bioinformatics.ca

Canadian Bioinformatics Workshops

www.bioinformatics.ca

bioinformaticsdotca.github.io

Creative Commons

This page is available in the following languages:
Afrikaans български Català Dansk Deutsch Ελληνικά English English (CA) English (GB) English (US) Esperanto Castellano Castellano (AR) Español (CL) Castellano (CO) Español (Ecuador) Castellano (MX) Castellano (PE) Euskara Suomeksi français français (CA) Galego עברית hrvatski Magyar Italiano 日本語 한국어 Macedonian Melayu Nederlands Norsk Sesotho sa Leboa polski Português română slovenski jezik српски srpski (latinica) Sotho svenska 中文 華語 (台灣) isiZulu

creative commons

Attribution-Share Alike 2.5 Canada

You are free:

to Share — to copy, distribute and transmit the work

to Remix — to adapt the work

Free Cultural APPROVED FOR Works

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar licence to this one.

- For any reuse or distribution, you must make clear to others the licence terms of this work.
- Any of the above conditions can be waived if you get permission from the copyright holder.
- The author's moral rights are retained in this licence.

Disclaimer

Your fair dealing and other rights are in no way affected by the above.
This is a human-readable summary of the Legal Code (the full licence) available in the following languages:
English French

Learn how to distribute your work using this licence

Schedule

Day 1		Day 2	
Time (EDT)	Wednesday Aug. 16	Time (EDT)	Thursday Aug. 17
10:00	Introductions and Technology Check	10:00	Module 5: Gene Finding with NNs (Lecture and Lab)
10:45	Module 1: Introduction to Machine Learning (Lecture)	11:30	Break (30 min)
12:15	Break (30 min)	12:00	Module 6: Machine Learning with Keras and Scikit-Learn (Lecture/Lab)
12:45	Module 2: Decision Trees (Lecture and Lab)	14:00	Break (1 hour)
14:15	Break (45 min)	15:00	Module 7: Machine Learning with Keras and Scikit-Learn (Cont'd)
15:00	Module 3: Neural Networks (Lecture and Lab)	16:00	Break (30 min)
16:30	Break (30 min)	16:30	Module 8: Information Extraction with ChatGPT (Lecture and Lab)
17:00	Module 4: Neural Networks for 2º Structure (Lecture and Homework)	17:45	Survey and Closing Remarks
18:00	End of Day 1	18:00	End of Day 2

Module(Lecture/Lab) 2: Introduction to Decision Trees

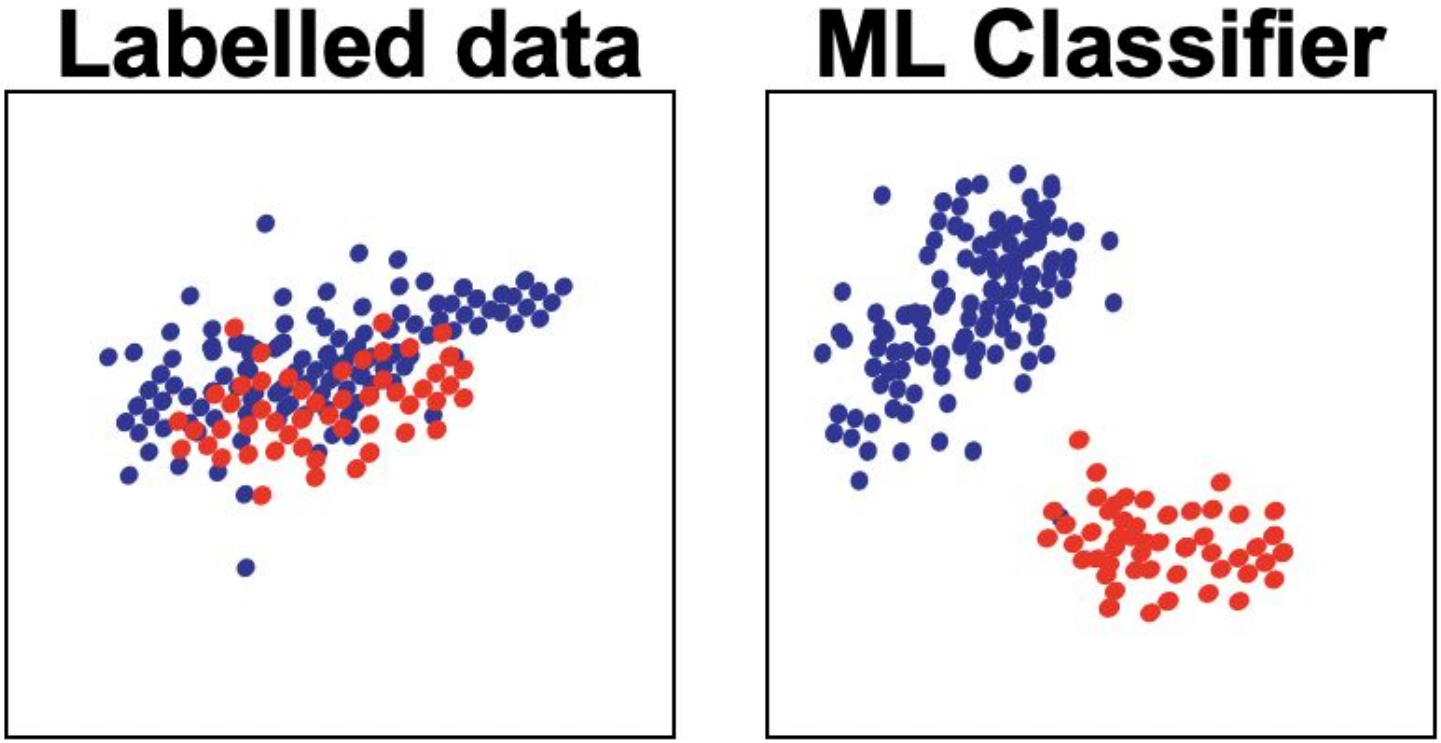
David Wishart

Machine Learning for Bioinformatics

Aug. 16-17, 2023

bioinformatics.ca

Classification & ML



Learning Objectives

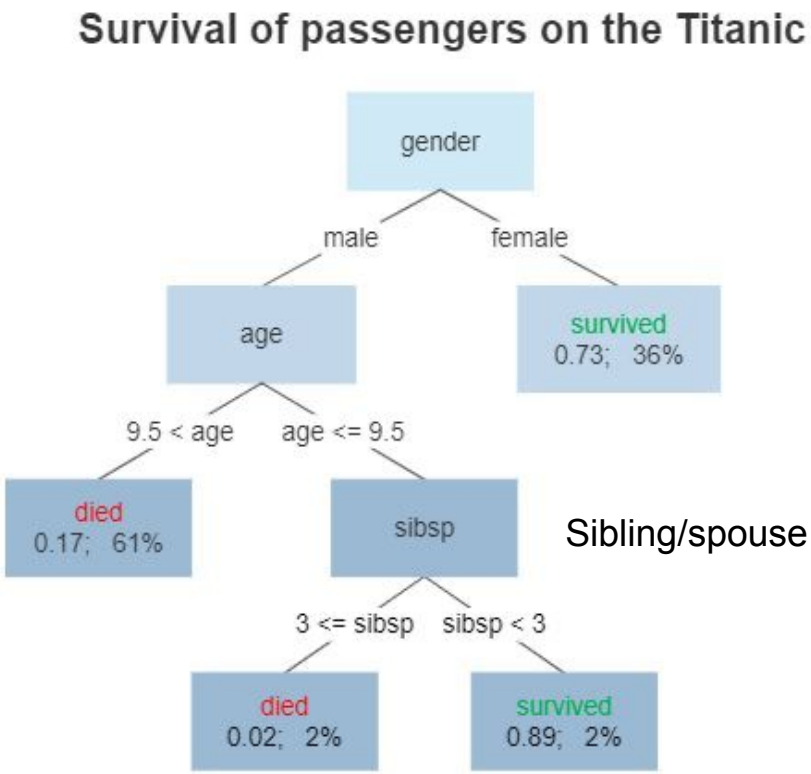
- To review classification & clustering
- To introduce Decision Trees (DTs) in ML
- To introduce the concepts of information gain, Shannon entropy and Gini index
- To review feature selection
- To introduce the Iris classification problem
- To review Python code for a simple Iris DT
- To explore other Iris DT models with different data types
- Lab – experiment on your own with Colab

Clustering vs. Classification

- **Clustering** – A process by which objects that are logically similar in characteristics are grouped together. In clustering, the object classes are yet to be defined or labeled
- **Classification** – A process by which labeled objects (with the label being based on the objects' properties or characteristics) are categorized based on their properties
- **Clustering is different than Classification**
- Supervised ML is focused on using algorithms that learn how to assign class labels

Decision Trees

- Simplest ML algorithms to understand and implement
- Supervised learning method for classification or regression
- Computer learns to split, categorize or regress data based on decisions (greater than, less than, yes/no) and "cost" of decisions
- Tree-like structure consists of branches (edges) and leaves (nodes)



Decision Trees *cont...*

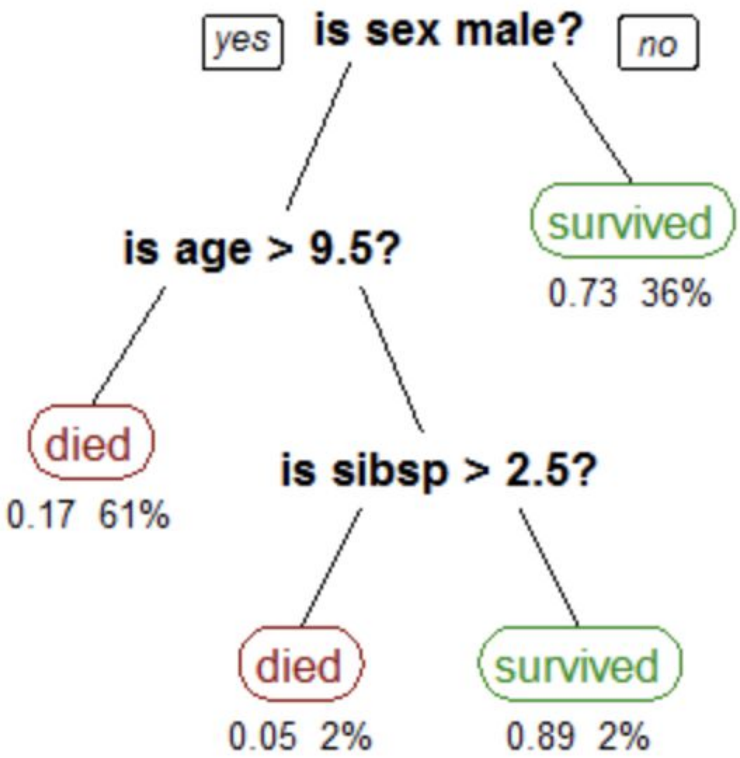
- **Definition - A decision tree is a flowchart-like structure in which each internal node represents a test on an attribute (e.g. whether a coin flip comes up heads or tails) and each branch represents the outcome of the test and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.**

Decision Trees *cont...*

- Two types of decision trees
- Classification tree – classifies objects or data sets, predicts classification or categorical values
- Regression tree – predicts continuous values instead of categorical or nominal values
- DTs are called CARTs (Classification And Regression Trees)

Decision Trees *cont...*

- Growing a decision tree involves deciding on which features to choose and what conditions to use for splitting, along with knowing when to stop
- Black – decision or condition rule
- Red/green – leaf node or final result
- Edges – path for next decision or final result



DT Terminology

- **Root Node** – Represents the entire population or sample set, this is what is divided into two or more homogenous sets
- **Splitting** – Process of dividing a node into 2 or more sub-nodes
- **Decision Node** – when a sub-node splits into further sub-nodes it is called a decision node
- **Leaf/Terminal Node** – Nodes that do not split
- **Parent/Child Node** – A node which is divided into sub-nodes is a parent node, the sub-nodes are child nodes

DT Advantages vs. Disadvantages

Advantages

- Easy to understand and interpret
- White-box (not a black box) model
- Handles both numerical and categorical data
- Requires little to no data preparation
- Mirrors human decision making and methods
- Built-in feature selection
- No need for data normalization or other statistical fixes

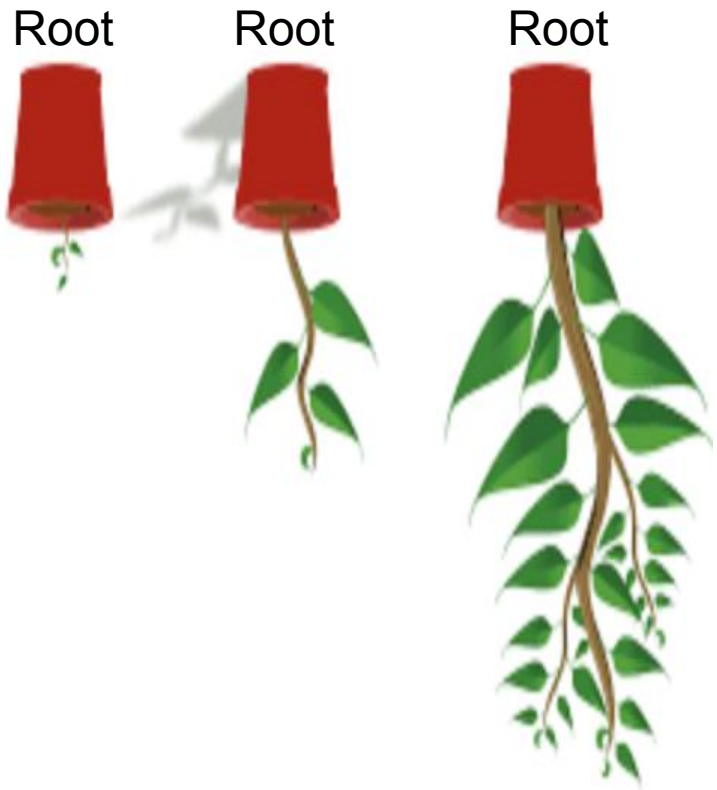
Disadvantages

- Not very robust
- Susceptible to small changes in training (bagging and boosting fix this)
- Heuristic method
- Prone to overfitting
- Greedy algorithm not guaranteed to give best solution (RFs fix this problem)
- Inherent bias to data with more categorical levels

How To ”Learn” a Decision Tree?

Recursive Binary Splitting or Iterative Dichotomization

- In RBS or ID3, all features are considered, and different split points are tried and tested using a cost function
- The split with the lowest cost or highest information gain (IG) is selected
- If we have 3 features, then there are 3 candidate splits
- Calculate the cost (IG) of each split, the split that costs least (or has highest IG) is chosen as the root
- Repeat cost (or IG) calculation with remaining classes



Cost Measures (Information Gain)

- **Based on the concept of entropy, which is the degree of uncertainty or disorder where p_i = probability of being in class i, E(S) = entropy or Shannon entropy**
- **Information Gain (IG) is used to determine which feature/attribute gives the maximum amount of information about a class**

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Shannon Entropy is a log function which can be costly to calculate

$$IG_{feature} = E(S)_{dataset} - E(S)_{feature}$$

The split with the maximum IG is the one that is the tree root

Notes on Shannon Entropy

- All p values are fractional (<1)
- Having a negative sign ensures E(S) is positive
- For 2 classes the max entropy is 1

$$S = -\sum_{i=1}^{2} \frac{1}{2} \log_2 \frac{1}{2} = -\sum_{i=1}^{2} \frac{1}{2} \cdot (-1) = 1$$

- For 4 classes the max entropy is 2
- For 8 classes the max entropy is 3
- For 16 classes the max entropy is 4

Entropy/IG Example

Features

Label/Class

Observations

Age	Mileage	Road Tested	Buy
Recent	Low	Yes	Buy
Recent	High	Yes	Buy
Old	Low	No	Don't buy
Recent	High	No	Don't buy

Entropy/IG Example *cont...*

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Age	Mileage	Road Tested	Buy
Recent	Low	Yes	Buy ✓
Recent	High	Yes	Buy ✓
Old	Low	No	Don't buy ✗
Recent	High	No	Don't buy ✗

Calculating Entropy for the root node

$$E = -\left(P(✓) * \log_2(P(✓)) + P(✗) * \log_2(P(✗))\right)$$

Probability formula:

$$P(✓) = \frac{\text{count of } ✓}{\text{total examples}}$$

$P(✓) = 2/4 = 0.5$

$P(✗) = 2/4 = 0.5$

Plugging these values in the formula we get:

$$E = -\left(0.5 * \log_2(0.5) + 0.5 * \log_2(0.5)\right)$$

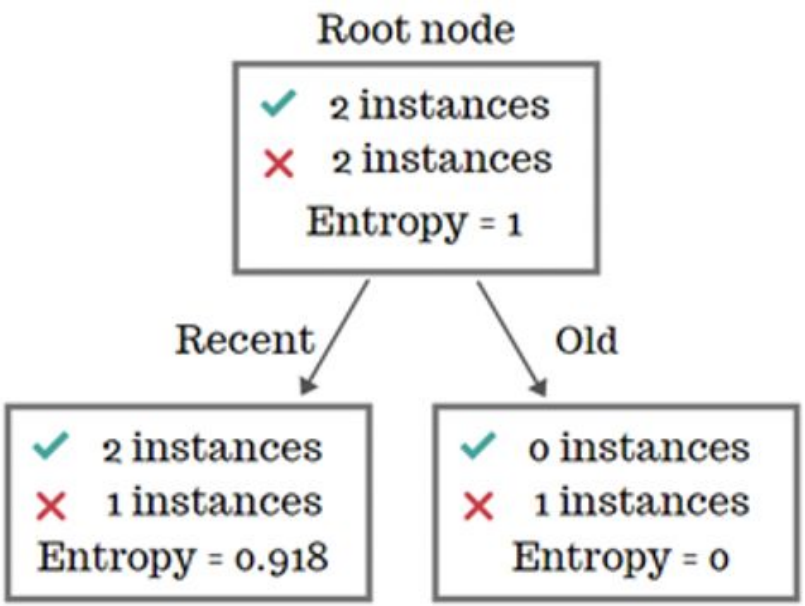
$$E = 1$$

Recall: $\log_2(0.5) = -1$
Recall: $\log_2(1.0) = 0$

Entropy/IG Example *cont...*

$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$$

Age	Mileage	Road Tested	Buy
Recent	Low	Yes	Buy ✓
Recent	High	Yes	Buy ✓
Old	Low	No	Don't buy ✗
Recent	High	No	Don't buy ✗



$-E_{Rec} = 0.666*\log_2(0.666) + 0.333*\log_2(0.333) = 0.666*0.585+0.333*1.585 = 0.529+0.389 = 0.918$

$-E_{Old} = 1.0*\log_2(1.0) = 0$

Entropy/IG Example *cont...*

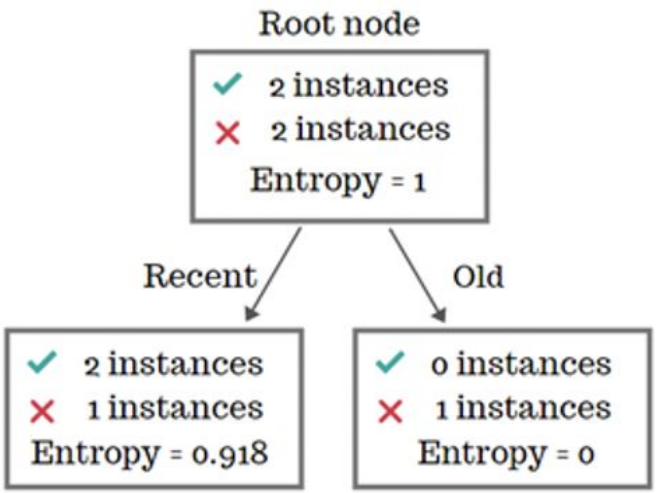
$$IG = E(S)_{parent} - E(S)_{children}$$

$$IG = E(S)_{parent} - WgtAve\text{-}E(S)_{children}$$

$$WgtAve\text{-}E(Children) = \tfrac{3}{4}(0.918) + \tfrac{1}{4}(0.0) = 0.688$$

$$IG = 1.0 - 0.688 = 0.3112$$

Age	Mileage	Road Tested	Buy
Recent	Low	Yes	Buy ✓
Recent	High	Yes	Buy ✓
Old	Low	No	Don't buy ✗
Recent	High	No	Don't buy ✗

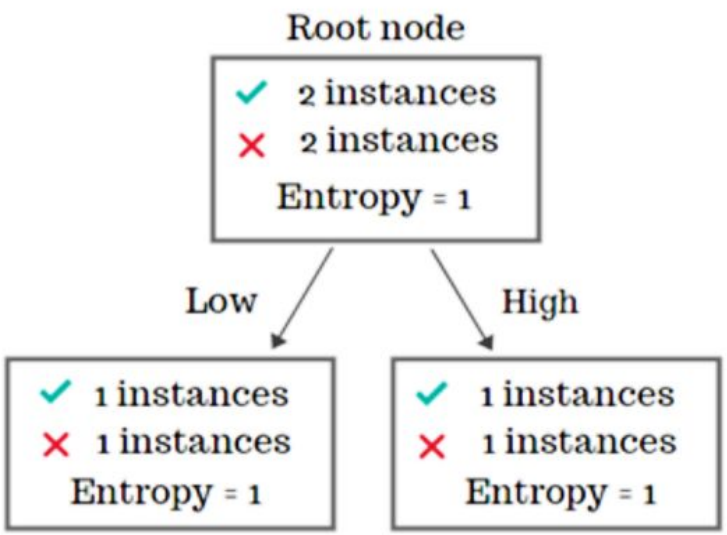


Entropy/IG Example *cont...*

$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$$

Information gain for Milage

Age	Mileage	Road Tested	Buy
Recent	Low	Yes	Buy ✓
Recent	High	Yes	Buy ✓
Old	Low	No	Don't buy ✗
Recent	High	No	Don't buy ✗



$$IG = E(S)_{parent} - E(S)_{children}$$

Children Entropy = 2/4 (1) + 2/4 (1) = 1

Information gain = 1 - 1 = 0

$-E_{Low} = 0.5*\log_2(0.5) + 0.5*\log_2(0.5) = 0.5*1.0+0.5*1.0 = 1.0$

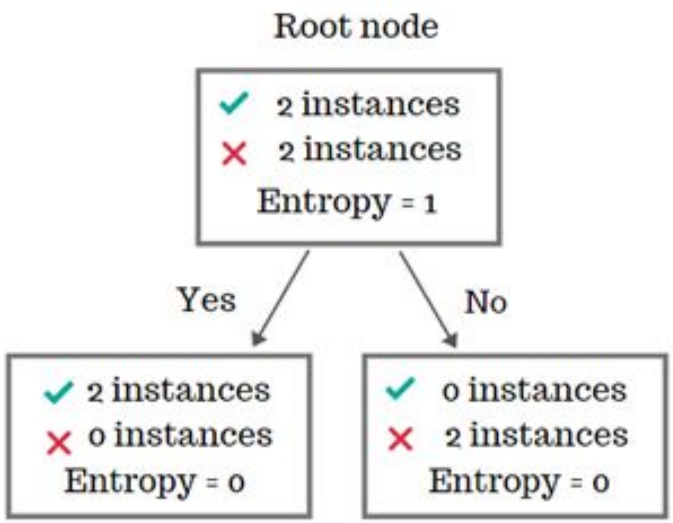
$-E_{High} = 0.5*\log_2(0.5) + 0.5*\log_2(0.5) = 0.5*1.0+0.5*1.0 = 1.0$

Entropy/IG Example *cont...*

$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$$

Information gain for Road Tested

Age	Mileage	Road Tested	Buy
Recent	Low	Yes	Buy ✓
Recent	High	Yes	Buy ✓
Old	Low	No	Don't buy ✗
Recent	High	No	Don't buy ✗



$$IG = E(S)_{parent} - E(S)_{children}$$

Children Entropy = 2/4 (0) + 2/4 (0) = 0

Information gain = 1 - 0 = 1

$-E_{Yes} = 1.0*\log_2(1.0) = 0.0$

$-E_{No} = 1.0*\log_2(1.0) = 0.0$

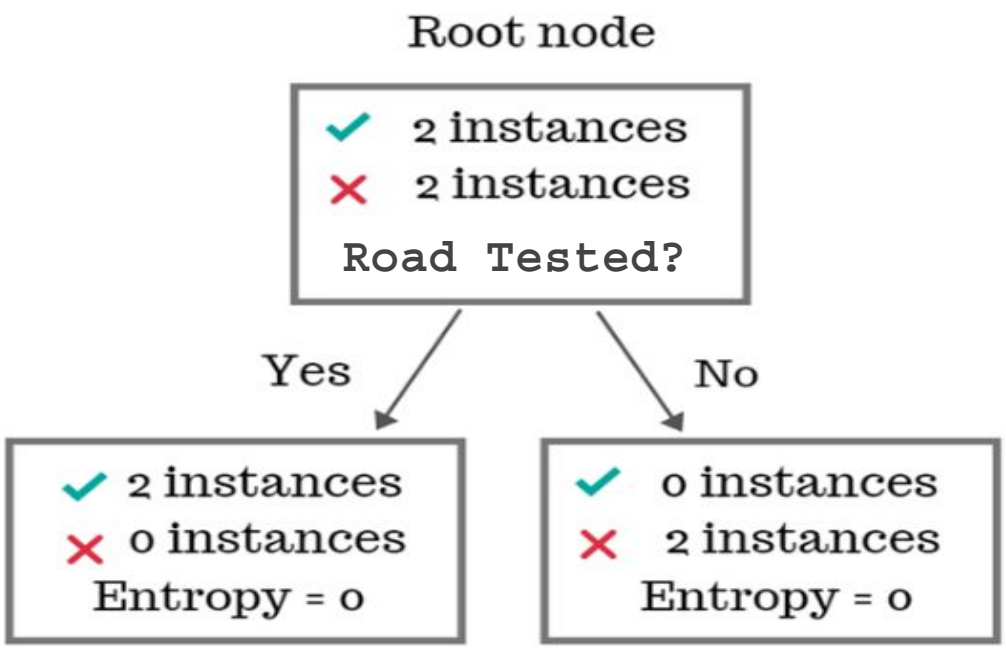
Entropy/IG Example *cont...*

	Age	Mileage	Road Tested	Buy
	Recent	Low	Yes	Buy
	Recent	High	Yes	Buy
	Old	Low	No	Don't buy
	Recent	High	No	Don't buy
Info. Gain	0.3112	0	1	

Maximum IG

Result

- **The maximum information gain is for the feature 'Road Tested' so this would be the Root node**



Other Cost Measures (Gini Index)

- **The Gini index measures the degree or probability of a particular variable being wrongly classified when it is randomly chosen**
- **Gini index is between 0 and 1**
- **GI = 0 when all elements belong to a certain class and GI = 1 when elements are randomly distributed across various classes**
- **p_i = probability of being in class i for a given cutoff/class**

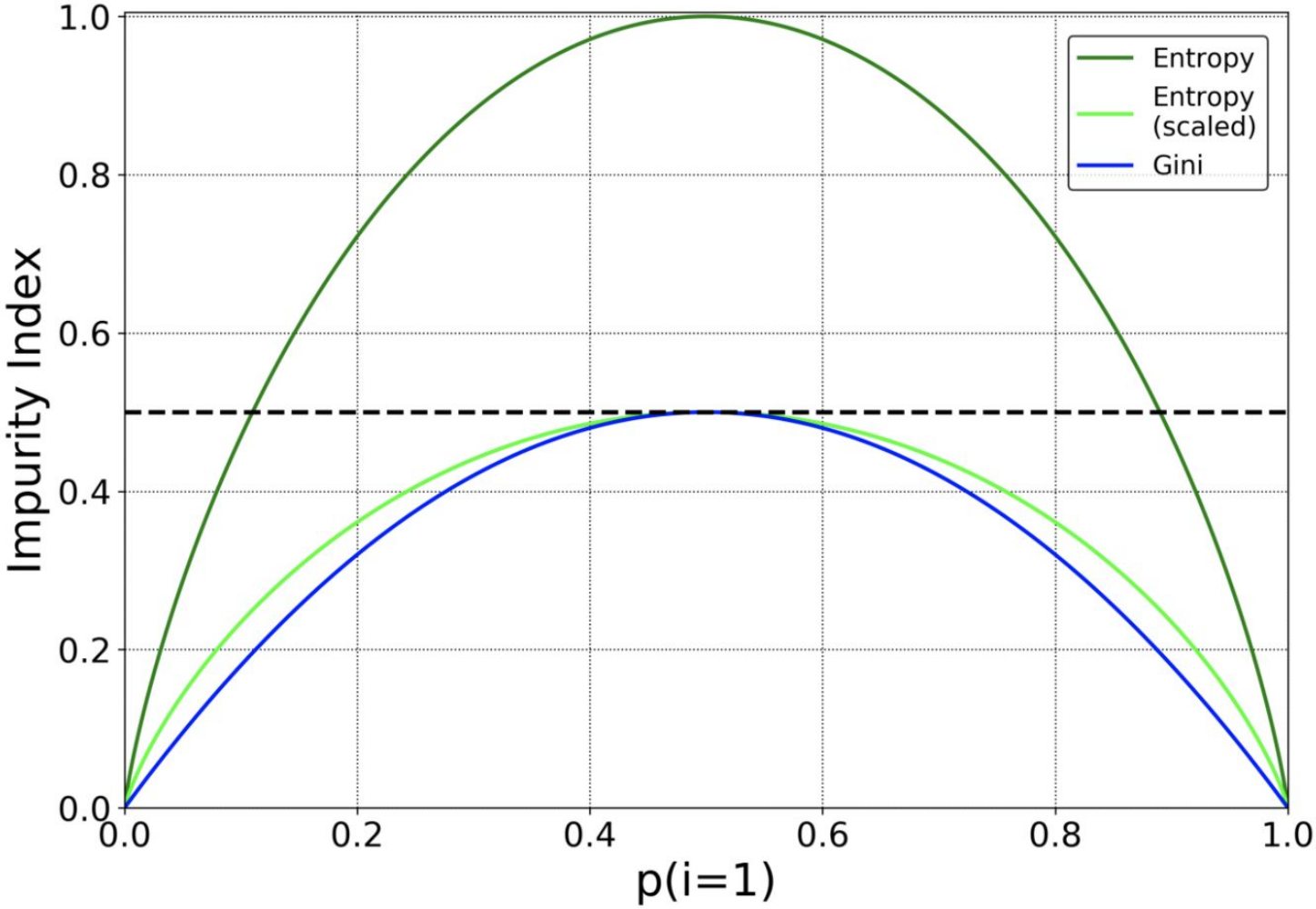
$$\text{Gini} = 1 - \sum_{i=1}^{n} (p_i)^2$$

Gini Index is not as expensive to calculate as E(S) or IG
The split with the lowest GI is the one chosen as the tree root

Feature Selection & Cost Measures

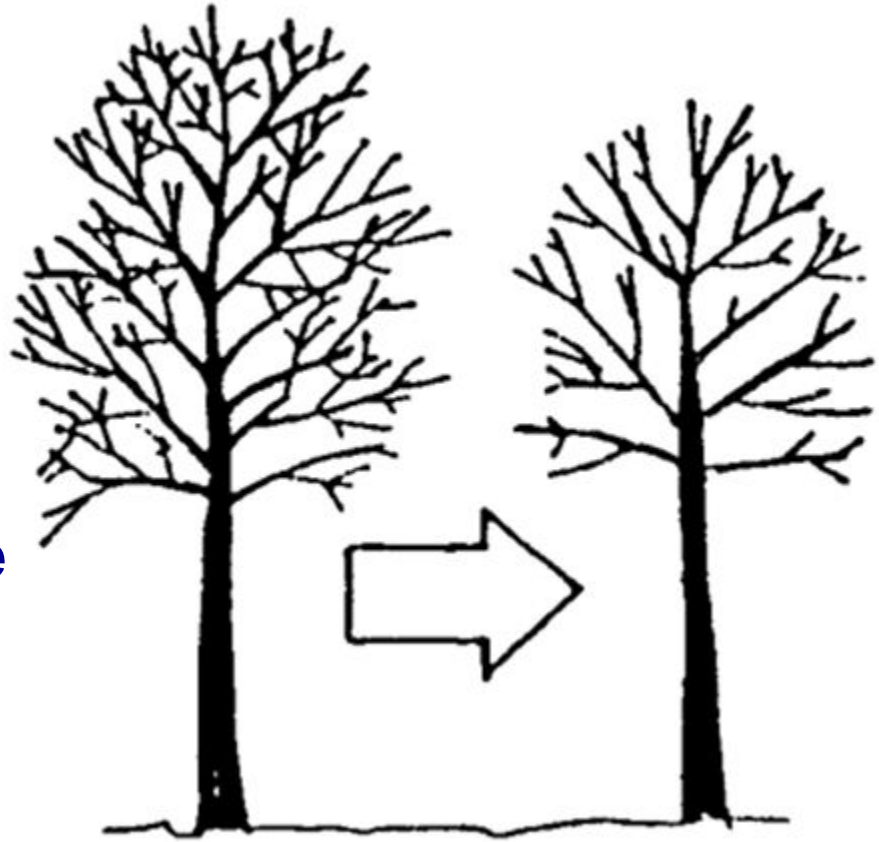
- Information Gain (IG) and Gini Index (GI) are two methods to perform feature selection in a data set being processed by a DT
- Features with the highest information gain or lowest Gini index are most useful and are always placed at the root of a DT
- Recall that feature selection is also way of reducing the amount of data used in training an ML model
- Adding feature selection at the beginning of the DT process is a way of pruning your tree

Gini Index vs. Entropy



Pruning Trees

- Involves removing branches that use features of low importance
- Improves decision tree performance
- Reduces overfitting
- Reduces complexity of the decision tree
- Two methods:
  - Reduced error pruning
  - Weakest link pruning

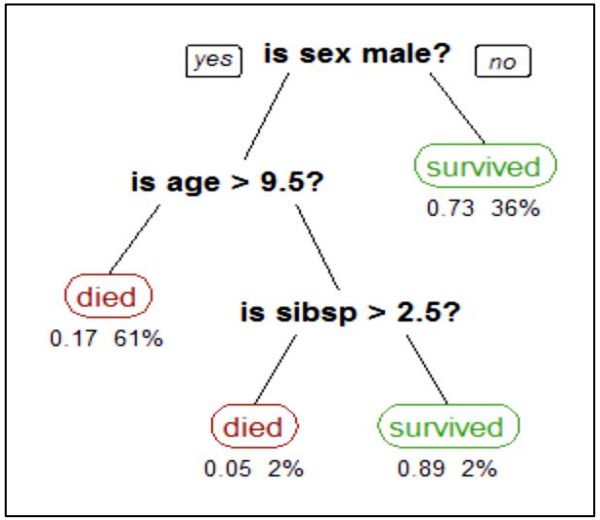


Feature Selection & Cost Measures

All Features

Feature Selection

Final Features



Titanic Survivor Data

Passenger ID	Survived	Male/Female	Age	Sibsize
0001	0	0	32	0
0002	1	1	7	4
0003	0	0	67	1
0004	0	0	3	5
0005	1	1	45	2
0006	0	1	54	1
0007	1	0	53	1
0009	1	1	10	2
…	…	…	…	…
1317	1	0	6	4
	Useful Label	Useful Feature	Useful Feature	Useful Feature

Titanic Survivor Data

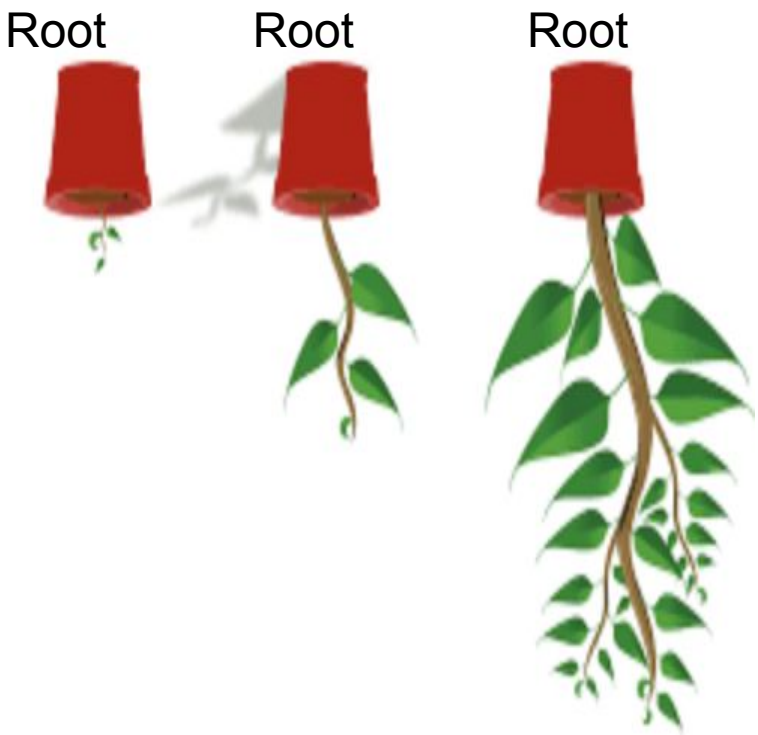
Passenger ID	Survived	Male/Female	Age	Sibsize	Zodiac Sign
0001	0	0	32	0	3
0002	1	1	7	4	12
0003	0	0	67	1	1
0004	0	0	3	5	5
0005	1	1	45	2	6
0006	0	1	54	1	3
0007	1	0	53	1	12
0009	1	1	10	2	8
…	…	…	…	…	…
1317	1	0	6	4	10
	Useful Label	Useful Feature	Useful Feature	Useful Feature	Useless Feature

Feature Selection

Passenger ID	Survived	Male/Female	Age	Sibsize	Zodiac Sign
0001	0	0	32	0	3
0002	1	1	7	4	12
0003	0	0	67	1	1
0004	0	0	3	5	5
0005	1	1	45	2	6
0006	0	1	54	1	3
0007	1	0	53	1	12
0009	1	1	10	2	8
…	…	…	…	…	…
1317	1	0	6	4	10
		GI = 0.11	GI = 0.23	GI = 0.28	GI = 0.98

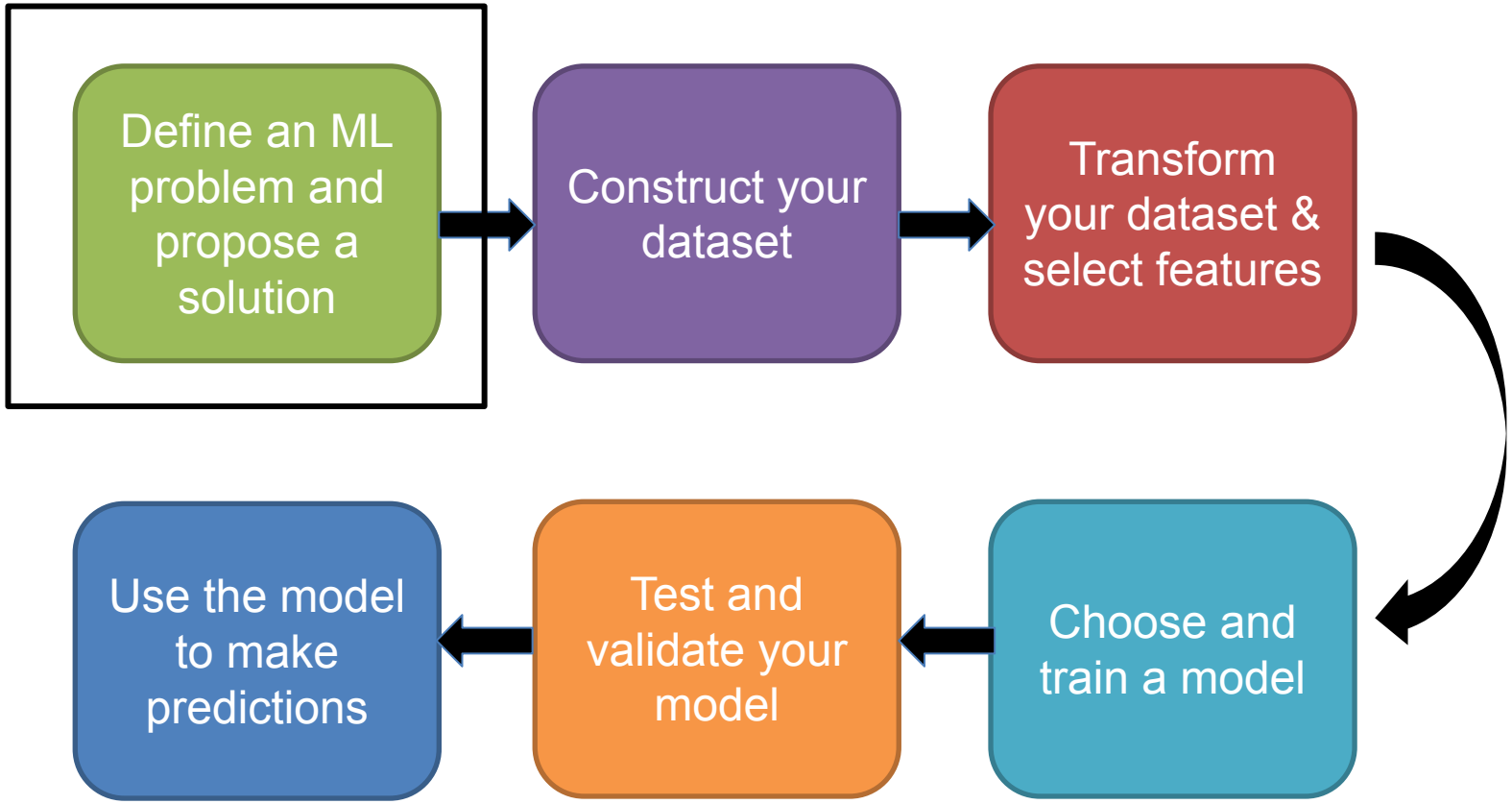
Recursive Binary Splitting When to Stop Splitting?

- Set a minimum number of training inputs to use in each leaf (minimum number of affected objects per decision)
- Set a maximum depth to model (length of the longest path from the root)
- Depth should be a small fraction of total number of features
- Use feature selection to help set depth



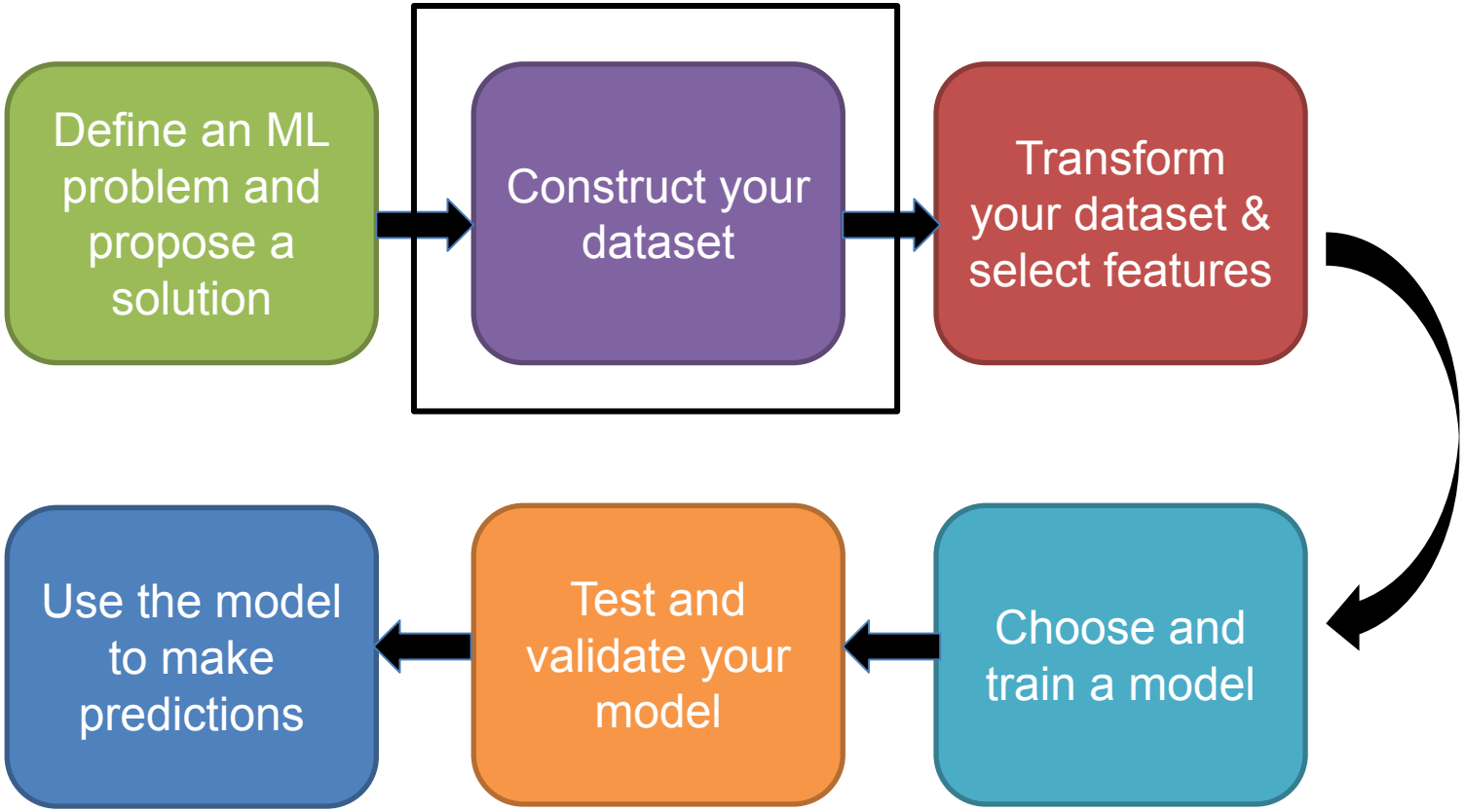
Let’s Try A Real Example

Machine Learning Workflow



How do I classify Iris flowers in my area based on their floral dimensions?

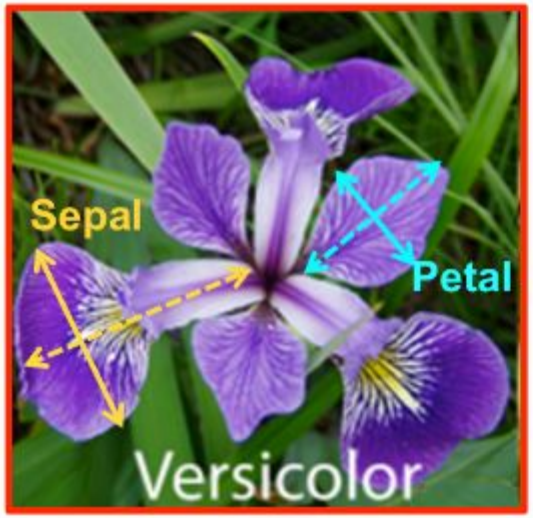
Machine Learning Workflow



Find a good training/testing data set

Iris Flower Data Set

- Iris flower data set introduced in 1936 by Ronald Fisher (Statistician) to demonstrate linear discriminant analysis
- Consists of 50 samples each of the Iris species: *Iris setosa, Iris virginica, Iris versicolor*
- Four features measured from each sample, length and width of sepals and petals in cm
- Different iris species can be differentiated by their petal/sepal dimensions



Iris Flower Data Set

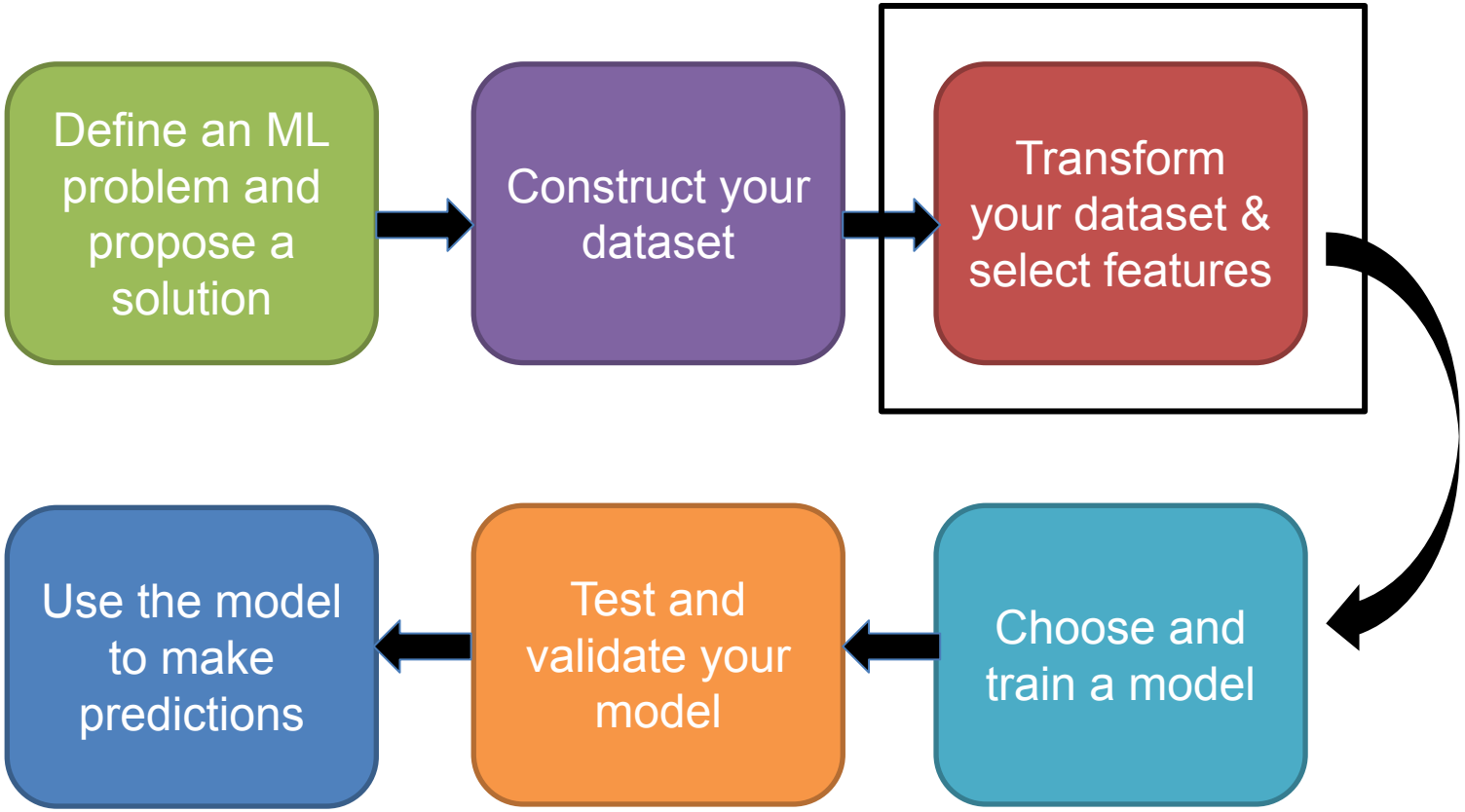
Table I

Iris setosa				*Iris versicolor*				*Iris virginica*			
Sepal length	Sepal width	Petal length	Petal width	Sepal length	Sepal width	Petal length	Petal width	Sepal length	Sepal width	Petal length	Petal width
5·1	3·5	1·4	0·2	7·0	3·2	4·7	1·4	6·3	3·3	6·0	2·5
4·9	3·0	1·4	0·2	6·4	3·2	4·5	1·5	5·8	2·7	5·1	1·9
4·7	3·2	1·3	0·2	6·9	3·1	4·9	1·5	7·1	3·0	5·9	2·1
4·6	3·1	1·5	0·2	5·5	2·3	4·0	1·3	6·3	2·9	5·6	1·8
5·0	3·6	1·4	0·2	6·5	2·8	4·6	1·5	6·5	3·0	5·8	2·2
5·4	3·9	1·7	0·4	5·7	2·8	4·5	1·3	7·6	3·0	6·6	2·1
4·6	3·4	1·4	0·3	6·3	3·3	4·7	1·6	4·9	2·5	4·5	1·7
5·0	3·4	1·5	0·2	4·9	2·4	3·3	1·0	7·3	2·9	6·3	1·8
4·4	2·9	1·4	0·2	6·6	2·9	4·6	1·3	6·7	2·5	5·8	1·8

Text from original 1936 paper – note the actual data was collected in 1935 by a botanist working in the Gaspe region of Quebec (Edgar Anderson)

Setosa has small petals, *Virginica* has long, wide petals, *Versicolor* is intermediate

Machine Learning Workflow



Format Data/Select Features

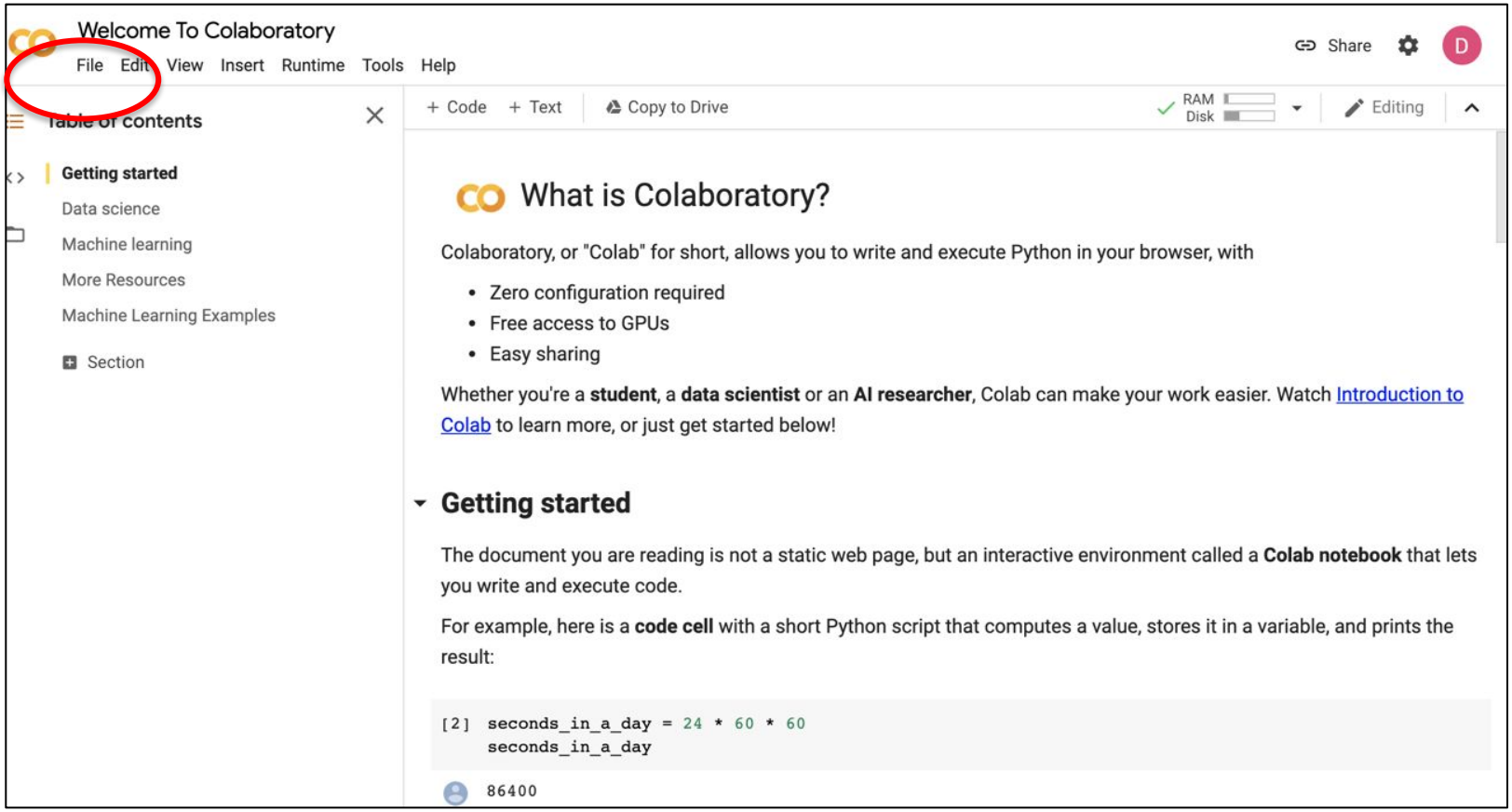
Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Species
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
7	3.2	4.7	1.4	Iris-versicolor
6.4	3.2	4.5	1.5	Iris-versicolor
6.9	3.1	4.9	1.5	Iris-versicolor
5.5	2.3	4	1.3	Iris-versicolor
6.5	2.8	4.6	1.5	Iris-versicolor
6.3	3.3	6	2.5	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica
7.1	3	5.9	2.1	Iris-virginica
6.3	2.9	5.6	1.8	Iris-virginica
6.5	3	5.8	2.2	Iris-virginica

Let's Try Programming Decision Tree to Classify Iris Flowers

IrisDT4.ipynb

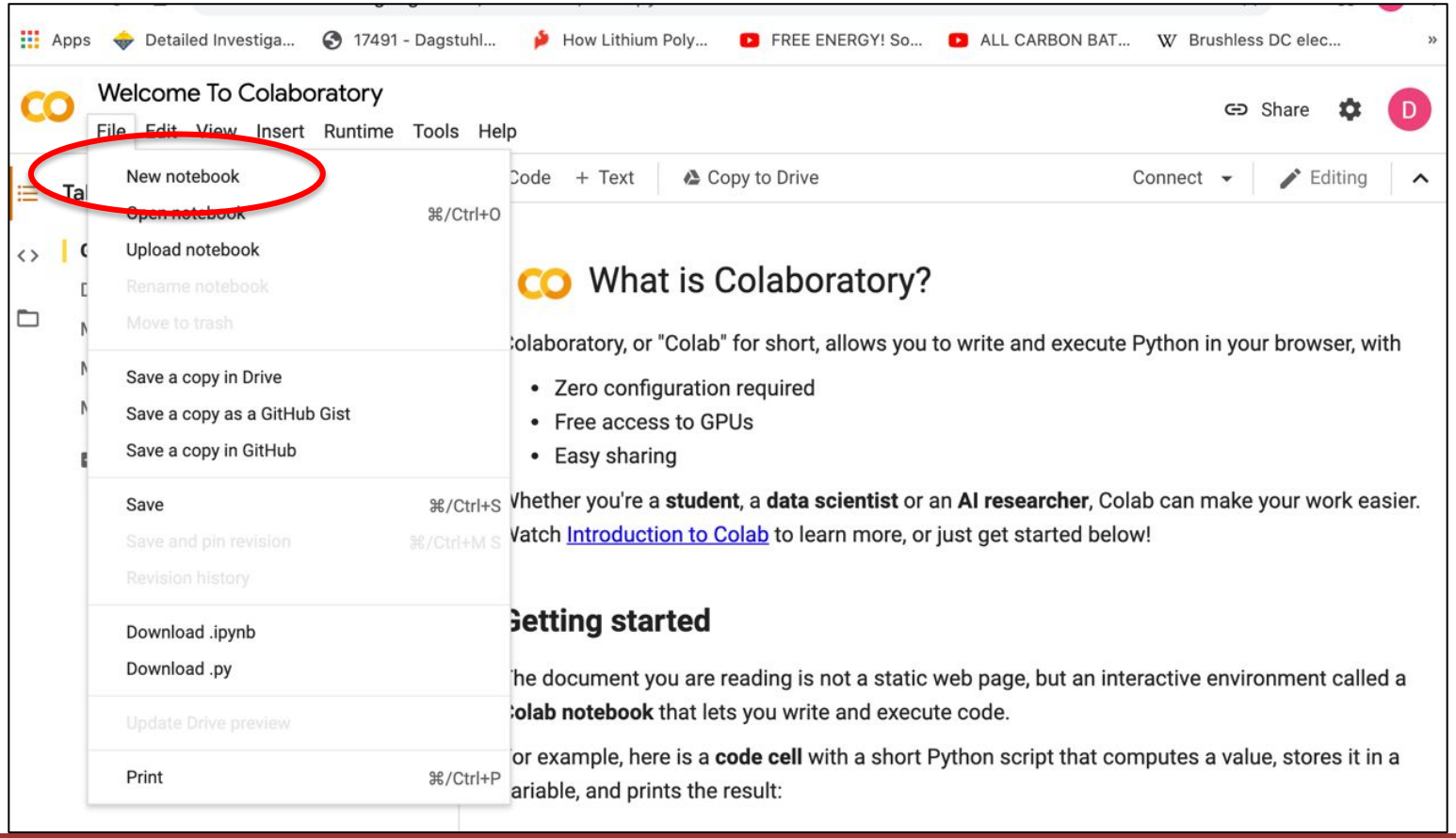
Create A Google Colab Notebook

- **Step 1: Go to Colab (Google Colab)**



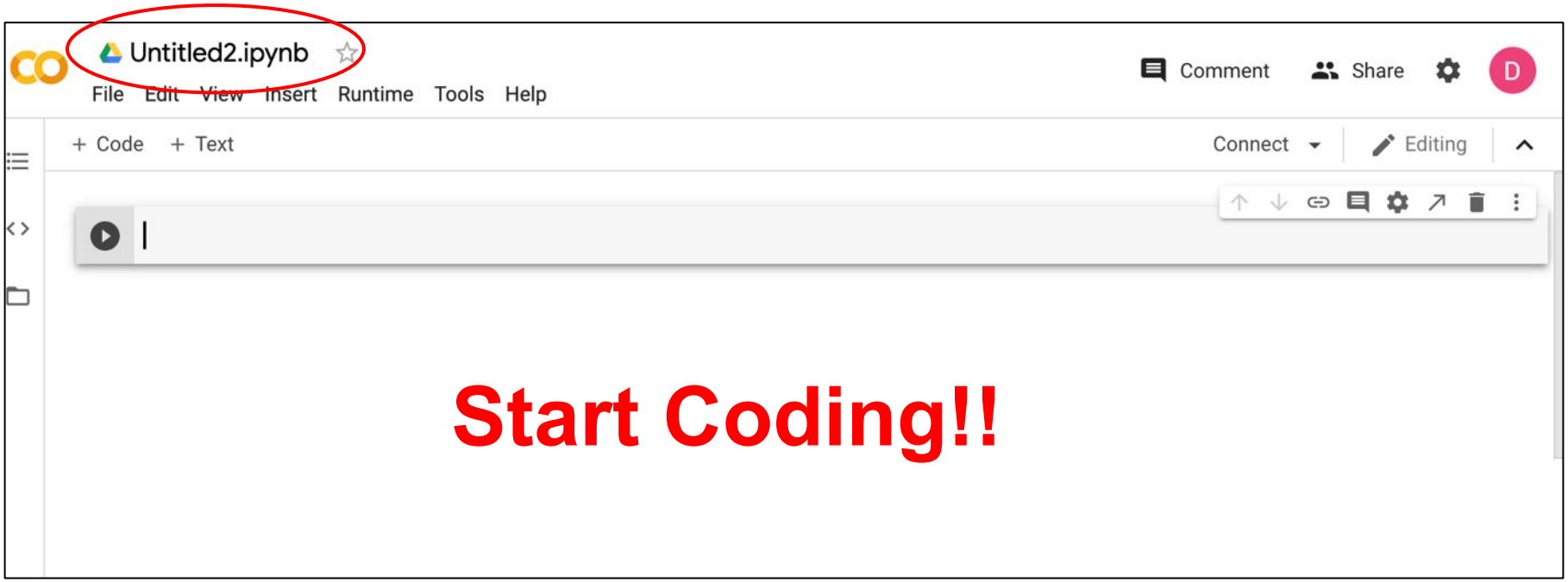
Open a New Notebook

- **Step 2: Go to File and Select "New notebook"**



Rename the Notebook

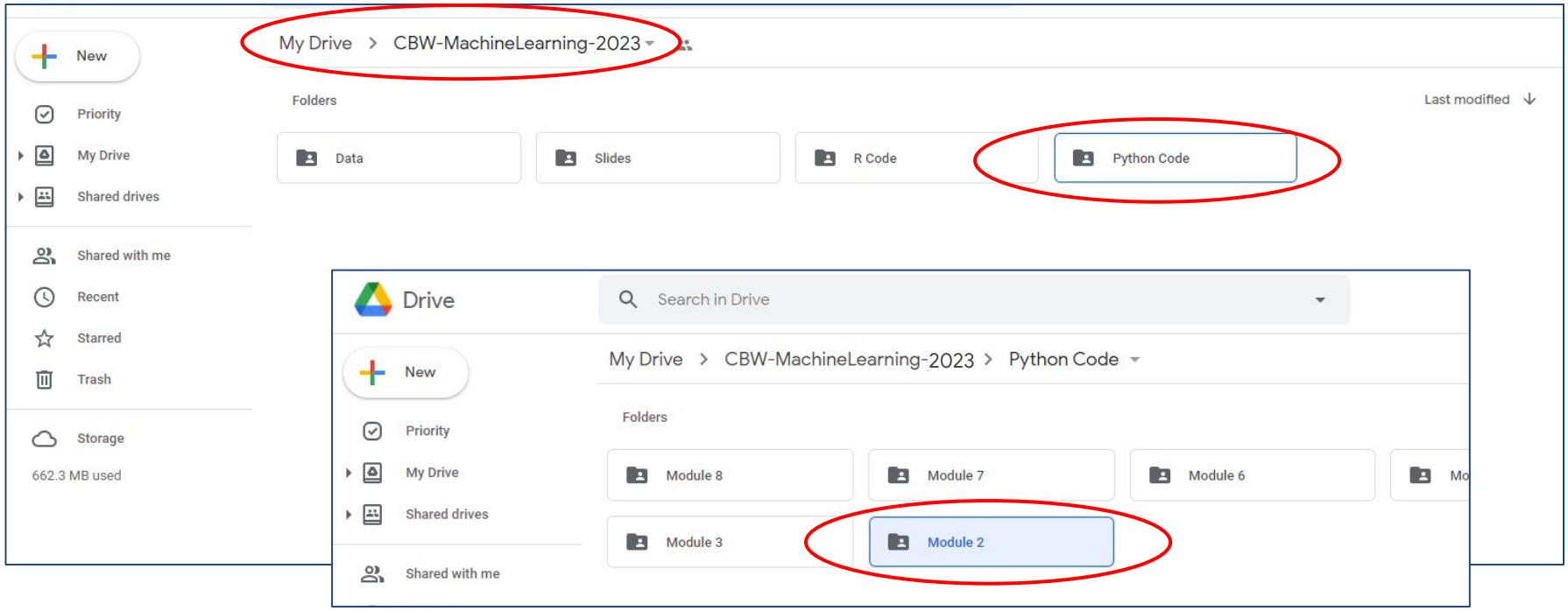
- **Rename the notebook by clicking on the name and type in IrisDT4.ipynb in the red circle as shown below**



To Save Time….

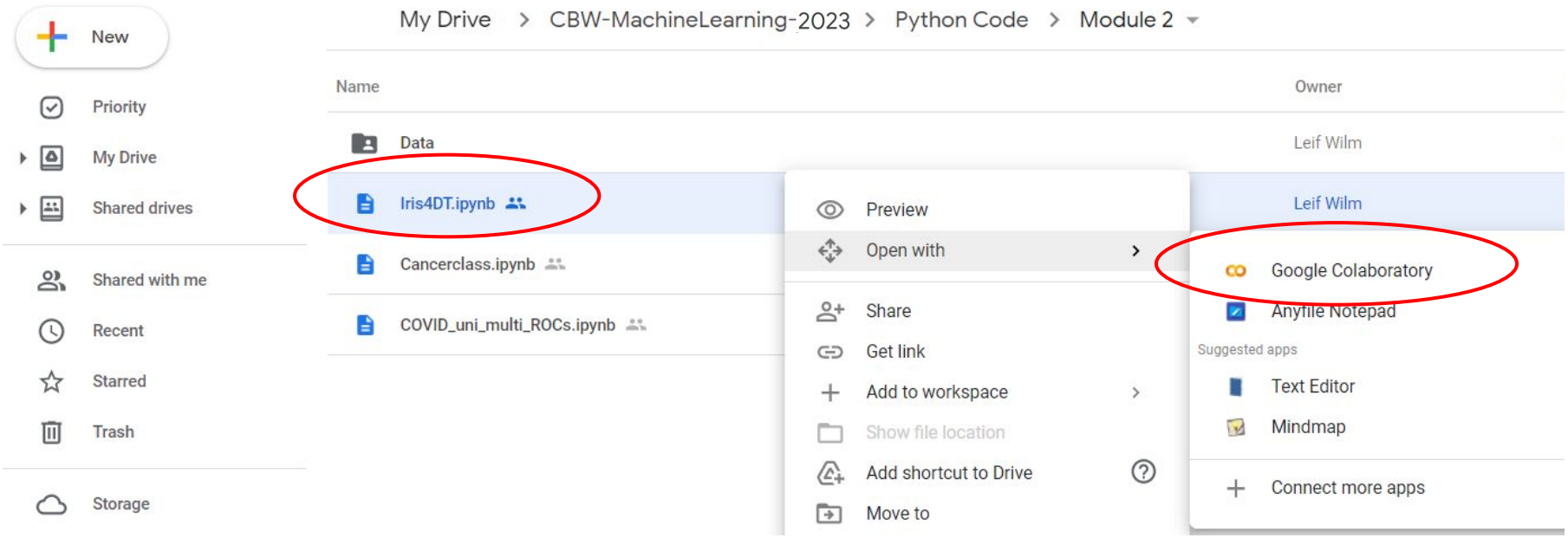
Open the Colab File

- **Find the Module 2 folder under Python Code in folder CBW-MachineLearning-2023 in your Google Drive**



Open the Colab File *cont...*

- **Right click on 'IrisDT4.ipynb' and select open with Google Colaboratory**



General Algorithm

- Read data
- Check data
- Create training/testing data sets
- Create a test splitting function (test_split)
- Create a Gini Index calculation function (gini_index)
- Create optimal split function (get_split)
- Create terminal node function (to_terminal)
- Create a recursive splitting function (split)
- Create a program to call the trained DT

Import Functions for Python Math

- Numpy allows for mathematical operations and array handling
- Pandas is used to read data and for providing dataframe capabilities

```
import numpy as np
import pandas as pd
```

Code for Reading Data In

```
data = pd.read_csv('data1.csv')
#data.head()
data.loc[np.r_[0:3, 51:53, 101:103], :]
```

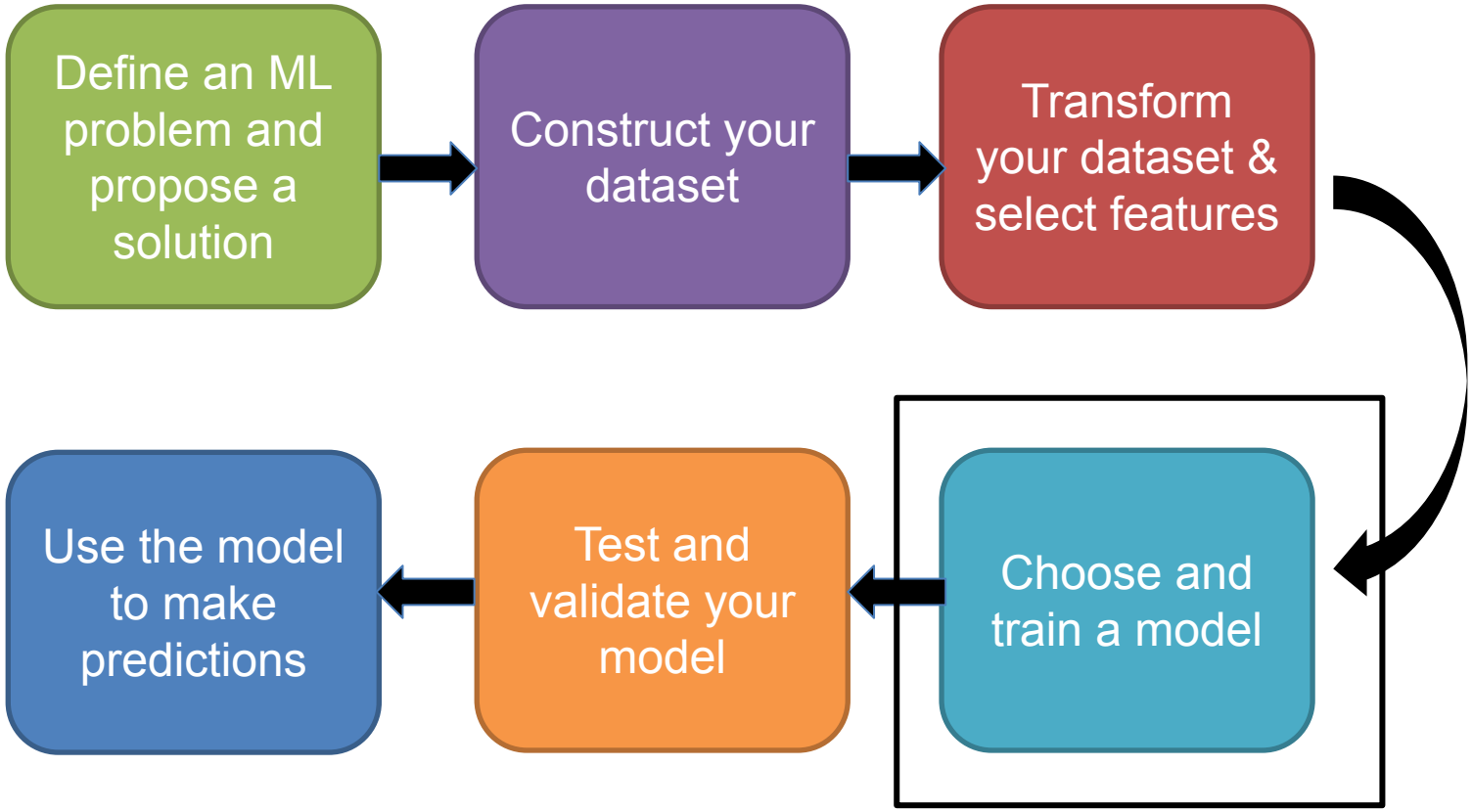
	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
51	6.4	3.2	4.5	1.5	Iris-versicolor
52	6.9	3.1	4.9	1.5	Iris-versicolor
101	5.8	2.7	5.1	1.9	Iris-virginica
102	7.1	3.0	5.9	2.1	Iris-virginica

Data Check (Missing Values, Label Check)

- **A verify dataset function is defined, which will check to make sure that there is no missing data**

```
def verify_dataset(data):
#Use data_found as a dummy variable to determine if to print
missing value information
  data_found = 1
  for each_column in data.columns:
    if data[each_column].isnull().any():
            print("Data missing in Column " + each_column)
            data_found = 0
            quit()
    if data_found == 1:
        print("Dataset is complete. No missing value")
    return
#Call verify_dataset and check data
verify_dataset(data)
```

Machine Learning Workflow



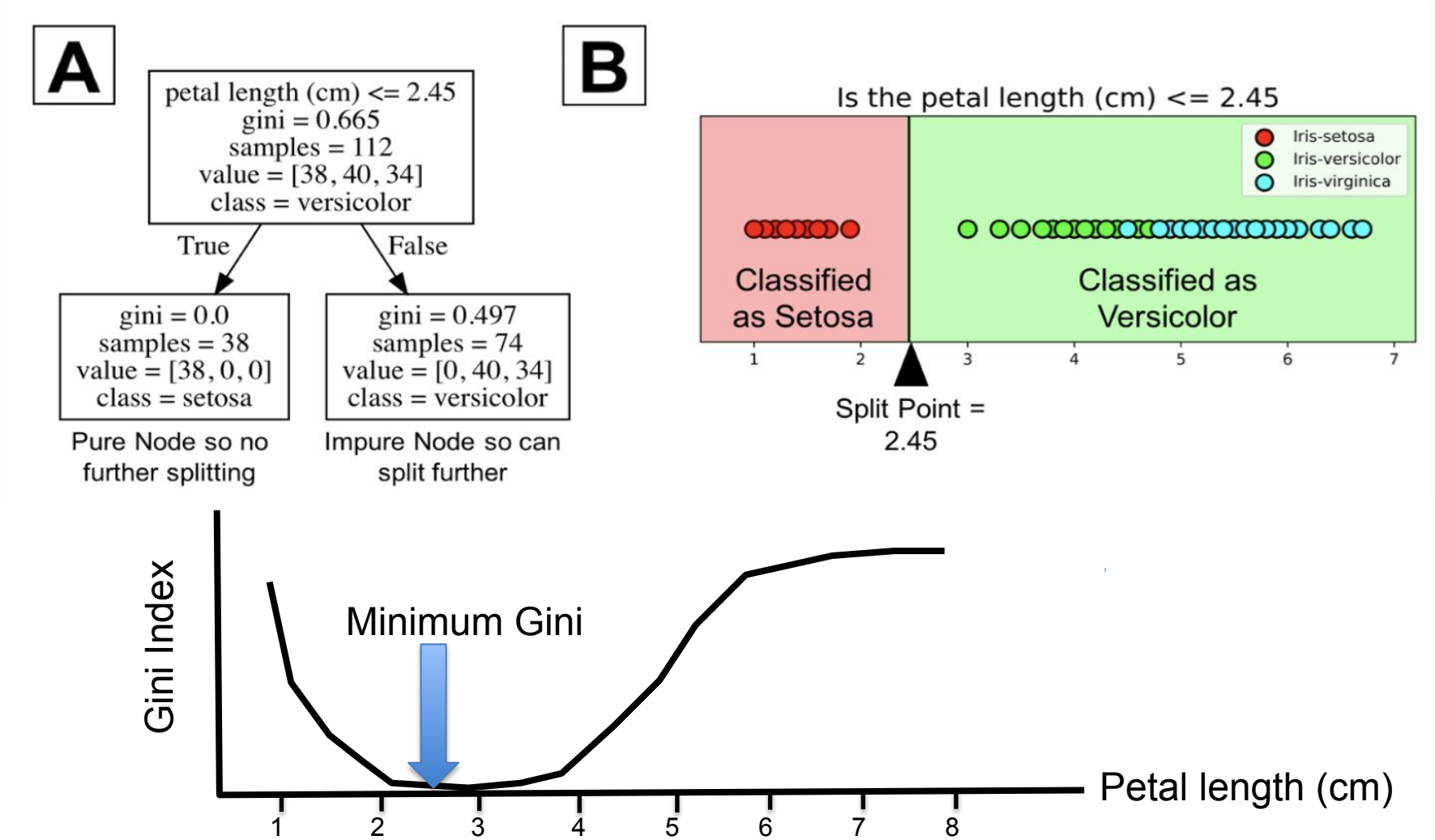
We have chosen an RBS Decision Tree Model

Create a Training & Testing Set

- **The data set of 150 flowers is divided into 70% for training and 30% for testing (3-fold cross validation)**

```
#Splitting The Database in training and testing
def split_dataset_test_train(data):
#Use the .sample() function to scramble the data set
    data = data.sample(frac=1).reset_index(drop=True)
#Determine the integer location (iloc) from beginning of array (:)
#to 0.7*150 and do a "cleanup" with a reset call
    training_data = data.iloc[:int(0.7 * len(data))].reset_
    index(drop=True)
#Determine the integer location (iloc) from 0.7*150 to end of
#array (: ) and do a "cleanup" with a reset call
    testing_data = data.iloc[int(0.7 * len(data)):].reset_
    index(drop=True)
    return [training_data, testing_data]

testtrain = split_dataset_test_train(data)
print(testtrain)
```

Calculating the Gini Index Over Different Split Points



Creating Data Splits

- Before calculating the gini index over split points, the data must first be split
- This function will split the rows of data given the feature (index) and cutoff (value)
- The values of feature and cutoff are iterated through in the main splitting function

```
def test_split(index, value, dataset):
  #Initializing empty lists that will contain the split data
  left, right = list(), list()
  #For each row in the data, split the data according to the
  #value of the feature being examined
  for row in dataset:
    if row[index] < value:
      left.append(row)
    else:
      right.append(row)
  #The two groups are returned
  return left, right
```

Calculate Gini Index for a Given Split (Pt. 1)

- A function gini_index will calculate the gini index value for given splits of data
- The input 'groups' are the left and right data groups returned by the 'test_split' function
- The input 'classes' are the three labels for the classes 0,1, and 2
- A check is performed to make sure that we do not perform gini index calculation for an empty group (if no data meets the cutoff criteria in the 'test_split' function)

```
# Calculate the Gini index for a split dataset
def gini_index(groups, classes):
  # count all samples at split point
  n_instances = float(sum([len(group) for group in groups]))
  # sum weighted Gini index for each group
  gini = 0.0
  for group in groups:
    size = float(len(group))
    # avoid divide by zero
    if size == 0:
      continue
```

Calculate Gini Index for a Given Split (Pt. 2)

- A variable 'score' is initialized which will contain the summation term of the final gini index calculation
- Summation is performed over each of the class values 0,1,2
- The function returns the final gini index

```
# Calculate the Gini index for a split dataset
def gini_index(groups, classes):
...
    score = 0.0
    # score the group based on the score for each class
    for class_val in classes:
      p = [row[-1] for row in group].count(class_val) / size
      score += p * p
    # weight the group score by its relative size
    gini += (1.0 - score) * (size / n_instances)
  return gini
```

Determine Optimal Splits (Pt. 1)

- To determine the optimal split points for the features, a 'get_split' function is defined which implements the previous 'gini_index' and 'test_split' functions
- The best index (feature), value (cutoff), score (gini index), and groups are initialized
- For each feature, the min and max values are found

```
#Determine optimal split points for features using the lowest gini index
def get_split(dataset):
  class_values = [0,1,2]
  b_index, b_value, b_score, b_groups = 999, 999, 999, None

  for feature in range(len(dataset[0])-1):
    max_value = dataset[:][feature].max()
    min_value = dataset[:][feature].min()
```

Determine Optimal Splits (Pt. 2)

- The cutoff value is incremented from the min to max by 0.1 increments
- At each increment, the gini index is calculated for the test split
- Each calculated gini index is tested to see if it is lower than the previously calculated gini index
- The feature, cutoff, and data groups corresponding to the lowest gini index are returned

```
#Determine optimal split points for features using the lowest gini index
def get_split(dataset):
...
    for cutoff in np.arange(min_value, max_value, 0.1):
      groups = test_split(feature, cutoff, dataset)
      gini = gini_index(groups, class_values)
      if gini < b_score:
        b_index, b_value, b_score, b_groups = feature,cutoff,gini,groups
  return {'index':b_index, 'value':b_value, 'groups':b_groups}
```

Terminal Nodes

- To decide when to stop growing the tree, a maximum depth parameter is used
- The maximum depth parameter is the number of node layers starting from the root node
- When the tree stops growing, we create terminal nodes, which will provide the final class prediction given certain values for features
- To accommodate the possibility that the data is not split perfectly, predictions are based on the most common class value in the terminal group

```
# Create a terminal node value
def to_terminal(group):
 #Extracting the last value of each row (class label)
  outcomes = [row[-1] for row in group]
 #Return the most common label in the group
  return max(set(outcomes), key=outcomes.count)
```

Recursive Splitting (Pt. 1)

- To build the tree, splits are performed repeatedly using the 'get_split' function
- The first split results in left and right child nodes (groups of data). A split is then performed on each of the nodes & the process repeats
- A terminal node is created if either the left or right child node is empty of data, or the maximum depth is reached
- 'min_size' is used to force a terminal node if a data group contains too few samples for splitting
- To perform this recursive process, we define a function 'split' which will call itself

```
# 'Node' contains the feature, groups, and optimal cutoff
#returned by the 'get_split' function
def split(node, max_depth, min_size, depth):
  left, right = node['groups']
  del(node['groups'])
  # check for a no split
  if not left or not right:
    node['left'] = node['right'] = to_terminal(left + right)
    return
```

Recursive Splitting (Pt. 2)

- After the empty group check is performed, we check to see if the maximum depth has been reached
- If the current depth has reached the maximum depth, we create terminal nodes and stop the recursive splitting
- If max depth has not been reached, we then force the left child to a terminal node if it contains too few samples for splitting
- Otherwise, we proceed with splitting, calling the 'get_split' function on the left group and the calling 'split' on this node (recursion)

```
def split(node, max_depth, min_size, depth):
...
# check for max depth
  if depth >= max_depth:
    node['left'], node['right'] = to_terminal(left), to_terminal(right)
    return
  # process left child
  if len(left) <= min_size:
    node['left'] = to_terminal(left)
  else:
    node['left'] = get_split(left)
    split(node['left'], max_depth, min_size, depth+1)
```

Recursive Splitting (Pt. 3)

- The right child groups are also processed in the same fashion
- When the 'split' function calls itself on the new child groups, the current depth is incremented by one
- This recursive function will stop executing when either maximum depth is reached, or either the left or right child nodes contain no data

```
# process right child
if len(right) <= min_size:
  node['right'] = to_terminal(right)
else:
  node['right'] = get_split(right)
  split(node['right'], max_depth, min_size, depth+1)
```

Making Predictions

- To make predictions, a row of data containing values for the features is used to navigate the tree to a terminal node
- This is implemented with another recursive function that will first check if the current node is a terminal node
- If not, the function will call itself repeatedly until the correct terminal node is found

```
# Make a prediction with a decision tree
def predict(node, row):
  if row[node['index']] < node['value']:
    if isinstance(node['left'], dict):
      return predict(node['left'], row)
    else:
      return node['left']
  else:
    if isinstance(node['right'], dict):
      return predict(node['right'], row)
    else:
      return node['right']
```

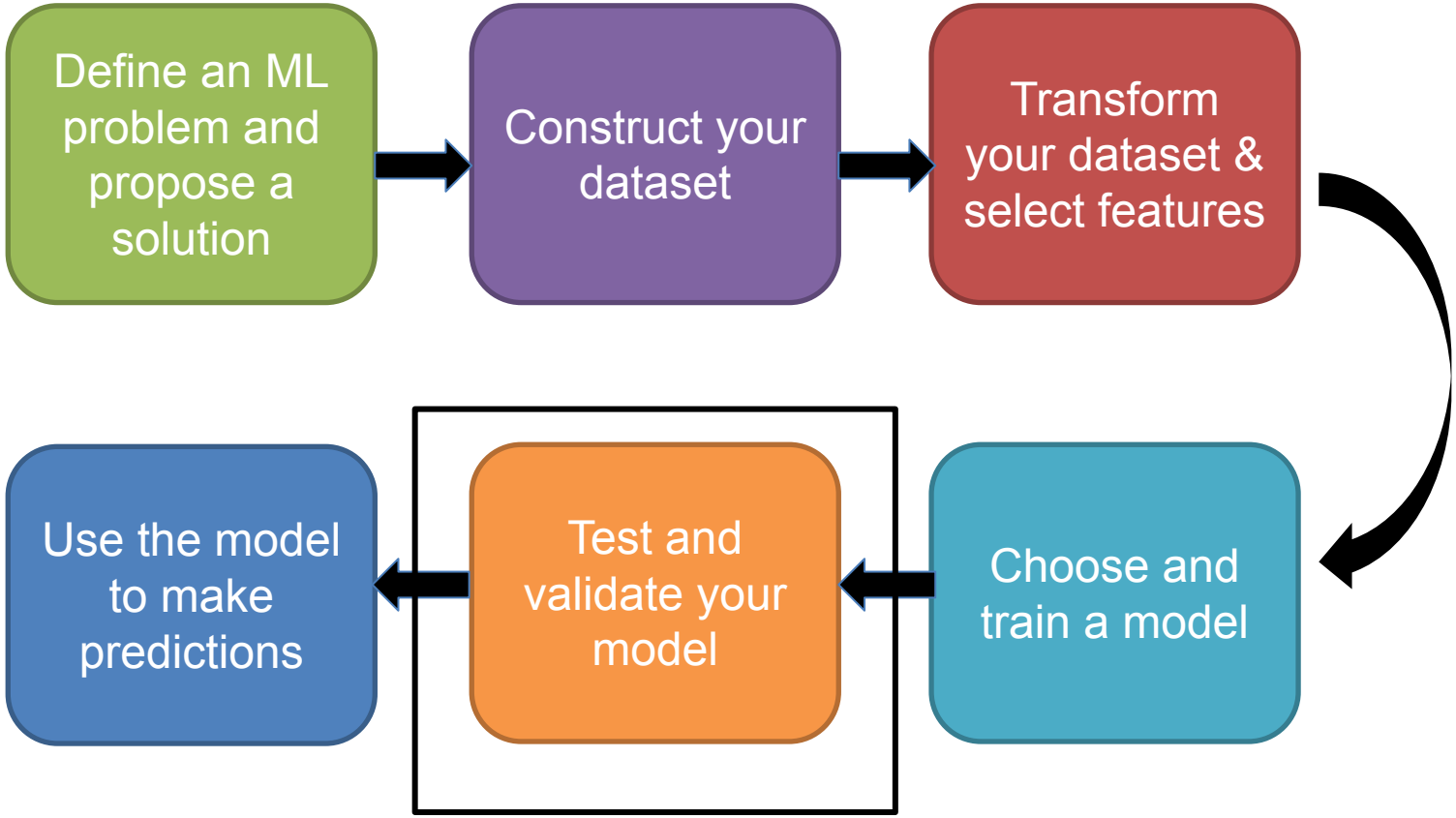
Program Statistics

- Total number of lines: 123
- Total number of comment lines: 32
- Total number of coding lines: 91
- Time to train on training data: 1.5 sec
- Typical time to train and perform test run with program: 2.5 sec

Performance on Training Set (Confusion Matrix)

	Setosa (Observed)	Virginica (Observed)	Versicolor (Observed)
Setosa (Predicted)	1	0	0
Virginica (Predicted)	0	1	0
Versicolor (Predicted)	0	0	1

Machine Learning Workflow



Test on Training Data Set

Performance on Testing Set (Confusion Matrix)

	Setosa (Observed)	Virginica (Observed)	Versicolor (Observed)
Setosa (Predicted)	1	0	0
Virginica (Predicted)	0	1	0
Versicolor (Predicted)	0	0.07	0.93

Comparison Between Training & Testing

Training

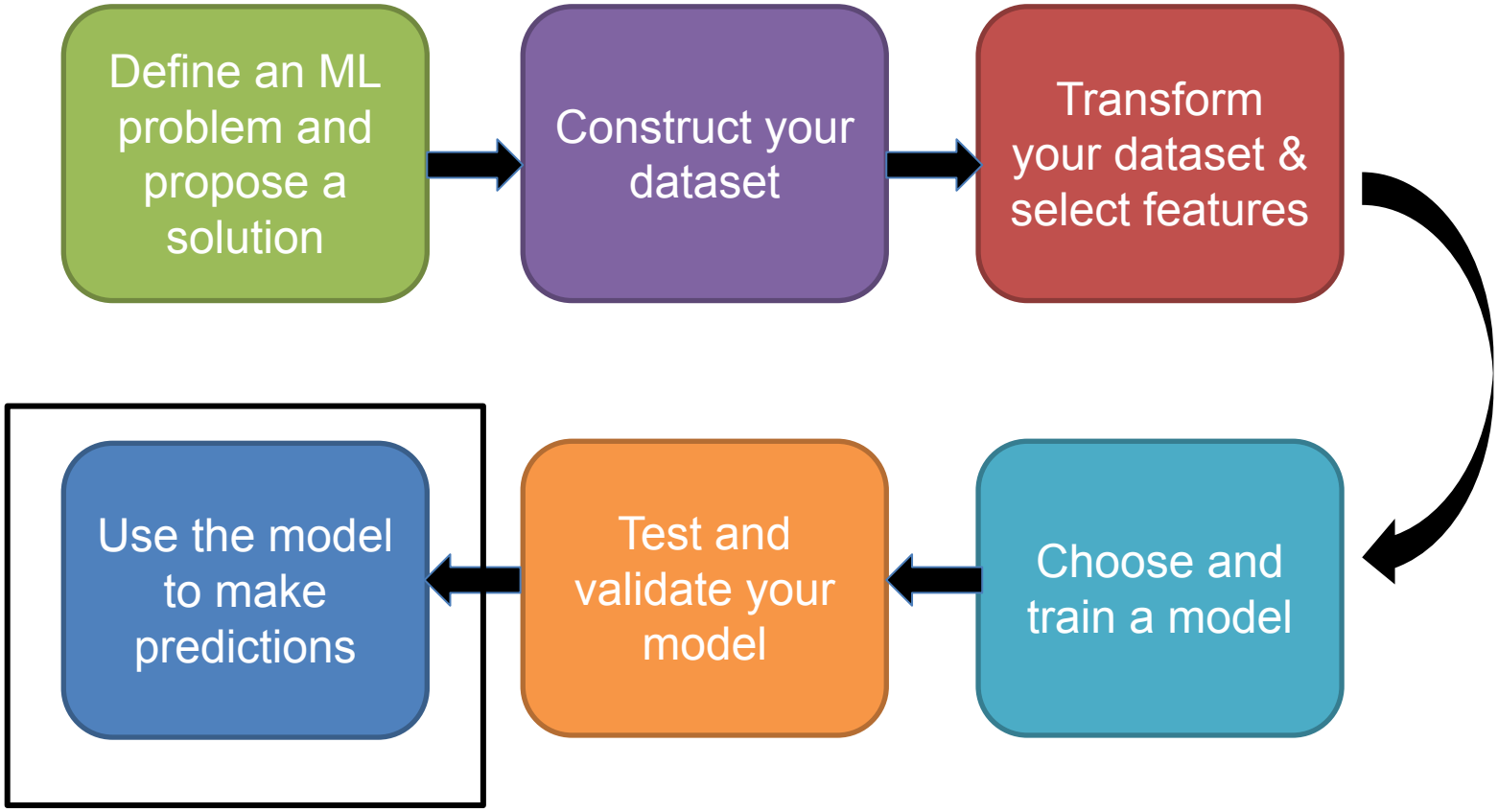
	Setosa (Observed)	Virginica (Observed)	Versicolor (Observed)
Setosa (Predicted)	1	0	0
Virginica (Predicted)	0	1	0
Versicolor (Predicted)	0	0	1

Testing

	Setosa (Observed)	Virginica (Observed)	Versicolor (Observed)
Setosa (Predicted)	1	0	0
Virginica (Predicted)	0	1	0
Versicolor (Predicted)	0	0.07	0.93

Normally you want training & testing sets to be within 5-6% of each other
If they are much more than this, you have likely over-trained the system

Machine Learning Workflow



Summary

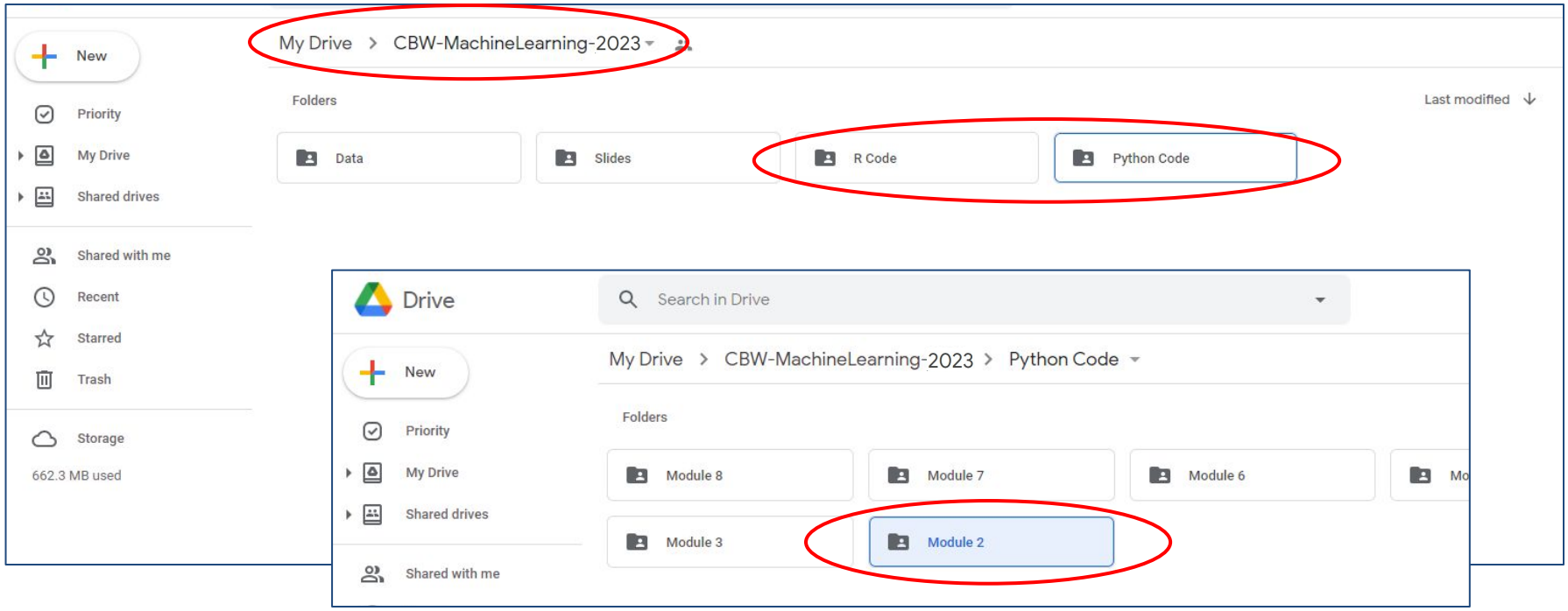
- We now have a decision tree program written in "pure" python that predicts iris flower classes
- It has been thoroughly trained on a training set of 105 flowers and tested on a test set of 45
- This is fairly generic code so you could use or adapt this code for many other kinds of classification problems such as patients (cases or controls) with different levels of gene, protein or metabolite expression
- In the next section we will explore the code and run it on a few examples
- If you would like to work with in R rather than Python, follow the steps indicated in the Lab to find the corresponding scripts

Module 2 - Lab

Let’s Take a Look at the Program

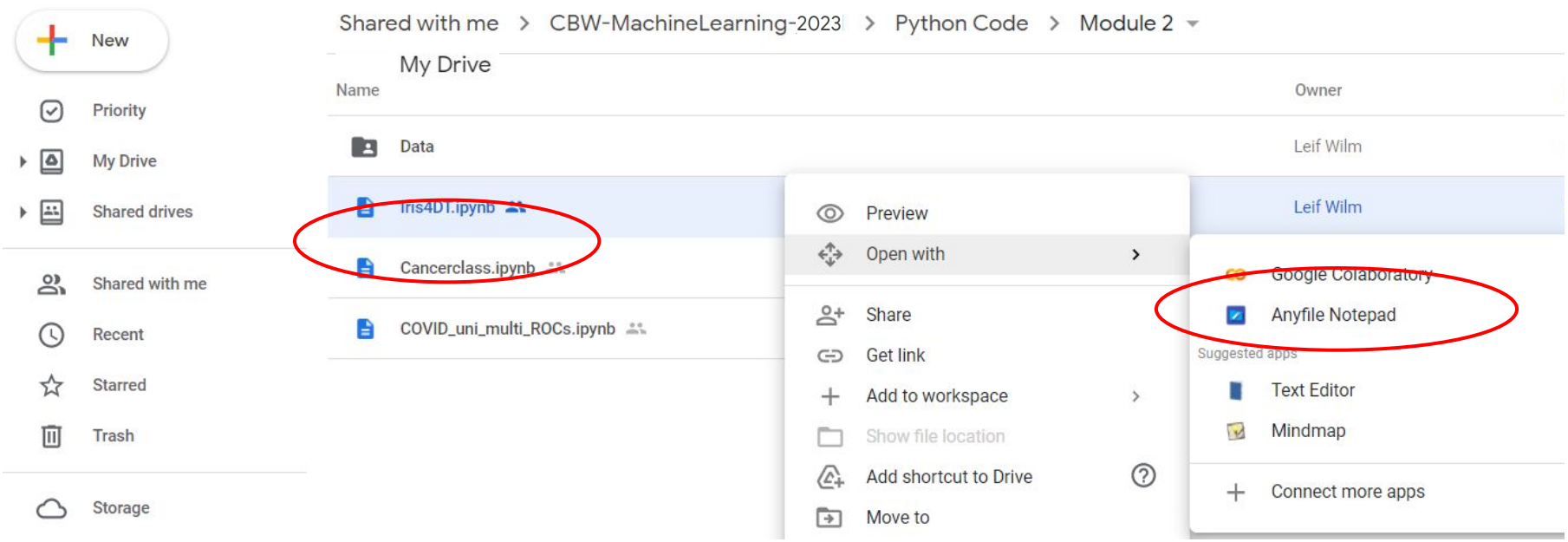
Open the Colab File in the Language of Your Choosing

- The python script 'IrisDT4.ipynb', covered during lecture, will be used for the Lab portion
- However, you have the option of running the Lab in R
- To do so, find the R code 'IrisDT4_R.ipynb' in the R Code folder

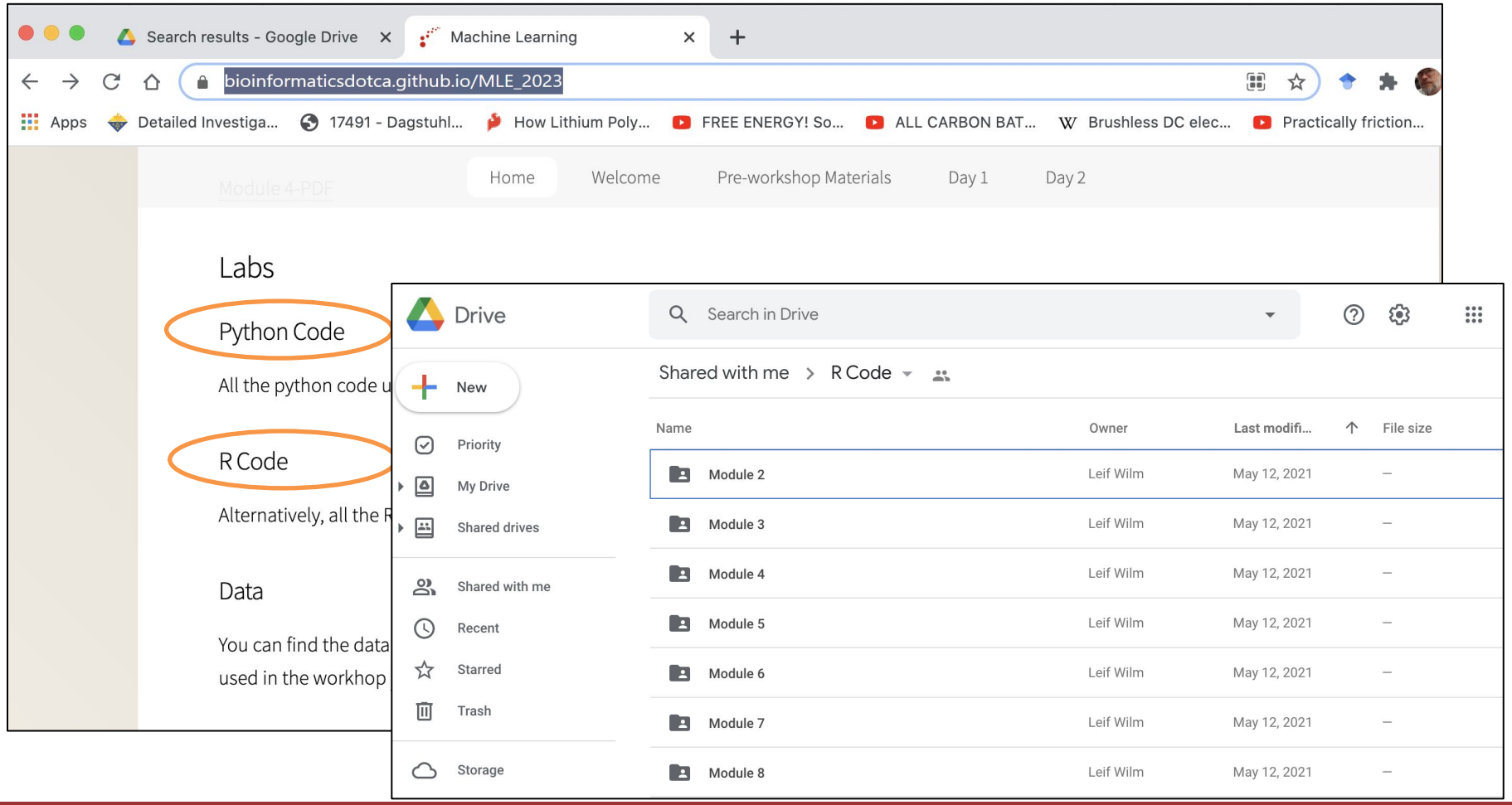


Open the Colab File in the Language of Your Choosing *cont...*

- **Right click on 'IrisDT4.ipynb' (or alternatively IrisDT4_R.ipynb in the R Code folder) and select open with Google Colaboratory**



Course Code Repository



Screenshots of R Code

Drive

Search in Drive

New

Priority

My Drive

Shared drives

Shared with me

Recent

Shared with me > R Code > Module 2

Name	Owner	Last modifi...	File size
IrisDT4_R.ipynb	Leif Wilm	7:14 PM	7 KB

colab.research.google.com/drive/1r9Fq9eNBI8W__CUw7ekn-1HwcbvnTn_U

Apps | Detailed Investiga... | 17491 - Dagstuhl... | How Lithium Poly... | FREE ENERGY! So... | ALL CARBON BAT... | Brushless DC elec...

IrisDT4_R.ipynb

File Edit View Insert Runtime Tools Help Changes will not be saved

+ Code + Text Copy to Drive

```
install.packages("rpart")
library("rpart") # This R package yileds a decision tree with high accuracy
```

```
[ ] data <- read.csv(file="data1.csv", header=TRUE)
    data[c(1:3,52:53,102:103),] #Print a few rows of data for each class/label
```

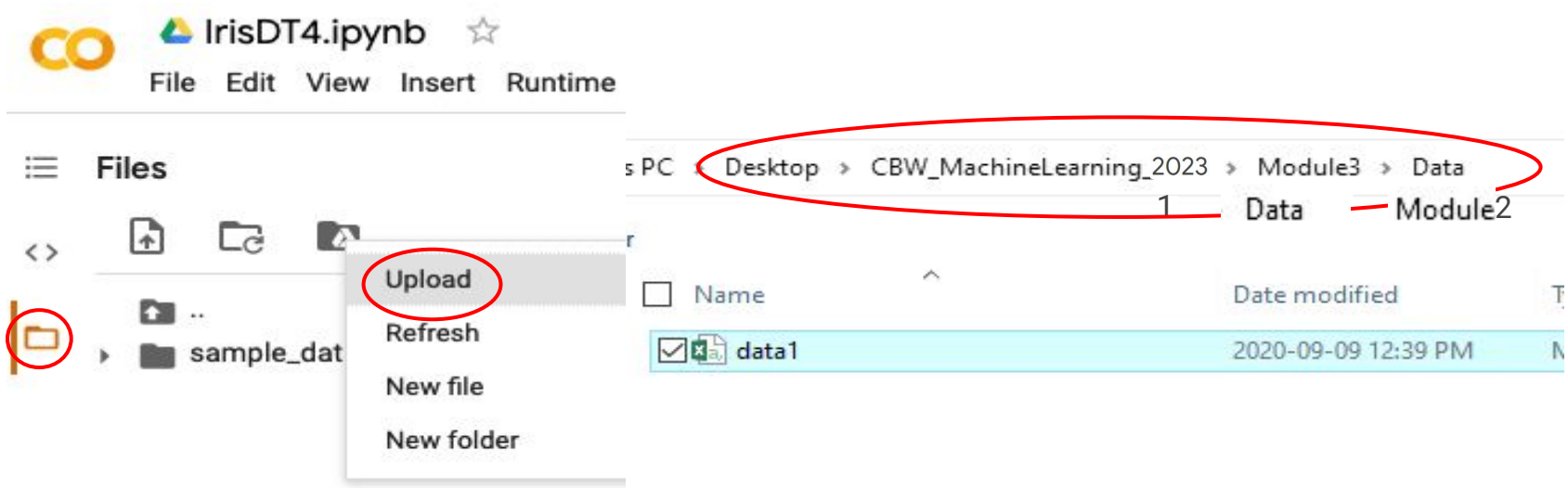
```
[ ] #Check the dataset to make sure no data is missing
    verify_dataset <- function(dataset) {
      data_found <- 1 #Flag to signal no missing data points were found in a column
      for (each_column in colnames(dataset)) { #Test all columns

        if (any(is.na(dataset[,each_column]))=="TRUE") { #Check for any missing data in column
          cat(paste0("Data missing in Column ", each_column)) #line prints which column is missing data
          data_found <- 0 #Alternative tag
          break() #Terminate loop and ultimately function since the flag data_found is 0
        }
      }
```

Study the Program

- Spend a few minutes studying the IrisDT4.ipynb (or IrisDT4_R.ipynb) program
- See if you can understand the logic of the program
- Ask questions of the TA's if you are unclear about some of the functions or Python code
- After studying the code, try running it

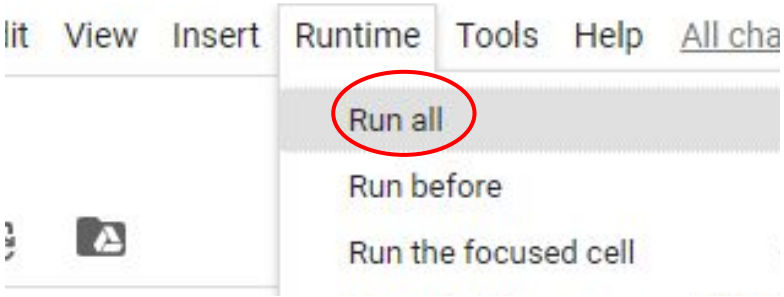
To Run: Upload the Data

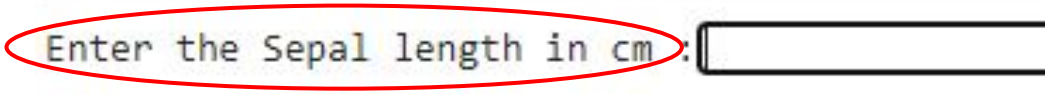
- Once the Colab file is open, click the folder icon to the right
- When the panel opens, right click inside the panel and click upload
- Then open the CBW_MachineLearning_2023 folder on your desktop
- Open the Data folder followed by the Module 2 folder, and open the corresponding data file named 'data1.csv'



Exercise 2.1

- Select “Run all” from the Runtime menu
- Go to the last cell and enter some values. Hit enter to type the next value
- SL=5.2, SW=3.2, PL=1.6, PW=0.3 (should be setosa)
- SL=6.6, SW=3.1, PL=4.6, PW=1.3 (should be versicolor)
- SL=6.3, SW=3.0, PL=5.5, PW=2.2 (should be virginica)
- SL=8.3, SW=2.0, PL=1.5, PW=1.2 (should be ???)
- SL=1.3, SW=1.0, PL=4.6, PW=0.2 (should be ???)





Exercise 2.2 - Lab

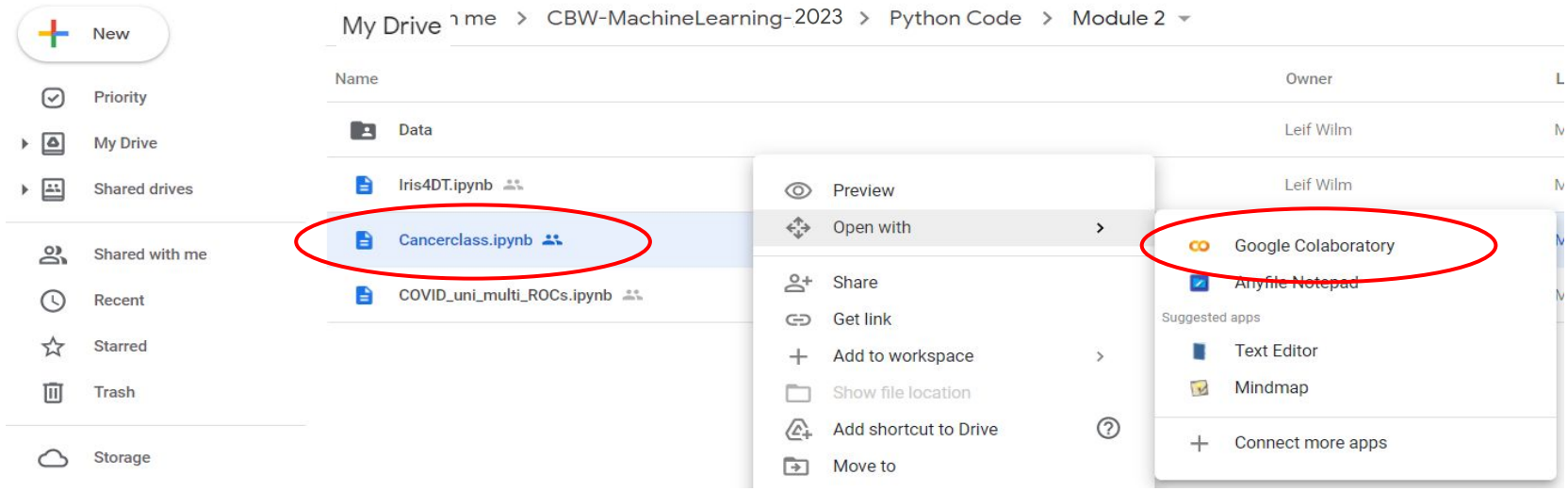
Let’s Take a Look at the CancerClass.ipynb Program

Exercise 2.2

- Here we will look at breast cancer gene expression data and 5-year mortality
- The data (see next slide) is formatted just like the iris data
- Modify the 'IrisDT4.ipynb' code (or if you wish to work in R, use 'IrisDT4_R.ipynb') so that you can train on the 'cancerdata1.csv' training data
- Note there are 3 conditions (cured, recurrence, deceased) and 4 genes (ESR1, PGR, BCL2, NAT1)
- What is your performance on the test data?

Open the Colab File

- You can compare your adjusted python code with ‘Cancerclass.ipynb’
- Right click on ‘Cancerclass.ipynb’ and select open with Google Colaboratory
- Alternatively, you can compare your adjusted R code with ‘Cancerclass_R.ipynb’ under Module 2 in the ‘R Code’ folder

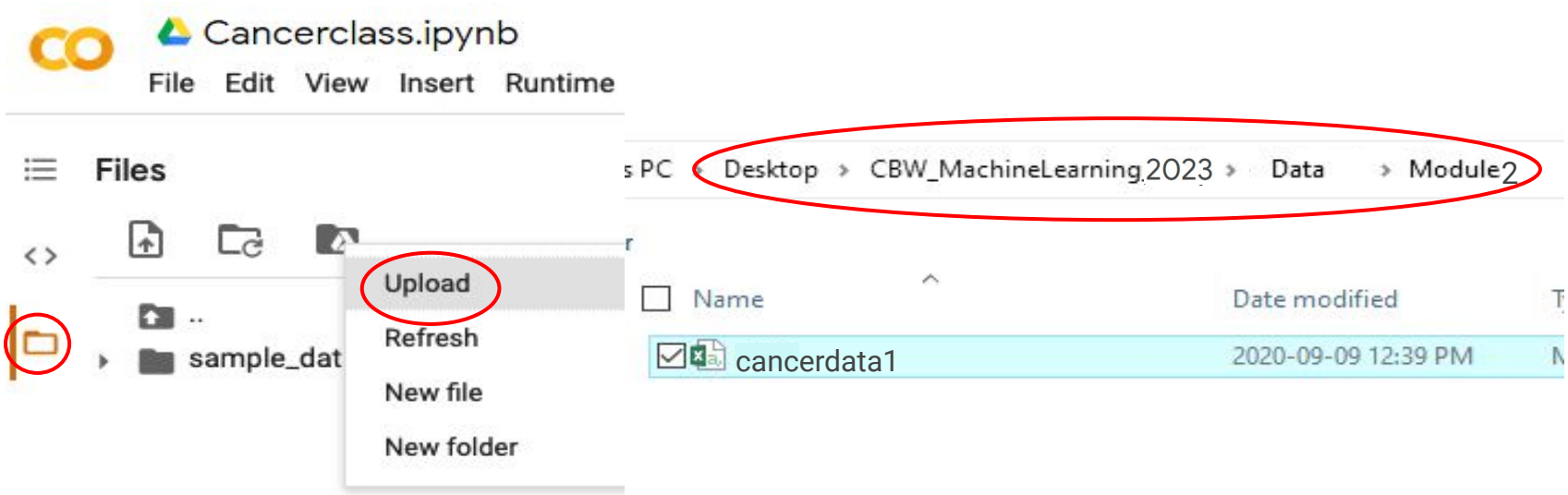


Study the Program

- Spend a few minutes studying the cancerclass.ipynb (or cancerclass_R.ipynb) program
- See if you can understand the logic of the program
- Ask questions of the TA's if you are unclear about some of the functions or Python code
- After studying the code, try running it

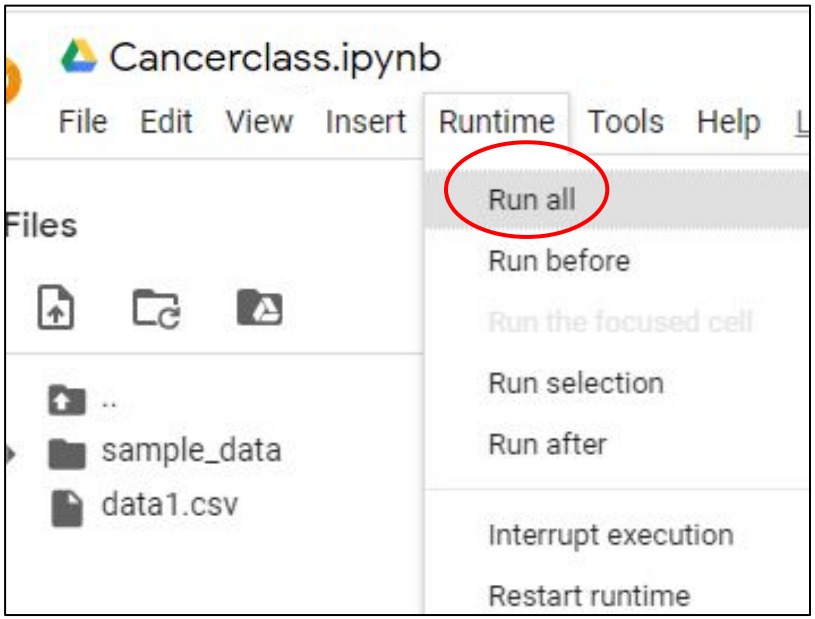
To Run: Upload the Data

- Once the Colab file is open, click the folder icon to the right
- When the panel opens, right click inside the panel and click upload
- Then open the CBW_MachineLearning_2023 folder on your desktop
- Open the Data folder followed by the Module 2 folder, and open the corresponding data file named 'cancerdata1.csv'



Run The Program

- **Select 'Run all' from the 'Runtime' menu**



Conclusion

- This concludes the “formal” section on decision trees for iris classification
- Feel free to explore the code, modify it if you wish or to review the slides and explanations – talk to the TAs

We are on a Coffee Break & Networking Session

Workshop Sponsors:

Canadian Centre for Computational Genomics

GenomeCanada