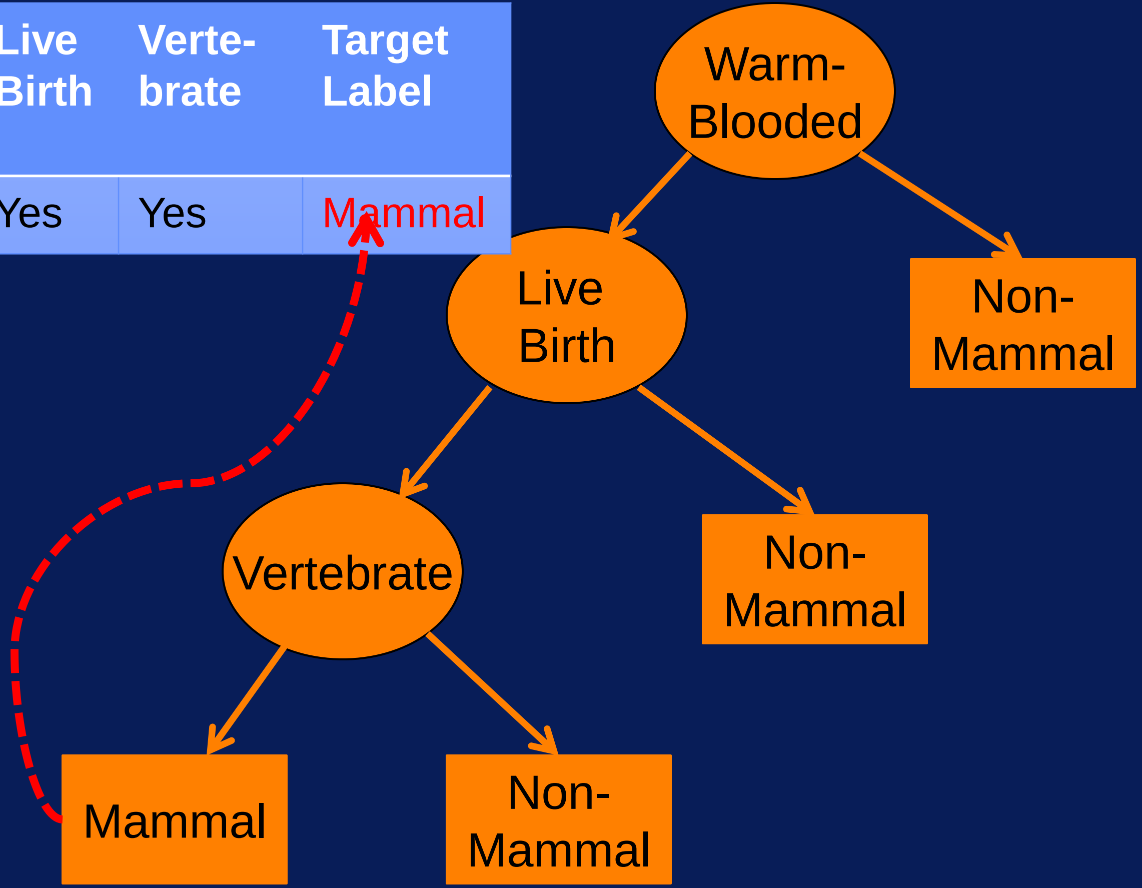
Classification Using Decision Tree



Tree Depth = 3
Tree Size = 6

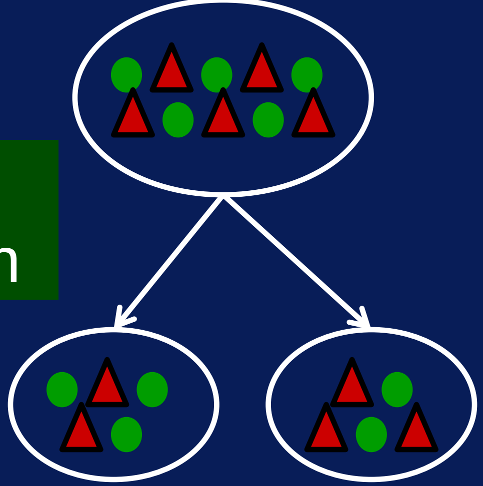
Example Decision Tree

Warm-Blooded	Live Birth	Verte-brate	Target Label
Yes	Yes	Yes	Mammal



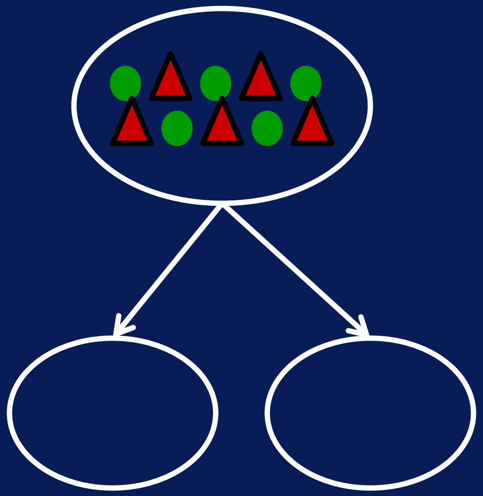
Constructing Decision Tree

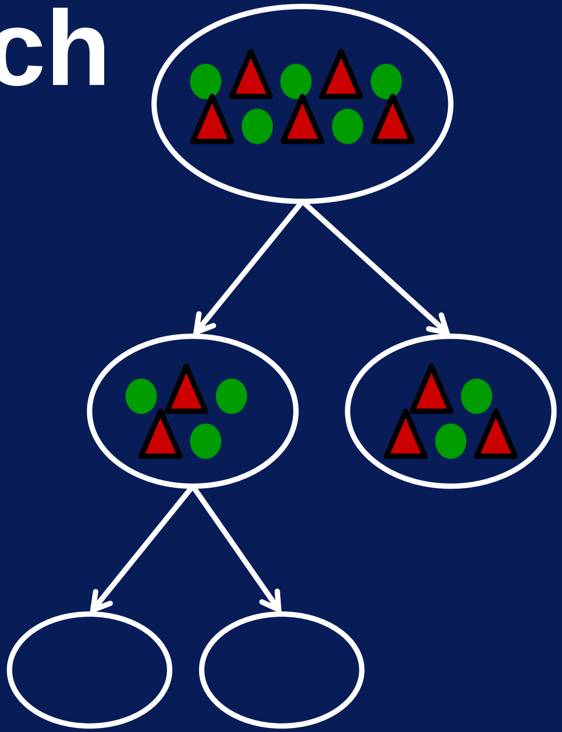
Tree Induction



- Start with all samples at a node.
- Partition samples based on input to create purest subsets.
- Repeat to partition data into successively purer subsets.

Greedy Approach

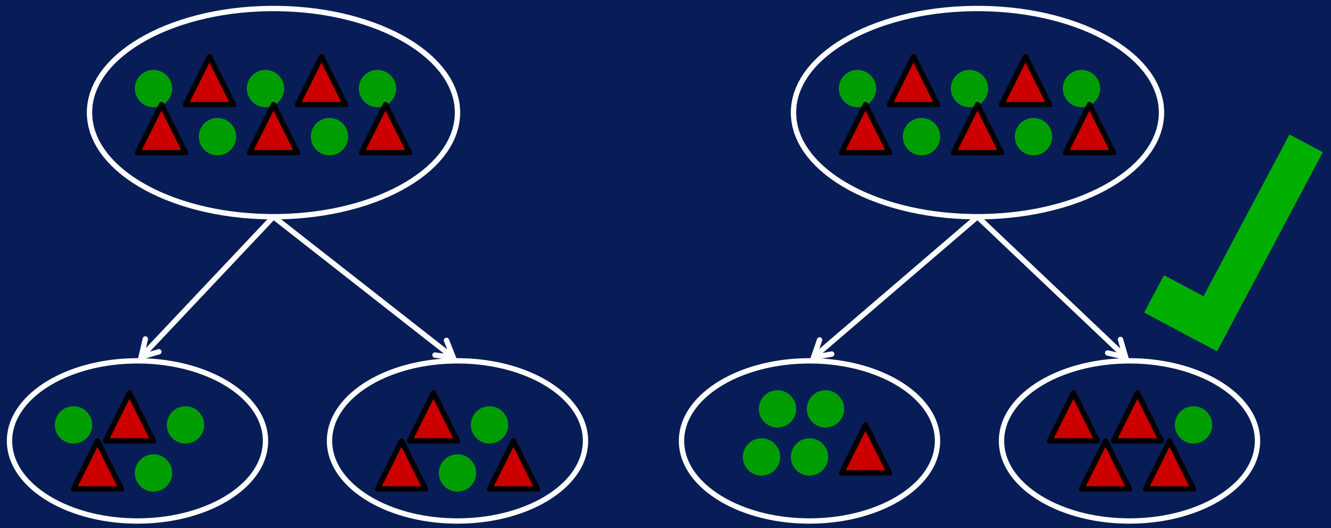




What’s the best way to split the current node?

How to Determine Best Split?

Want subsets to be as homogeneous as possible



Less homogeneous = More pure

More homogeneous = More pure

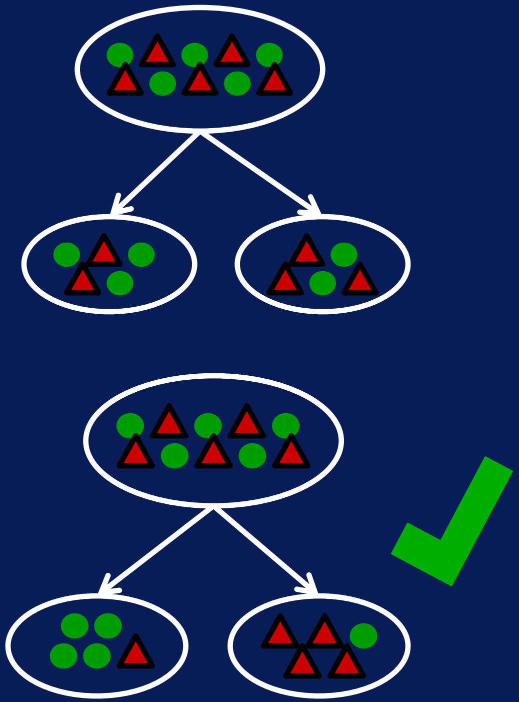
Impurity Measure

- To compare different ways to split data in a node

Higher =
Less pure

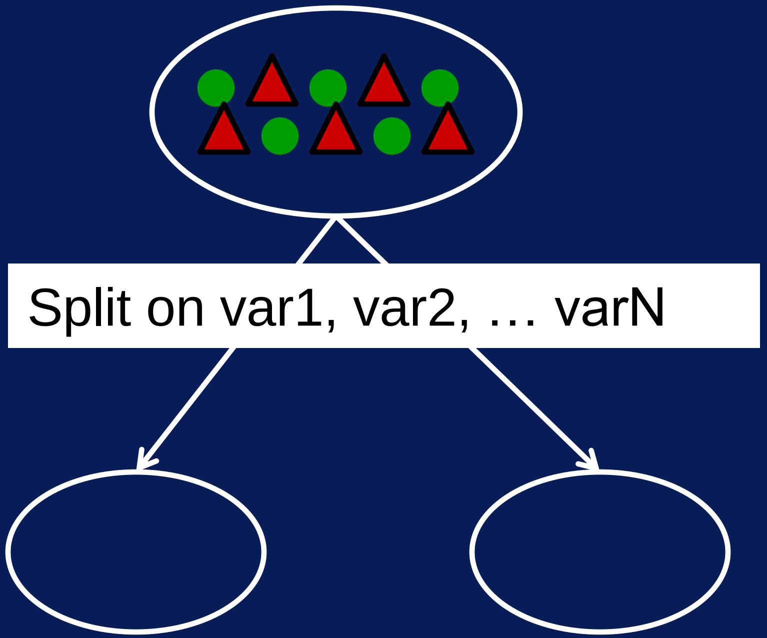
Gini
Index

Lower =
More pure



What Variable to Split On?

- Splits on all variables are tested

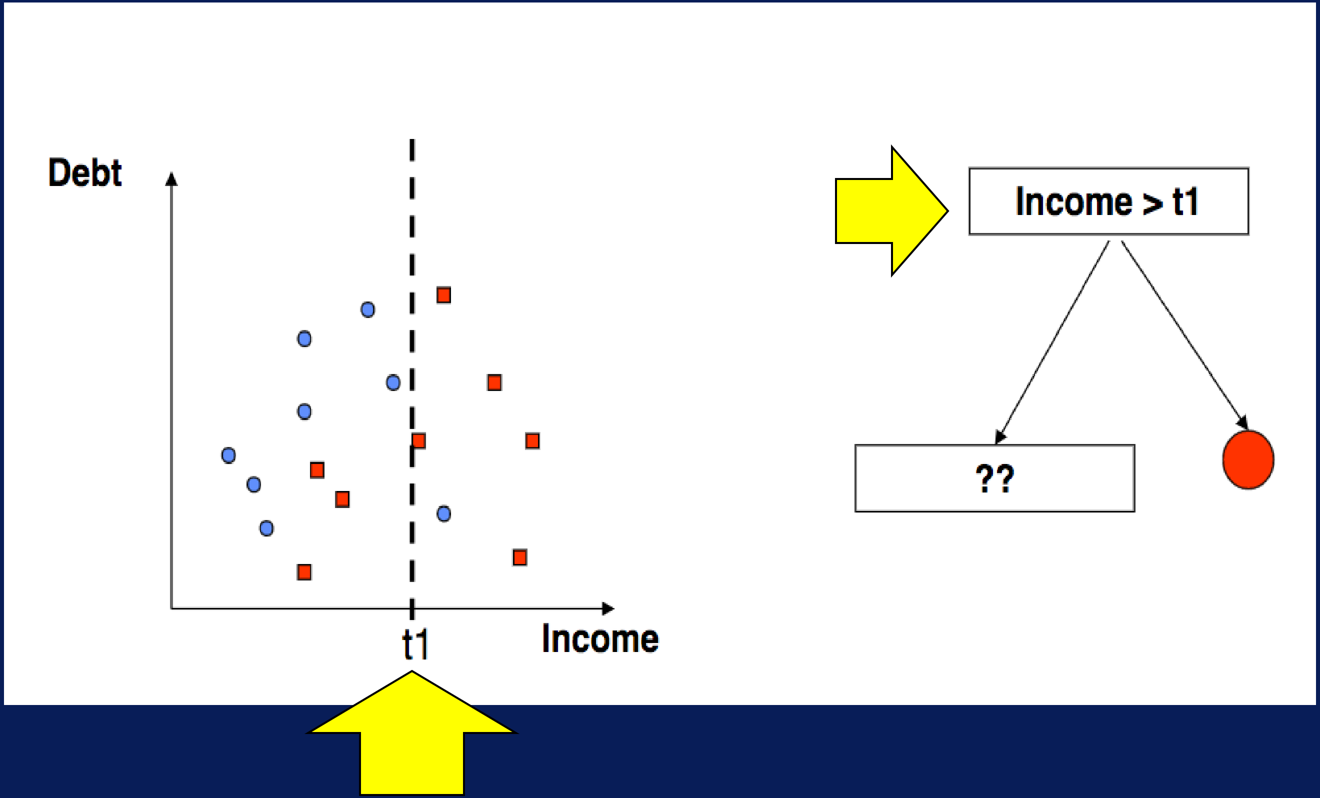


When to Stop Splitting a Node?

- All (or X% of) samples have same class label
- Number of samples in node reaches minimum
- Change in impurity measure is smaller than threshold
- Max tree depth is reached
- Others…

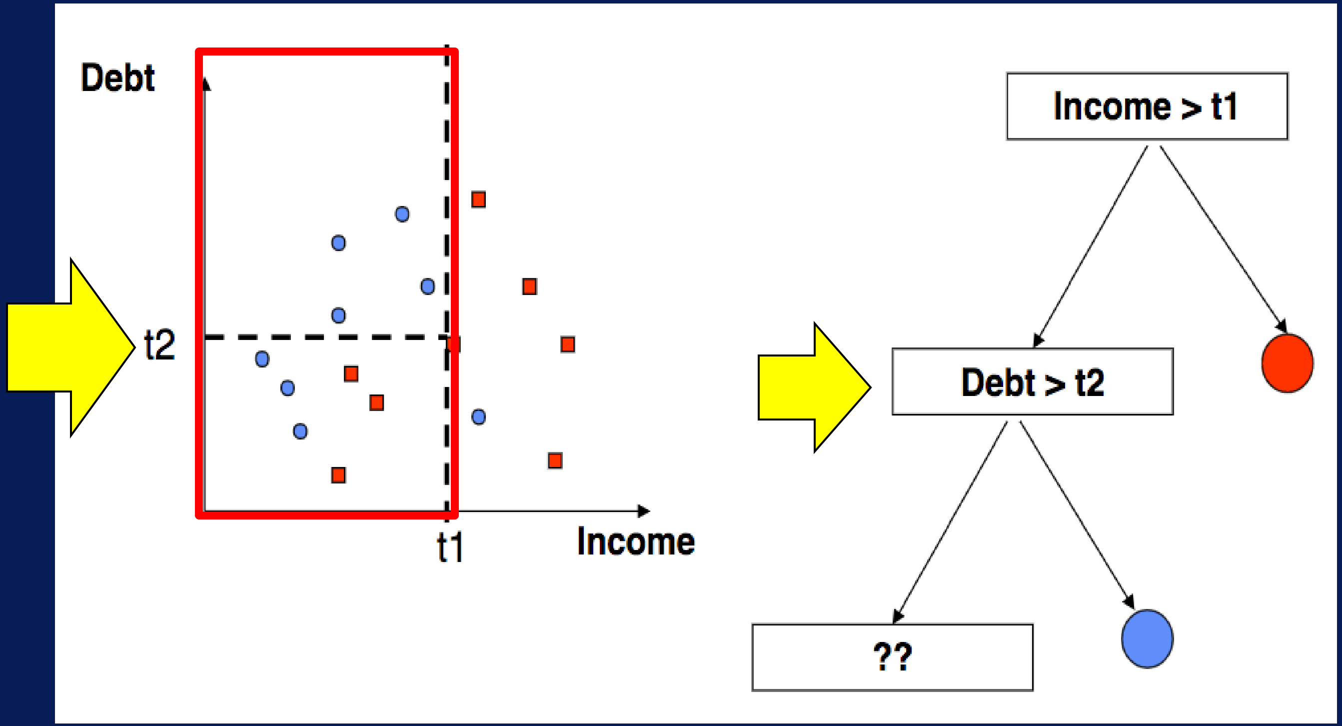
Tree Induction Example

- Split 1



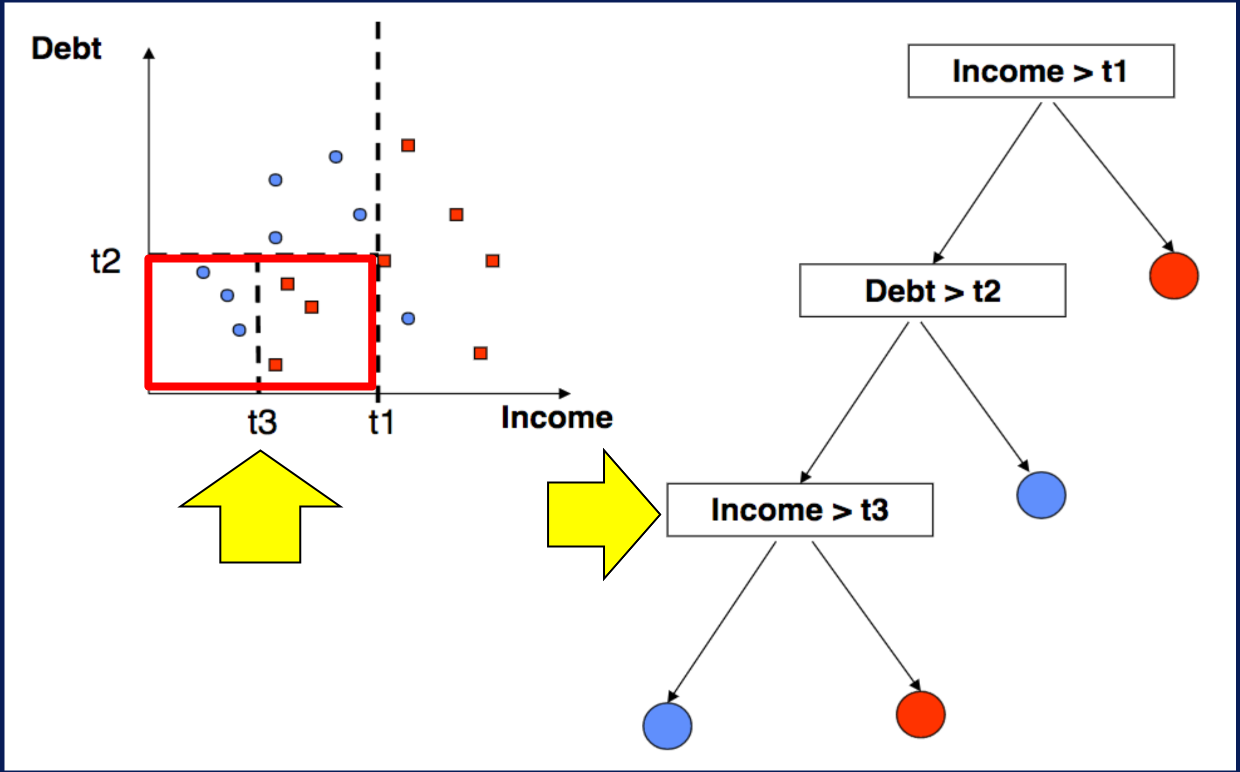
Tree Induction Example

- Split 2



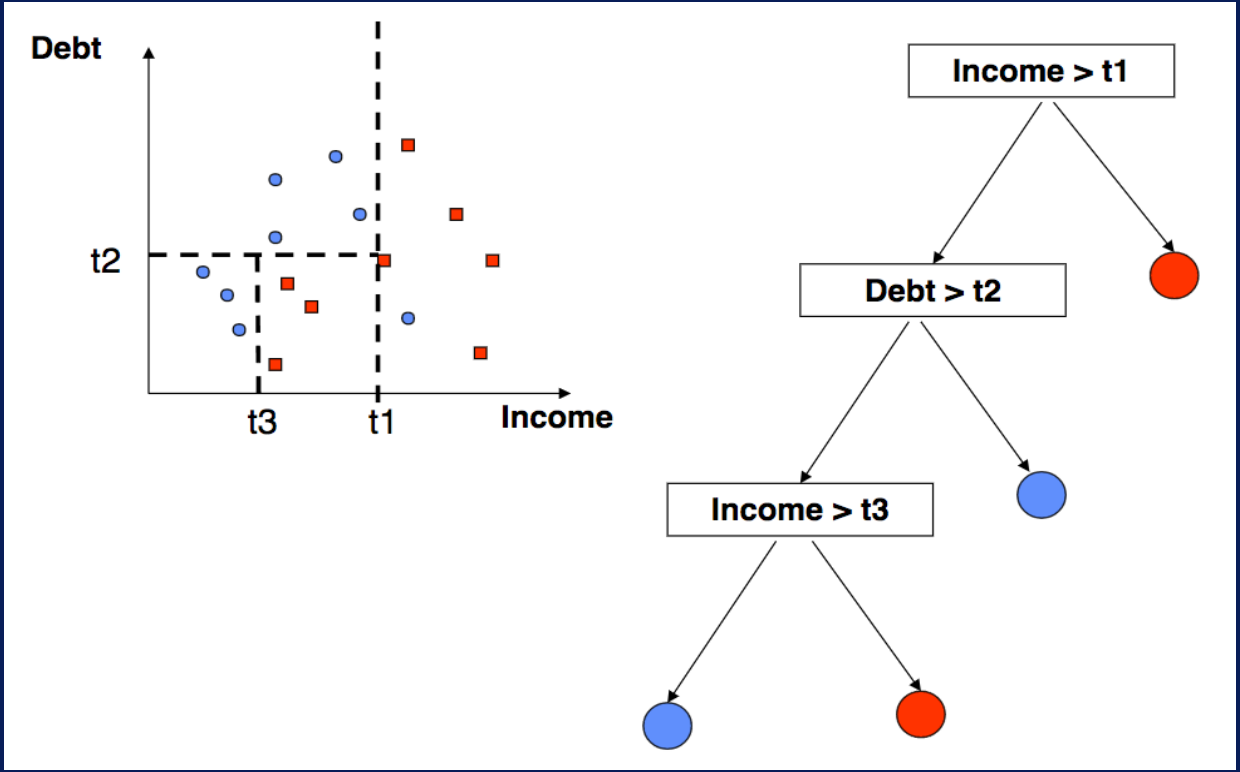
Tree Induction Example

- Split 3



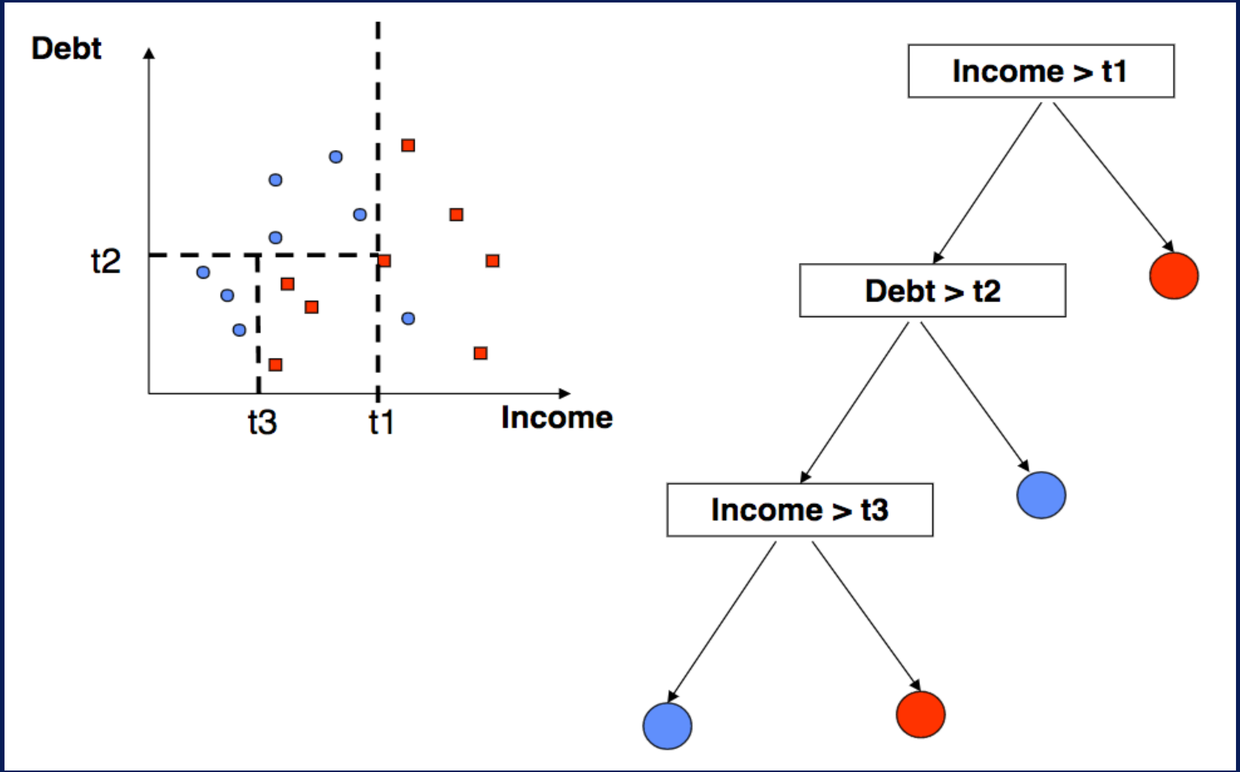
Tree Induction Example

- Resulting model



Decision Boundaries

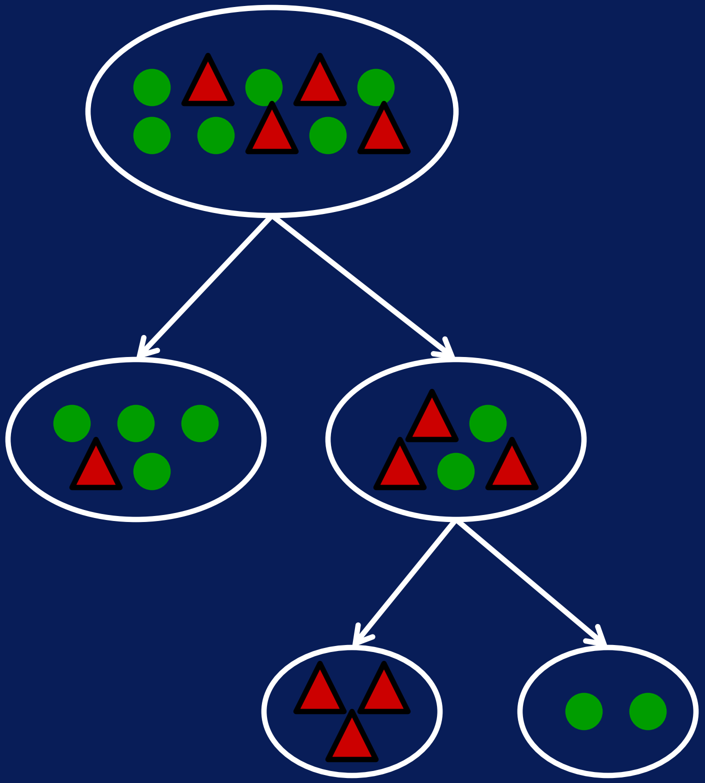
- Rectilinear = Parallel to axes



Decision Tree for Classification

- Resulting tree is often simple and easy to interpret
- Induction is computationally inexpensive
- Greedy approach does not guarantee best solution
- Rectilinear decision boundaries

Decision Tree

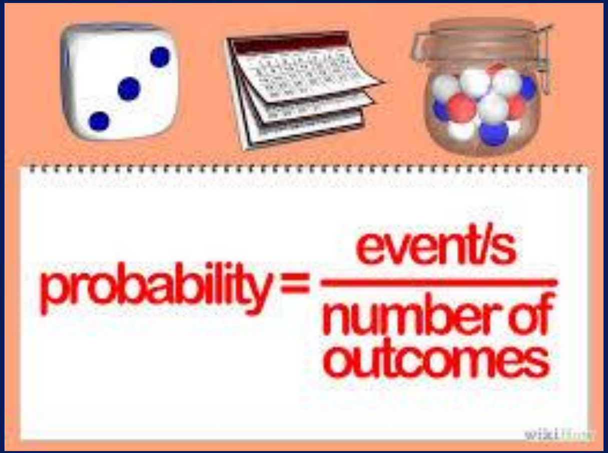


Naïve Bayes

After this video you will be able to..

- Discuss how a Naïve Bayes model works for classification
- Define the components of Bayes' Rule
- Explain what the 'naïve' means in Naïve Bayes

Naïve Bayes Overview



- Probabilistic approach to classification
  - Relationships between input features and class expressed as probabilities
  - Label for sample is class with highest probability given input

Naïve Bayes Classifier

Classification Using Probability

+

Bayes Theorem

+

Feature Independence Assumption

Probability of Event

Probability is measure of how likely an event is

Probability of Event ‘A’ Occurring

$$P(A) = \frac{\#\text{ ways for A}}{\#\text{ possible outcomes}}$$

Probability of Event

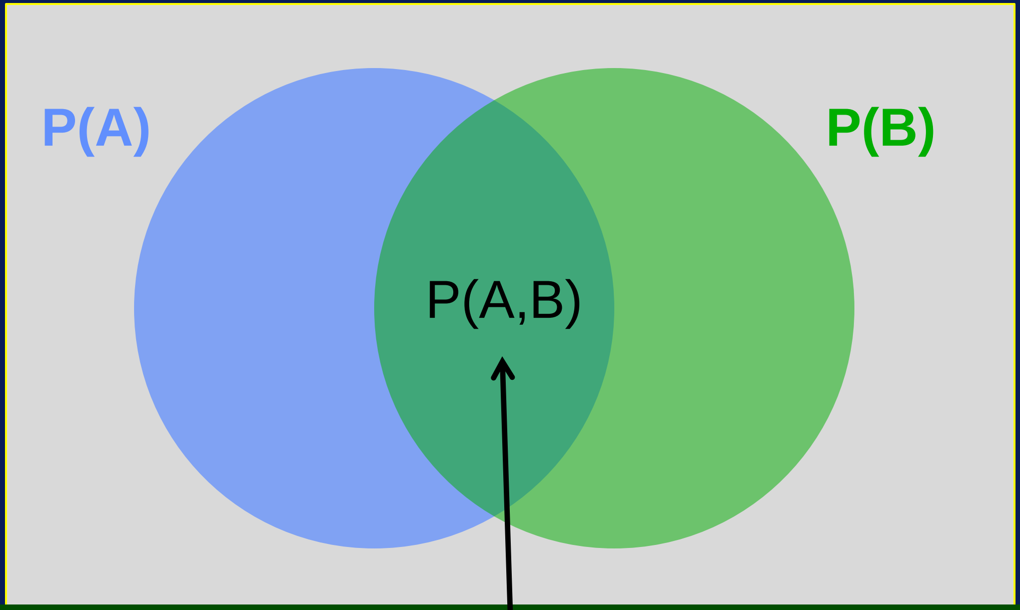
What is the probability of rolling a die and getting 6?

Probability of Rolling 6 on a Die

$$P(6) = \frac{\text{\# ways for getting 6}}{\text{\# possible outcomes}} = \frac{1}{6}$$

Joint Probability

Probability of events A and B occurring together



Joint Probability of A and B

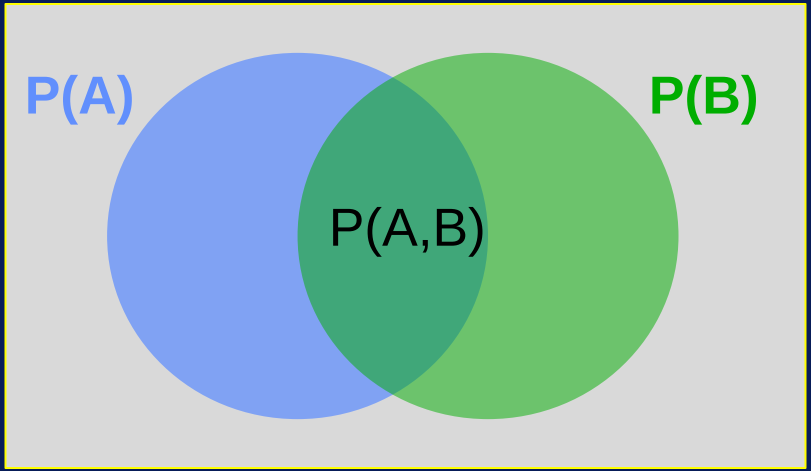
Joint Probability Example

What is the probability of two 6’s when rolling two dice?

Probability of Rolling Two 6’s

$$P(A,B) = P(A) * P(B) = \frac{1}{6} * \frac{1}{6} = \frac{1}{36}$$

Conditional Probability



Probability of event A occurring, given that event B occurred

$$P(A \mid B) = \frac{P(A,B)}{P(B)}$$

Conditional Probability

Bayes' Theorem

- Relationship between P(B | A) and P(A | B) can be expressed through Bayes' Theorem

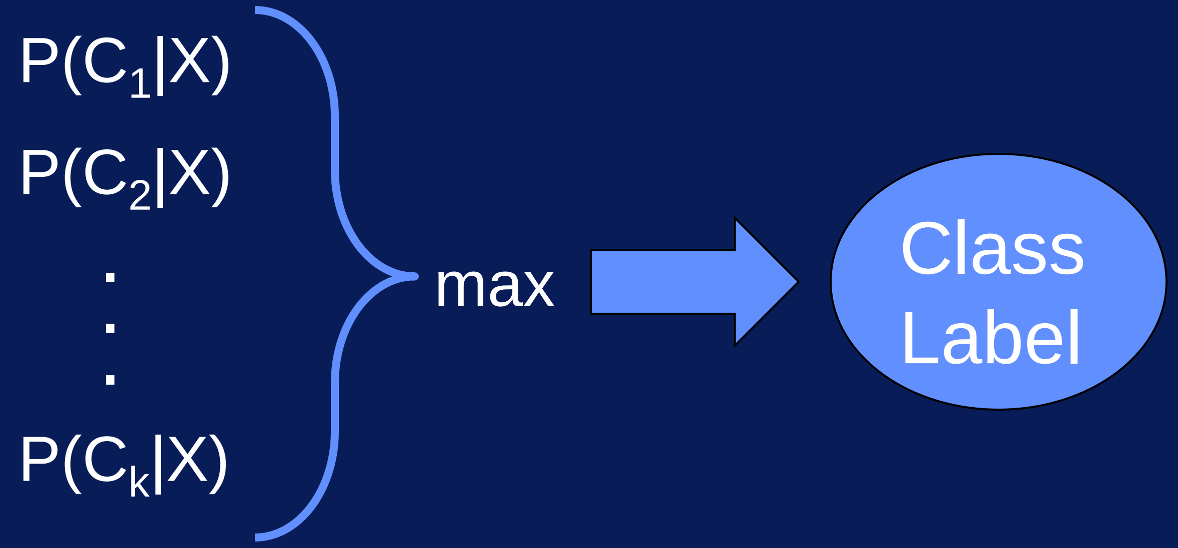
$$P(B \mid A) = \frac{P(A \mid B) * P(B)}{P(A)}$$

Bayes' Theorem

Classification with Probabilities

Given features X={X1,X2,…,Xn}, predict class C

Do this by finding value of C that maximizes P(C | X)



Bayes’ Theorem for Classification

- **But estimating P(C|X) is difficult**
- **Bayes’ Theorem to the rescue!**
  - Simplifies problem

Bayes' Theorem for Classification

Posterior Probability

Class-Conditional Probability

Prior Probability

$$P(C \mid X) = \frac{P(X \mid C) * P(C)}{P(X)}$$

Probability of observing values for input features

Bayes' Theorem for Classification

Need to calculate this

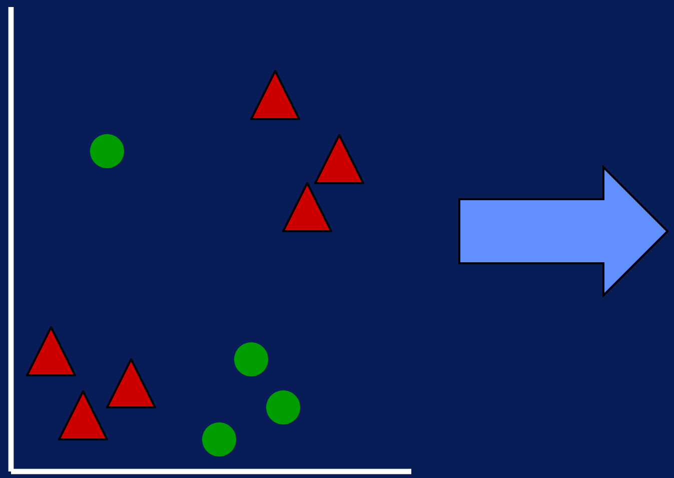
Can be estimated from data!

$$P(C \mid X) = \frac{P(X \mid C) * P(C)}{P(X)}$$

Constant (can be ignored)

To get P(C | X), only need to find P(X | C) and P(C), which can be estimated from the data!

Estimating P(C)



P(●) = 4/10 = 0.4

P(▲) = 6/10 = 0.6

To estimate P(C), calculate fraction of samples for class C in training data.

Estimating P(X | C)

Independence Assumption

- Features are independent of one another:

$$P(X_1,X_2,\ldots,X_n \mid C) = P(X_1|C) * P(X_2 \mid C) * \ldots * P(X_n \mid C)$$

To estimate P(X | C), only need to estimate $P(X_i \mid C)$ individually ➔ Much simpler!

Estimating $P(X_i \mid C)$

Home Owner	Marital Status	Loan Default
Yes	Single	No
No	Married	No
No	Single	No
Yes	Married	No
No	Divorced	Yes
No	Married	No
Yes	Divorced	No
No	Single	Yes
No	Married	No
No	Single	Yes

P(Home Owner = Yes | No) = 3/7 = 0.43

P(Marital Status = Single| Yes) = 2/3 = 0.67

Source: http://www-users.cs.umn.edu/~kumar/dmbook/index.php

Naïve Bayes Classification

- **Fast and simple**
- **Scales well**
- **Independence assumption may not hold true**
  - In practice, still works quite well
- **Does not model interactions between features**

Naïve Bayes Classifier

Classification Using Probability

\+

Bayes Theorem

\+

Feature Independence Assumption

Classification Using Decision Tree in KNIME

Learning Objectives

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

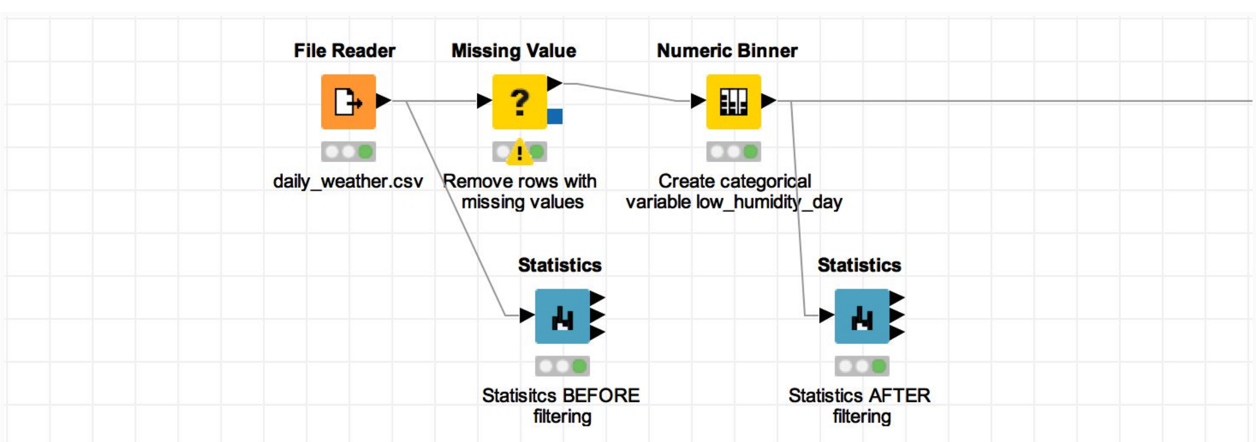
1. Create a categorical variable from a numeric variable
2. Examine the summary statistics of a dataset
3. Build a workflow for a classification task using a decision tree

Problem Description

Now that we have explored the data and looked at how to handle missing values, the next step is to build a classification model to predict days with low humidity. Recall that low humidity is one of the weather conditions that increase the dangers of wildfires, so it would be helpful to be able to predict low-humidity days. We will build a decision model to classify low-humidity days vs. non-low-humidity days based on weather conditions observed at 9am.

Steps

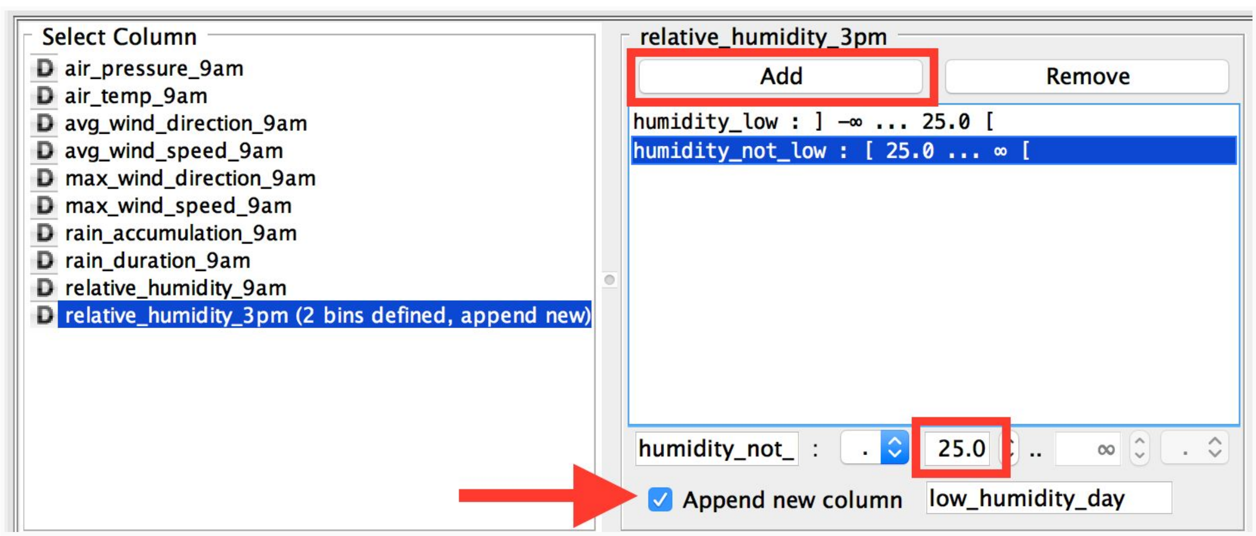
Prepare the data



Let's build a workflow to build a decision tree model to classify low-humidity days vs. days with normal or high humidity. The model will be used to predict low-humidity days

1. Start a new workflow in your local workspace.
2. Use a **File Reader** node to import the daily_weather.csv dataset. Use the configuration dialog to specify the location of the daily_weather.csv file.
3. Connect a **Missing Value** node to the File Reader node. This will handle the missing values that the dataset contains so the data can be analyzed properly. In the configure dialog, in the **Default** tab, choose **Remove Row*** to remove all rows with missing values.
4. As with the Data Exploration Hands-On, use the **Numeric Binner** node to create a new categorical variable with the condition "*if relative_humidity_3pm < 25% then humidity_low is true, else humidity_not_low is true*".

- Locate the **Numeric Binner** node, which is in the Manipulation>Column>Binning category. Drag it to the Workflow Editor, and connect it to the **Missing Value** node.
- Open the Configure Dialog for the Numeric Binner node. Select **relative_humidity_9am**, and **Add** 2 bins. Make one bin called "humidity_low" with the range **-∞ to 25** excluding 25, and another called "humidity_not_low" with the range**25 to** ∞. The endpoint brackets specify that humidity_low excludes 25.0, while humidity_not_low includes 25.0. This is necessary to capture the condition "if relative_humidity_3pm < 25% then low_humidity_day=1, else low_humidity_day=0". Click the checkbox to "**Append new column**" and name it **low_humidity_day**.

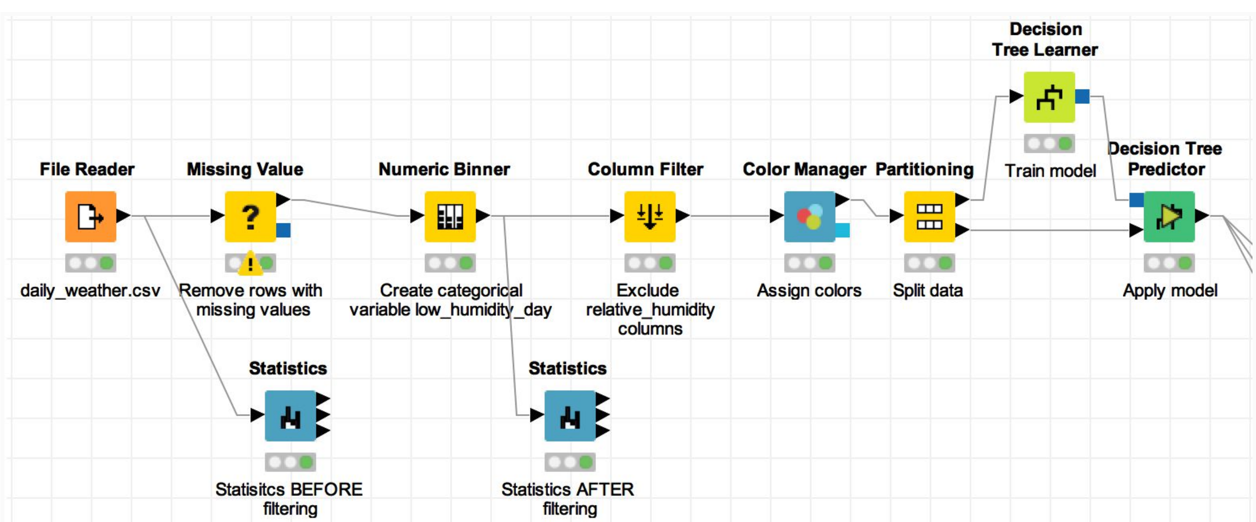


Examine Summary Statistics

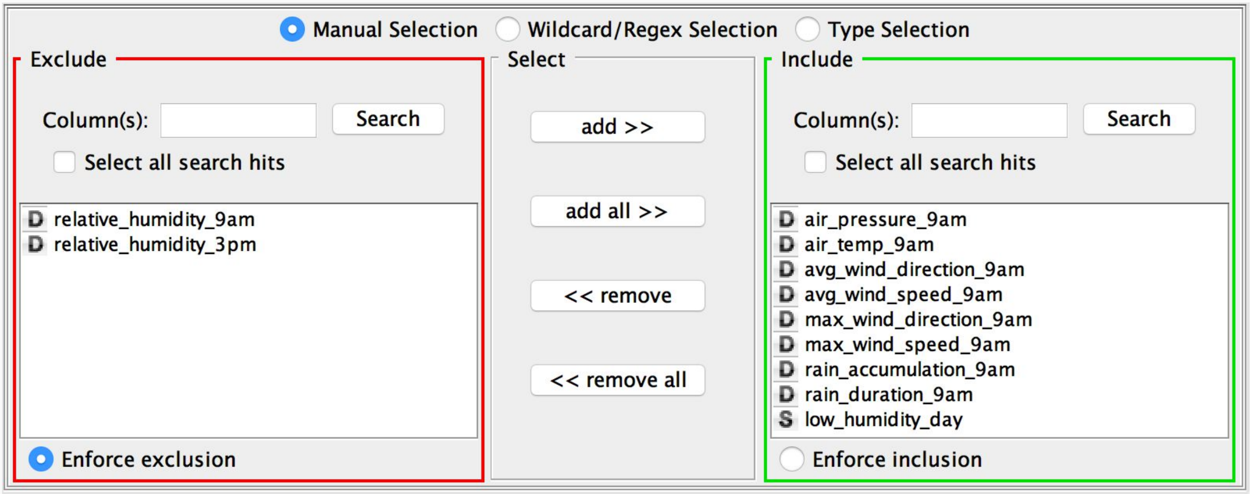
Before we build the workflow any further, let's use some **Statistics** nodes to check a few things about our processed data.

1. Connect a **Statistics** node to the output of the **File Reader** node. In the Configure Dialog of the Statistics node, change **Max no. of possible values per column (in output table)** to **1,500**, and **add all >>** columns to the **Include**side. Rename this node to "Statistics BEFORE filtering".
2. Connect a **Statistics** node to the output of the **Numeric Binner** node. In the Configure Dialog of this Statistics node, change **Max no. of possible values per column (in output table)** to **1,500**, and **add all >>** columns to the **Include**side. Rename this node to "Statistics AFTER filtering".
3. Execute and view both Statistics nodes, and you should see the resulting histograms have the same features in both. This is to ensure that the way we handled the missing values did not skew our data. Now we can check the following:
4. There are missing values for many of the variables in the "Statistics BEFORE filtering" node, but zero missing values in the "Statistics AFTER filtering" node.
5. The distributions of each variable in both "Statistics BEFORE filtering" and "Statistics AFTER filtering" should be about the same. You can spot check a couple of variables by looking at histograms, min, max, mean, and standard deviation.
6. In the "Statistics AFTER filtering" node, look at the **Nominal** tab to see the distribution of low_humidity_day. This shows that the samples are equally distributed between low-humidity days and days with normal or high humidity.

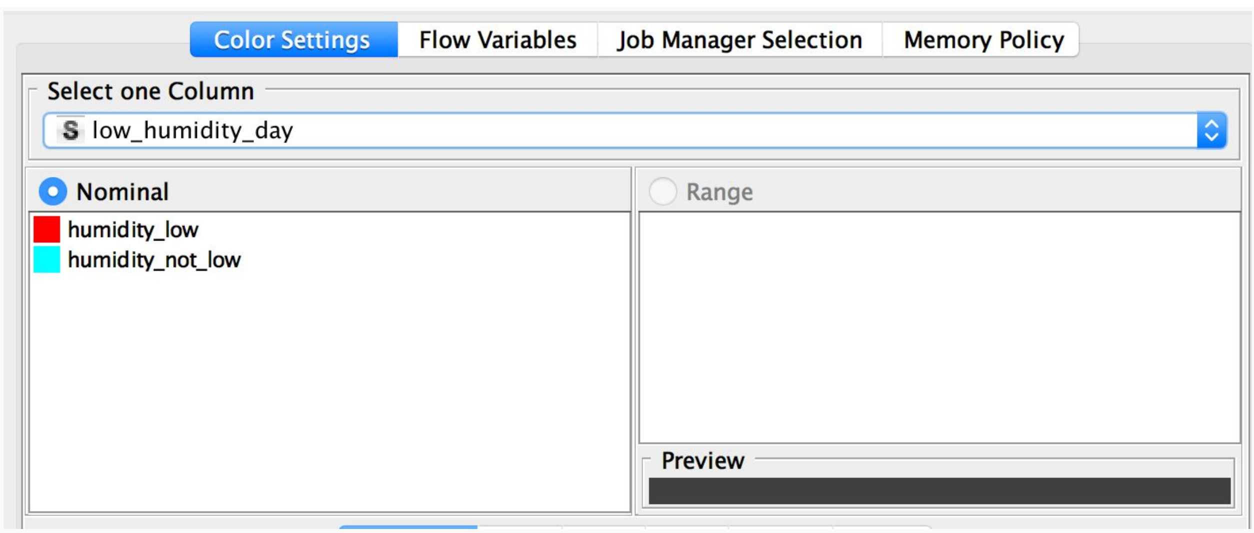
Build a Decision Tree Workflow



1. Connect a **Column Filter** node to the **Numeric Binner** node. In the Configure Dialog of the Column Filter node, exclude only the **relative_humidity_9am and relative_humidity_3pm** columns.

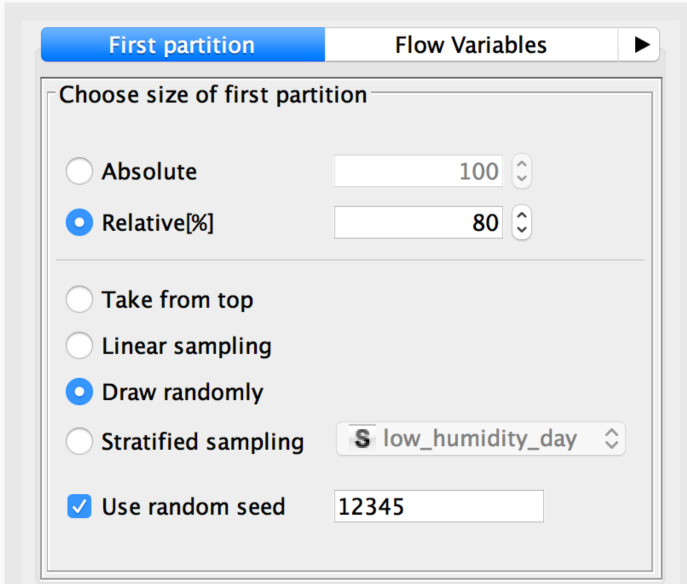


2. Connect a **Color Manager** node to the **Column Filter** node. This will color-code our categorical low_humidity_day variable so it is easier to visualize later on in the workflow. In the Configure Dialog of the Color Manager node, check that for low_humidity_day, the humidity_low is colored red and the humidity_not_low is colored blue.



3. Connect a **Partitioning** node to the **Color Manager** node. The Partitioning node is needed to split the data into training and testing portions. Training data is used to build the decision tree, and test data is used to evaluate the classifier on new data. In the Configure Dialog of the Partitioning node, choose **Relative[%] 80**, **Draw Randomly**, and **Use random seed 12345**. This will randomly select 80% of the data to the 1st output (the training data), and the rest to the 2nd output (the test data).

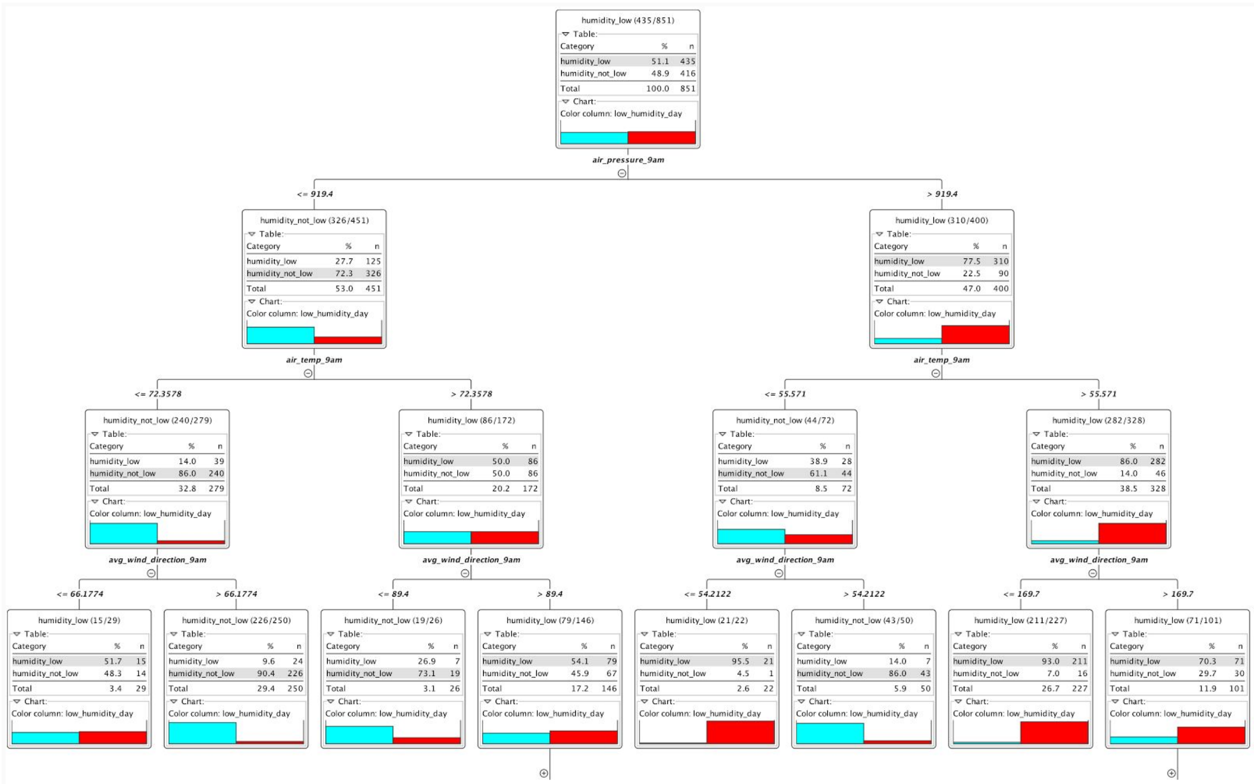
The random seed is set so that everyone can get the same training and test data sets to train and test the decision tree model for this exercise.



4. Connect a **Decision Tree Learner** node to the 1st output of the **Partitioning** node. This node will generate the decision tree classifier using the training data. In the Configure Dialog, change the **Min number of records per node** to **20**. This is a stopping criterion for the tree induction algorithm. It specifies that a node with this number of samples can no longer be split. The default value for this is 2, which is very small, and may result in overfitting.

5. Connect a **Decision Tree Predictor** to the 2nd output of the **Partitioning** node and to the output from the **Decision Tree Learner** node. This node will apply the model to the test data.

6. Execute the workflow. Right-click on the **Decision Tree Predictor** node and select 'View: Decision Tree View' to see the generated decision tree. You can zoom out and click the little '+' buttons to expand the nodes. The next Reading, **Interpreting a Decision Tree in KNIME**, describes how to interpret the resulting decision tree model.



Save Your Workflow

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

Classification in Spark

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

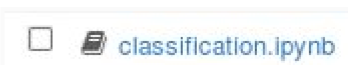
1. Generate a categorical variable from a numeric variable
2. Aggregate the features into one single column
3. Randomly split the data into training and test sets
4. Create a decision tree classifier to predict days with low humidity.

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 handling missing values notebook by clicking on *classification.ipynb:*



Step 2. **Load classes and data.** Execute the first cell in the notebook to load the classes used for this exercise.

```
In [1]: from pyspark.sql import SQLContext
        from pyspark.sql import DataFrameNaFunctions
        from pyspark.ml import Pipeline
        from pyspark.ml.classification import DecisionTreeClassifier
        from pyspark.ml.feature import Binarizer
        from pyspark.ml.feature import VectorAssembler, StringIndexer, VectorIndexer
```

Next, execute the second cell which loads the weather data into a DataFrame and prints the columns.

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

Out[2]: ['number',
         'air_pressure_9am',
         'air_temp_9am',
         'avg_wind_direction_9am',
         'avg_wind_speed_9am',
         'max_wind_direction_9am',
         'max_wind_speed_9am',
         'rain_accumulation_9am',
         'rain_duration_9am',
         'relative_humidity_9am',
         'relative_humidity_3pm']
```

Execute the third cell, which defines the columns in the weather data we will use for the decision tree classifier.

```
In [3]: featureColumns = ['air_pressure_9am','air_temp_9am','avg_wind_direction_9am','avg_wind_speed_9am'
               'max_wind_direction_9am','max_wind_speed_9am','rain_accumulation_9am',
               'rain_duration_9am']
```

Step 3. **Drop unused and missing data.** We do not need the *number* column in our data, so let's remove it from the DataFrame:

```
In [4]: df = df.drop('number')
```

Next, let's remove all rows with missing data:

```
In [5]: df = df.na.drop()
```

We can print the number of rows and columns in our DataFrame:

```
In [6]: df.count(), len(df.columns)

Out[6]: (1064, 10)
```

Step 4. **Create categorical variable.** Let's create a categorical variable to denote if the humidity is not low. If the value is less than 25%, then we want the categorical value to be 0, otherwise the categorical value should be 1. We can create this categorical variable as a column in a DataFrame using *Binarizer:*

```
In [7]: binarizer = Binarizer(threshold=24.99999, inputCol="relative_humidity_3pm", outputCol="label")
        binarizedDF = binarizer.transform(df)
```

The *threshold* argument specifies the threshold value for the variable, *inputCol* is the input column to read, and *outputCol* is the name of the new categorical column. The second line applies the *Binarizer* and creates a new DataFrame with the categorical column. We can look at the first four values in the new DataFrame:

```
In [8]: binarizedDF.select("relative_humidity_3pm","label").show(4)

        +--------------------+-----+
        |relative_humidity_3pm|label|
        +--------------------+-----+
        |   36.160000000000494|  1.0|
        |     19.4265967985621|  0.0|
        |   14.460000000000045|  0.0|
        |   12.742547353761848|  0.0|
        +--------------------+-----+
        only showing top 4 rows
```

The first row's humidity value is greater than 25% and the label is 1. The other humidity values are less than 25% and have labels equal to 0.

Step 5. **Aggregate features.** Let's aggregate the features we will use to make predictions into a single column:

```
In [9]: assembler = VectorAssembler(inputCols=featureColumns, outputCol="features")
        assembled = assembler.transform(binarizedDF)
```

The *inputCols* argument specifies our list of column names we defined earlier, and *outputCol* is the name of the new column. The second line creates a new DataFrame with the aggregated features in a column.

Step 6. **Split training and test data.** We can split the data by calling *randomSplit():*

```
In [10]: (trainingData, testData) = assembled.randomSplit([0.8,0.2], seed = 13234 )
```

The first argument is how many parts to split the data into and the *approximate* size of each. This specifies two sets of 80% and 20%. Normally, the seed should not be specified, but we use a specific value here so that everyone will get the same decision tree.

We can print the number of rows in each DataFrame to check the sizes (1095 * 80% = 851.2):

```
In [11]: trainingData.count(), testData.count()
Out[11]: (854, 210)
```

Step 7. **Create and train decision tree.** Let's create the decision tree:

```
In [12]: dt = DecisionTreeClassifier(labelCol="label", featuresCol="features", maxDepth=5,
                                     minInstancesPerNode=20, impurity="gini")
```

The *labelCol* argument is the column we are trying to predict, *featuresCol* specifies the aggregated features column, *maxDepth* is stopping criterion for tree induction based on maximum depth of tree, *minInstancesPerNode*is stopping criterion for tree induction based on minimum number of samples in a node, and *impurity* is the impurity measure used to split nodes.

We can create a model by training the decision tree. This is done by executing it in a *Pipeline:*

```
In [13]: pipeline = Pipeline(stages=[dt])
         model = pipeline.fit(trainingData)
```

Let's make predictions using our test data set:

```
In [14]: predictions = model.transform(testData)
```

Looking at the first ten rows in the prediction, we can see the prediction matches the input:

```
In [15]: predictions.select("prediction", "label").show(10)

+----------+-----+
|prediction|label|
+----------+-----+
|       1.0|  1.0|
|       1.0|  1.0|
|       1.0|  1.0|
|       1.0|  1.0|
|       1.0|  1.0|
|       1.0|  1.0|
|       0.0|  0.0|
|       1.0|  1.0|
|       1.0|  1.0|
|       1.0|  1.0|
+----------+-----+
only showing top 10 rows
```

Step 8. **Save predictions to CSV.** Finally, let's save the predictions to a CSV file. In the next Spark hands-on activity, we will evaluate the accuracy.

Let's save only the *prediction* and *label* columns to a CSV file:

```
In [16]: predictions.select("prediction", "label").write.save(path="file:///home/cloudera/Downloads/big-data-4/predictions.csv",
                                                               format="com.databricks.spark.csv",
                                                               header='true')
```

Interpreting a Decision Tree in KNIME

This document describes how to interpret a decision tree classifier. We will use the tree created in the Classification Using Decision Tree in KNIME Hands-On.

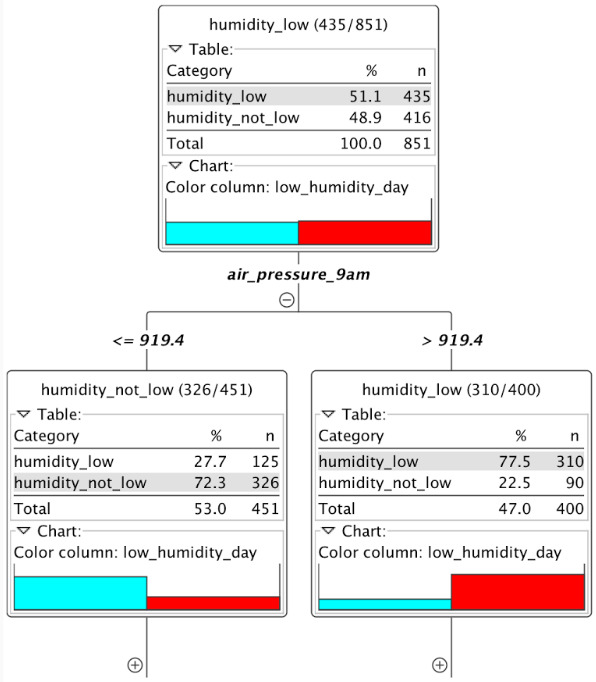
Classification Task

Recall that the task is to classify low-humidity days vs. days with normal or high relative humidity. The class label is based on the categorical variable **low_humidity_day**. This variable was created from the numeric variable relative_humidity_3pm. The class target low_humidity_day was created with the following categories:

- **humidity_low**: if relative_humidity_3pm < 25
- **humidity_not_low**: if relative_humidity_3pm >= 25

Decision Tree Model

First, let's take a look at just the first two levels of the tree. You can see the following image by right-clicking on the **Decision Tree Learner** node in the workflow and selecting "View: Decision Tree View":

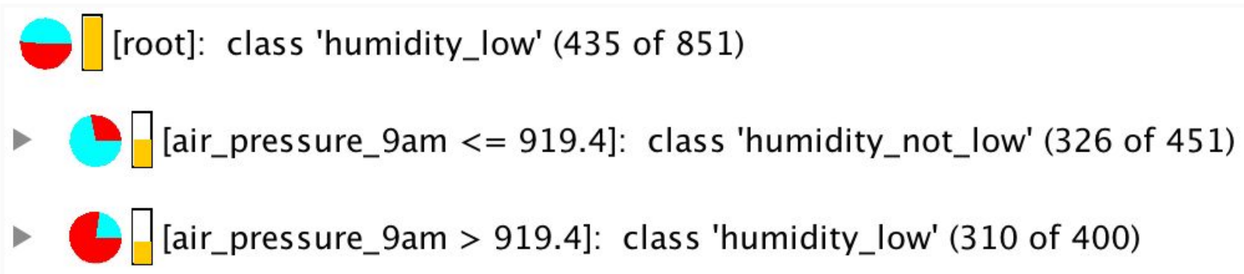


Looking at the root node (the top node), we see that there 851 samples in total. Of these, 435 or 51.1% of the samples are labeled as humidity_low; that is, the true label of these samples is humidity_low. Of the total number of samples, 416 or 48.9% are labeled as humidity_not_low. So at the root node, approximately half of the samples are humidity_low and half are humidity_not_low. This is indicated by the color bars at the bottom of the root node: blue is for humidity_not_low, and red is for humidity_low, and the height of each bar specifies the percentage of samples labeled with the respective category.

Split #1 on air_pressure_9am

The first split is on the variable **air_pressure_9am**. Samples with air_pressure_9am <= 919.4 are placed in the left child node where most of the samples are labeled as humidity_not_low. Samples with air_pressure > 919.4 are placed in the right child node where most of the samples are labeled as humidity_low. Note that the color red is associated with the humidity_low class. What this first split specifies is that high values of air_pressure are associated with humidity_low. This makes sense since high air pressure usually corresponds to sunny days, which have normal or high relative humidity.

To look at more levels in the decision tree, we need a more compact view. So we will now switch to the ‘simple’ view. The following image shows the same tree structure as the image of the decision tree above, and is generated by clicking on the Decision Tree Learner node and selecting "View: Decision Tree View (simple)":

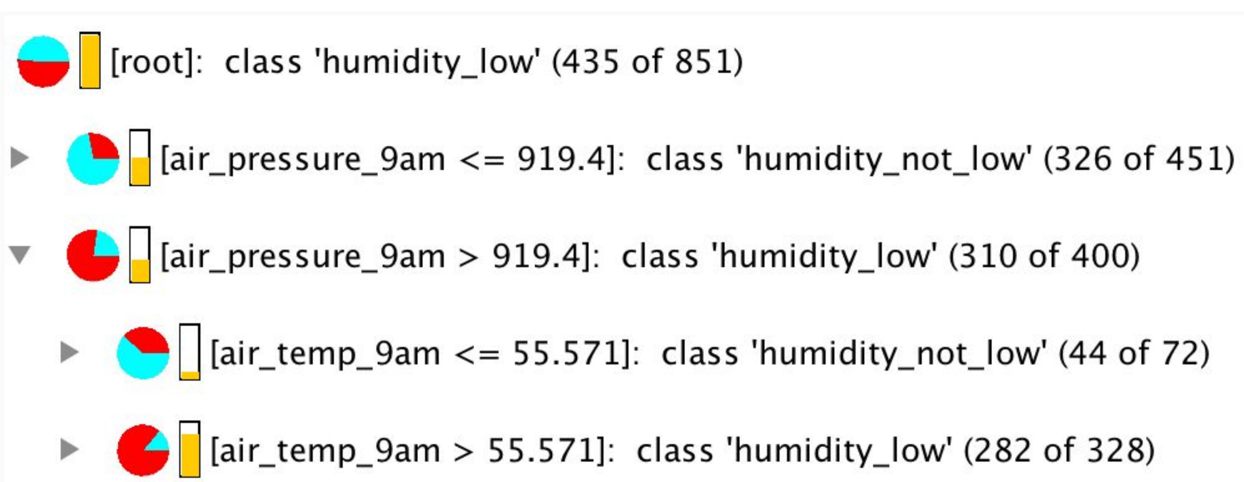


The root node is shown as the top line, followed by the children nodes resulting from the split on air_pressure_9am. Again, samples with air_pressure_9am <= 919.4 are placed in the left child node (shown right under the root node) where most of the samples are labeled as humidity_not_low. In other words, the true label for most samples in the left child node is humidity_not_low, which is indicated by the pie chart symbol being mostly blue, and the numbers in parentheses specifying that 326 out of 451 samples in that node are labeled humidity_not_low. Samples with air_pressure_9am > 919.4 are placed in the right child node where most of the samples are labeled as humidity_low.

Note that no prediction has been made yet since classification decisions are not made until a leaf node is reached.

Split #2 on air_temp_9am

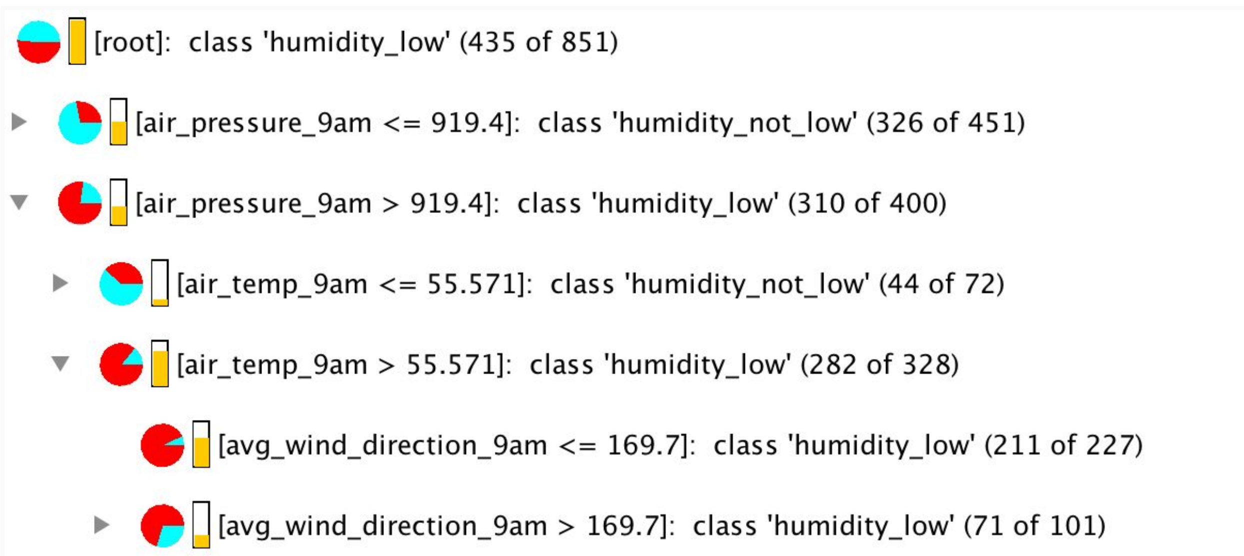
Let's continue down the branch of the right child, where most of the samples have true label as humidity_low. If we expand that node, we get the following tree:



We see that this split is based on the variable **air_temp_9am**. If a sample has a value for air_temp_9am <= 55.571, then it placed in the left child node, where most of the samples are labeled as humidity_not_low. And if a sample has a value for air_temp_9am > 55.571, then it is placed in the right child node, where most of the samples are labeled as humidity_low. What this means is that low-humidity days are associated with warmer days. This makes sense since days with high humidity tend to be rainy days with cooler temperatures, while days with low humidity are sunny days with warmer temperatures.

Split #3 on avg_wind_direction_9am

Continuing with the humidity_low branch, we expand the right child node to get the following:



The third split is based on the variable **avg_wind_direction_9am**. Samples with avg_wind_direction_9am <= 169.7 are placed in the left child node where most of the samples are labeled as humidity_low. Samples with avg_wind_diection_9am > 169.7 are placed in the right child node. Notice that most of the samples in the right child node are also labeled humidity_low, but there is still additional processing needed with those samples since the right child node is not a leaf node.

Classification Rules

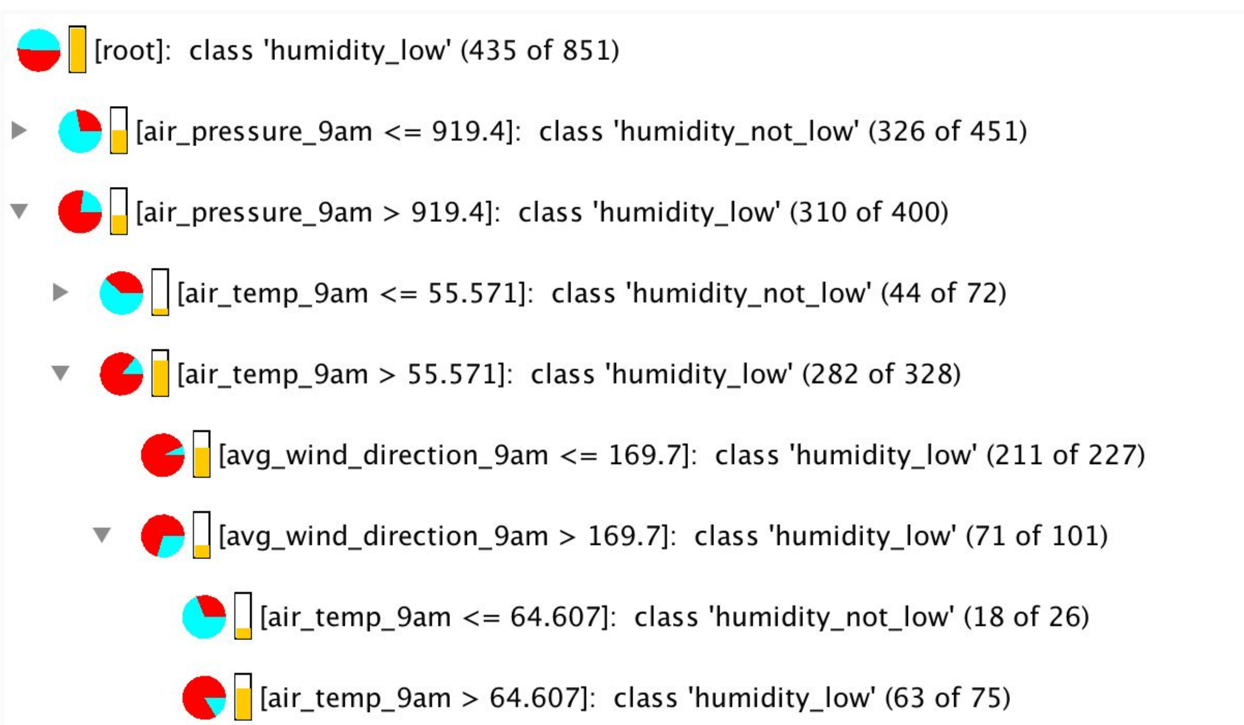
With the left child node, we have now reached a leaf node! Traversing from the root node to this leaf node, we can now see how a sample is classified as humidity_low:

1. If air_pressure_9am > 919.4 and
2. If air_temp_9am > 55.571 and
3. If avg_wind_direction_9am <= 169.7
4. Then sample is classified as humidity_low

This translates to days with high air pressure and warmer temperatures, with wind direction from the east are likely to be days with low relative humidity.

We have discussed above that low humidity is more likely to occur on sunny days with high air pressure and warmer temperatures. Now let's consider wind direction. Values for wind direction start at 0 degree for due North, and increases clockwise. So wind direction <= 169.7 means that the wind is from an eastern direction. For San Diego, this means warmer, drier air from the inland areas as opposed to cooler air with more moisture from the ocean. So this relationship between winds from the east and days with low humidity makes sense.

Expanding the right child node with avg_wind_direction_9am > 169.7, we get:



For the left leaf node, we see the following rules:

1. If air_pressure_9am > 919.4 and
2. If air_temp_9am > 55.571 and
3. If avg_wind_direction_9am > 169.7 and

4. If air_temp_9am <= 64.607
5. Then sample is classified as humidity_not_low

This translates to the following: Days with high air pressure, winds from the west, and temperatures between 56 and 65 degrees Fahrenheit are likely to be days with normal or high relative humidity.

For the right leaf node, we get:

1. If air_pressure_9am > 919.4 and
2. If air_temp_9am > 55.571 and
3. If avg_wind_direction_9am > 169.7 and
4. If air_temp_9am > 64.607
5. Then sample is classified as humidity_low

This translates to: Days with high air pressure, winds from the west, and temperatures greater than 65 degrees Fahrenheit are likely to be days with low humidity.

This branch is now complete. There are three leaf nodes, so there are three ways to assign a prediction of either humidity_low or humidity_not_low to each sample that is sent down this branch of the tree.

Other branches of the tree can be interpreted in a similar way. As with any other real dataset, some cases may require complex rules to form a classification decision.

Feature Importance

Aside from interpretability, another advantage of decision tree is that the resulting model tells you which features are important in the classification task. If you expand the tree to show all leaf nodes, you will see all the variables that the tree uses to perform the classification task.

For our daily weather dataset, note that out of the seven original input variables, only four variables (air_pressure_9am, air_temp_9am, avg_wind_direction_9am, max_wind_direction_9am) are used in the construction of the tree. These four variables are deemed important variables for this classification task, while the other variables do not contribute to the classification decisions made by the model.