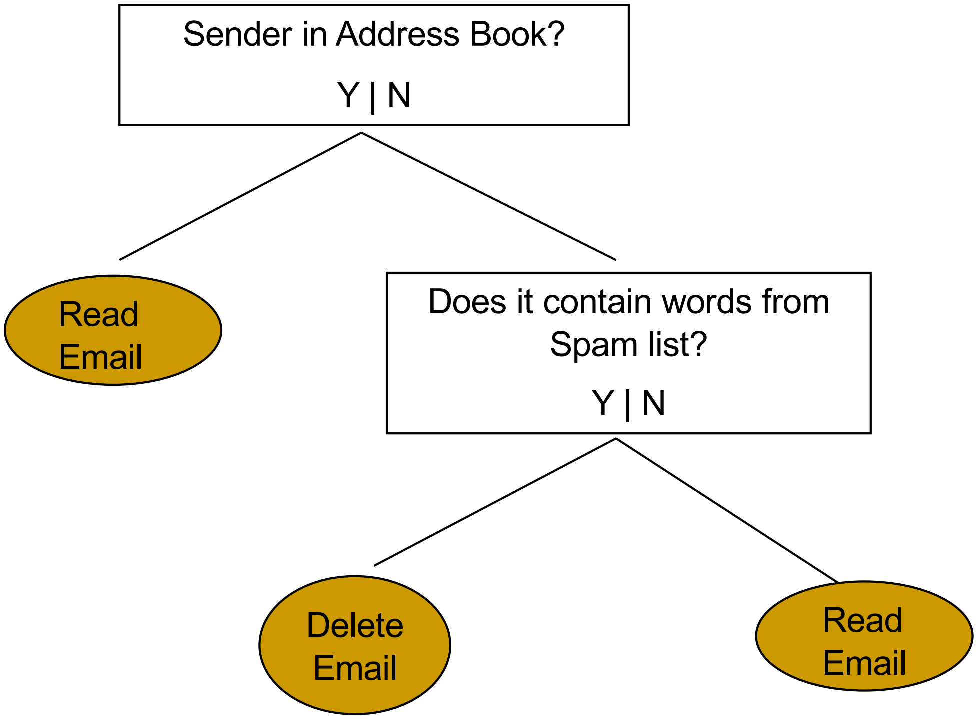
Decision Tree Classifiers

Ishwar K Sethi

Decision Tree Classifiers (DCT)

- The classifier has a tree structure where each node is either:
  - A leaf node with a class label, or
  - An internal node indicating some test to be carried out on the example passing through it
- Most tree classifiers use a single attribute-based test at internal nodes
- These classifiers are intuitive and easy to understand

A Simple DTC Example

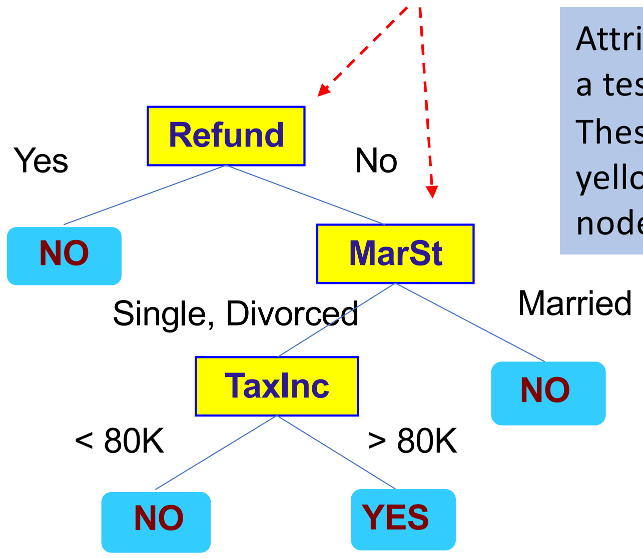


Another Example of a Decision Tree

	categorical	categorical	continuous	class
Tid	**Refund**	**Marital Status**	**Taxable Income**	**Cheat**
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**

Training Data

Splitting Attributes



Attributes involved in a test split the data. These are shown in yellow. Leaf/terminal nodes in blue.

Building Decision Trees aka DT Induction

- A two-phase procedure consisting of growing and pruning
- The growth phase searches for successive splits to divide training examples into smaller subsets of increasing purity or homogeneity in a top-down greedy manner
- Splitting is generally done by finding an attribute-value pair that maximizes some purity criterion

Decision Tree Induction

- Many Algorithms:
  - AMIG (Average Mutual Information Gain)
  - CART (Classification and Regression Trees)
  - ID3, C4.5
  - Perceptron Tree

Tree Induction

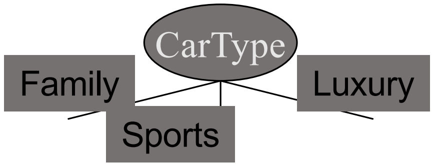
- Issues in tree growing
  - Determine how to split the records
    - How to specify the attribute test condition?
    - How to determine the best split?
  - Determine when to stop splitting

How to Specify A Test (Splitting) Condition?

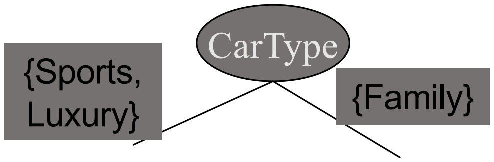
- Depends on attribute types
  - Nominal
  - Ordinal
  - Continuous
- Depends on number of ways to split
  - 2-way split
  - Multi-way split

Splitting Based on Nominal Attributes

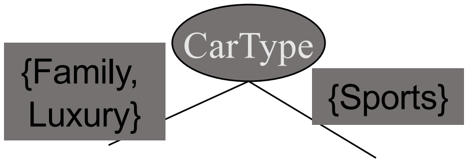
- Multi-way split: Use as many partitions as distinct values.



- Binary split: Divides values into two subsets. Need to find optimal partitioning.

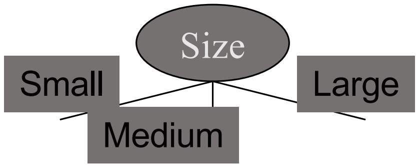


OR

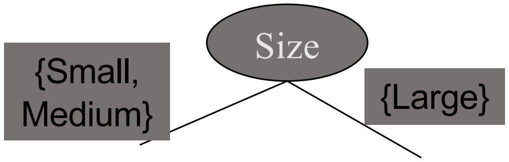


Splitting Based on Ordinal Attributes

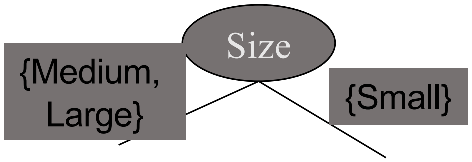
- Multi-way split: Use as many partitions as distinct values.



- Binary split: Divides values into two subsets. Need to find optimal partitioning.



OR

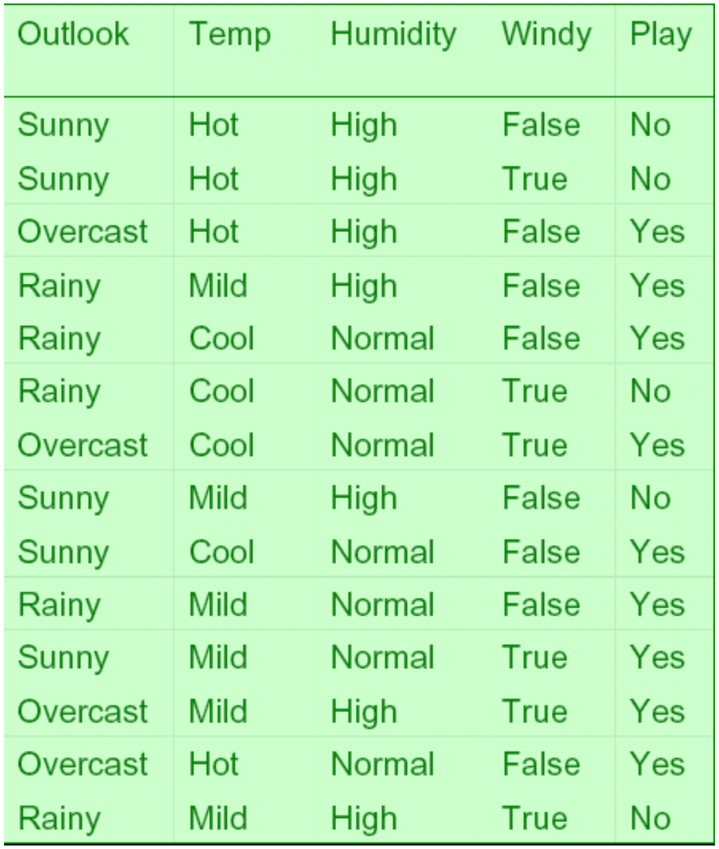


Splitting Based on Continuous Attributes

- Different ways of handling
  - Discretization to form an ordinal categorical attribute
    - Static – discretize once at the beginning
    - Dynamic – ranges can be found by equal interval bucketing, equal frequency bucketing (percentiles), or clustering.
  - Binary Decision: $(A < v)$ or $(A \geq v)$
    - consider all possible splits and finds the best cut

A Simple Example of Tree Building

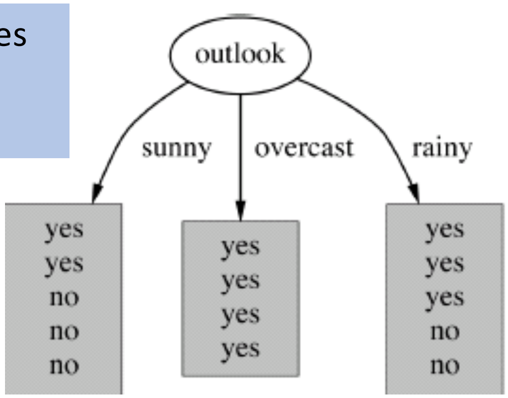
Training data consisting of four features, two classes

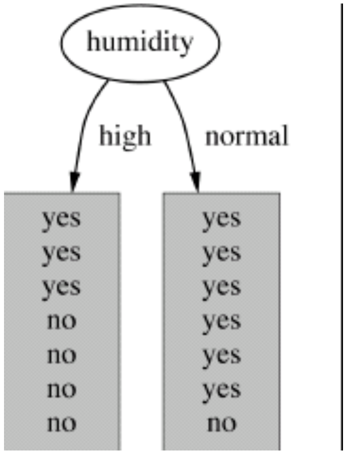


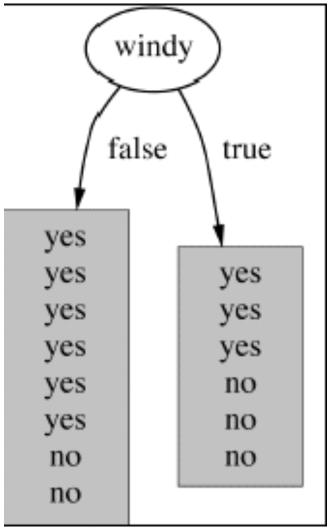
Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

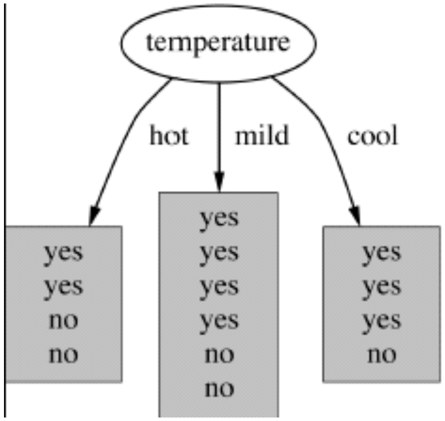
Which attribute to select?

Four possibilities at the starting node









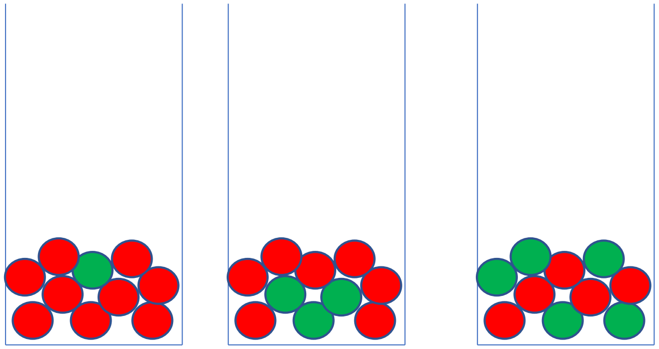
Measuring Split Quality

- Select an attribute that is likely to result in a smaller tree
- Small tree is likely to result from splits that are “purer”
- Use a purity measure

A Measure of Purity: Information Gain

- Let’s look at *entropy* first. Entropy is a measure of uncertainty. Let’s look at three jars with balls. Think of your action as blindly pulling out a red ball from each of the jar. For the left most jar, the uncertainty about the pulled ball being red is very little while it is much more for the right most jar. The entropy measure captures this notion of uncertainty. This is defined as shown below. The numbers $p_{1,}$ $p_{2,\ldots,}$ p_n represent the probabilities of different outcomes or the states of the system.

$$\text{entropy}(p_{1,}p_{2,}\ldots,p_n) = -p_1\log p_1 - p_2\log p_2 \ldots - p_n\log p_n$$



In our example, we have two states/outcomes for each jar. Plugging in the numbers, we get

Entropy of the leftmost jar = -0.1log0.1 – 0.9log0.9 => 0.14118174

Entropy of the center jar = -0.3log0.3 – 0.7log0.7 => 0.265295

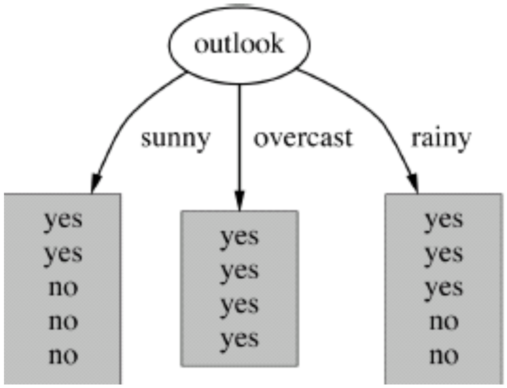
Entropy of the rightmost jar = -0.5log0.5 – 0.5log0.5 => 0.30103

We see that the action of blindly pulling a red ball has most uncertainty for the rightmost jar. Also, a low value of entropy means more purity. The above entropy calculations were done using base 10. When using base 2, the entropy is measured in bits.

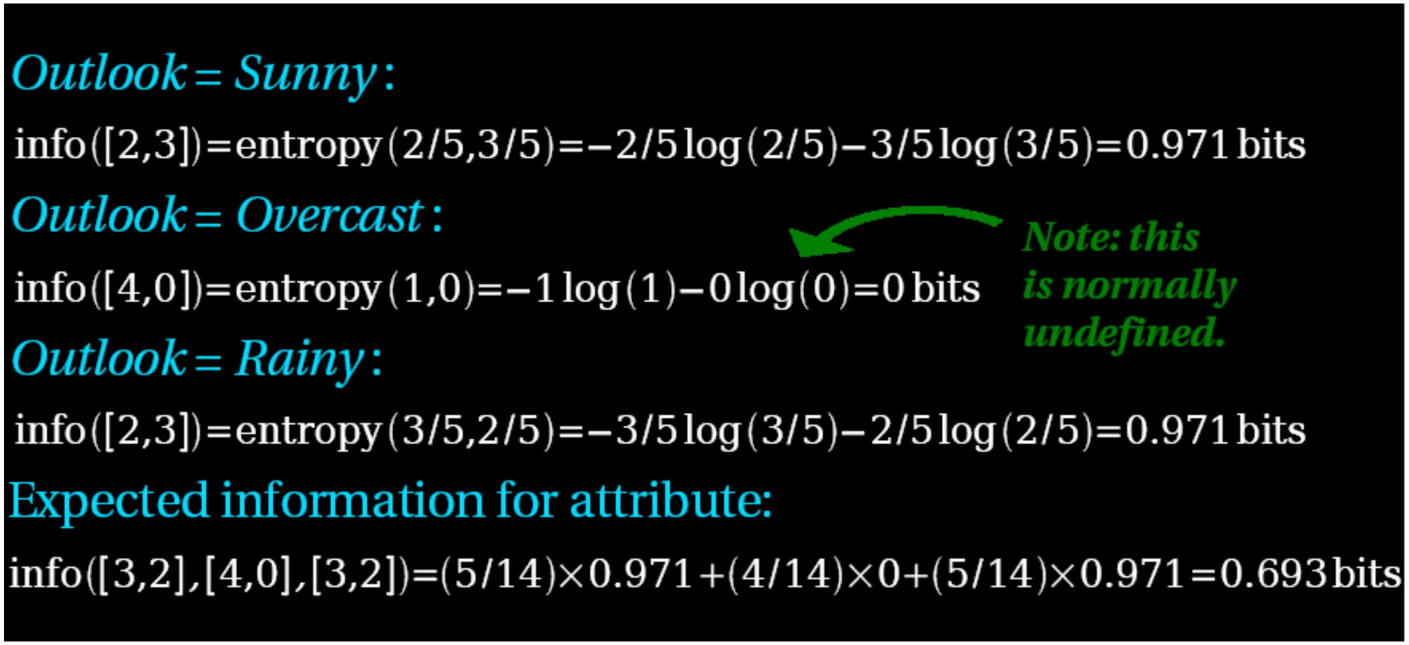
Information Gain

- Going back to the example of the previous slide, what we calculated was the entropy before the intended action.
- Once, a ball has been pulled out from a jar, the probabilities of outcomes change. This means the entropy associated with each jar changes.
- Information gain is then nothing but the difference between two entropies. It tells us how the uncertainty/purity is going to change as a result of a planned action.

Calculating Information Gain for the Play/No-Play Example



Entropy of the attribute “Outlook”



$Outlook = Sunny:$

$\text{info}([2,3]) = \text{entropy}(2/5,3/5) = -2/5\log(2/5) - 3/5\log(3/5) = 0.971\,\text{bits}$

$Outlook = Overcast:$

$\text{info}([4,0]) = \text{entropy}(1,0) = -1\log(1) - 0\log(0) = 0\,\text{bits}$

Note: this is normally undefined.

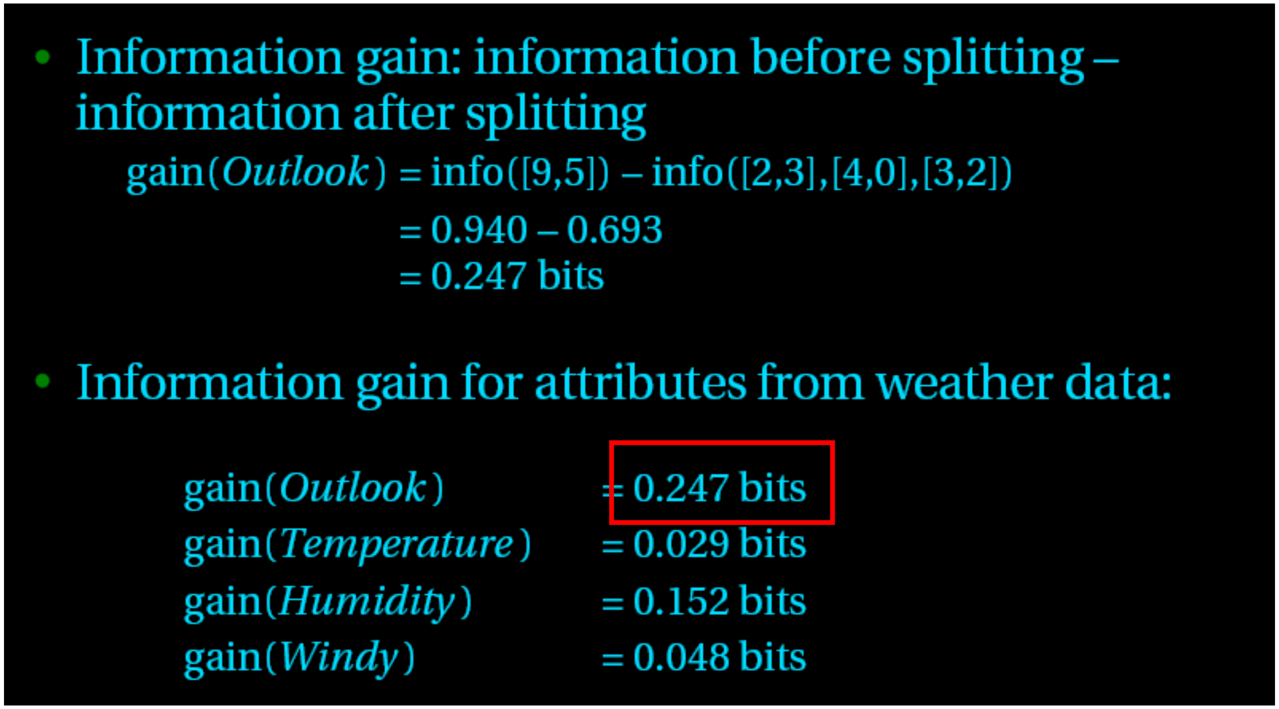
$Outlook = Rainy:$

$\text{info}([2,3]) = \text{entropy}(3/5,2/5) = -3/5\log(3/5) - 2/5\log(2/5) = 0.971\,\text{bits}$

Expected information for attribute:

$\text{info}([3,2],[4,0],[3,2]) = (5/14)\times 0.971 + (4/14)\times 0 + (5/14)\times 0.971 = 0.693\,\text{bits}$

Information Gains Based on Different Splits



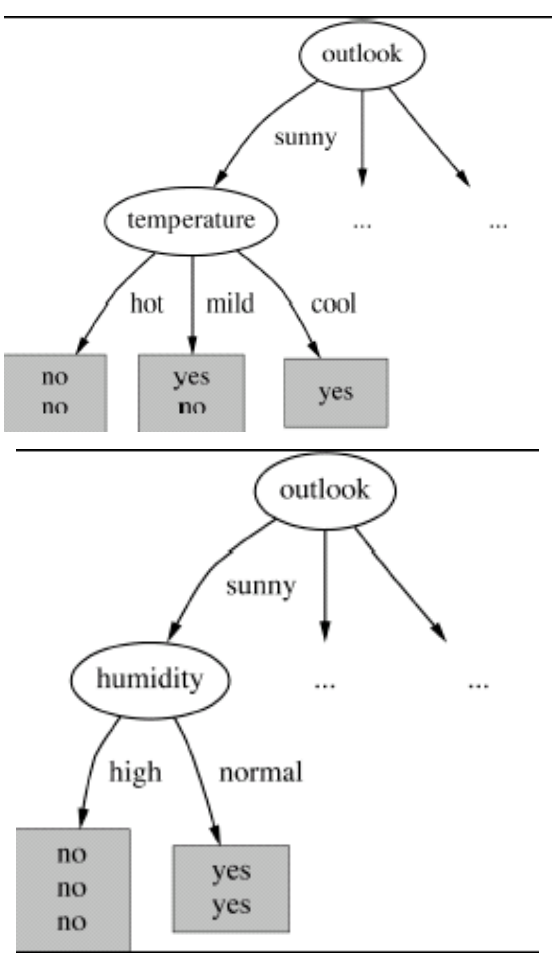
- Information gain: information before splitting – information after splitting

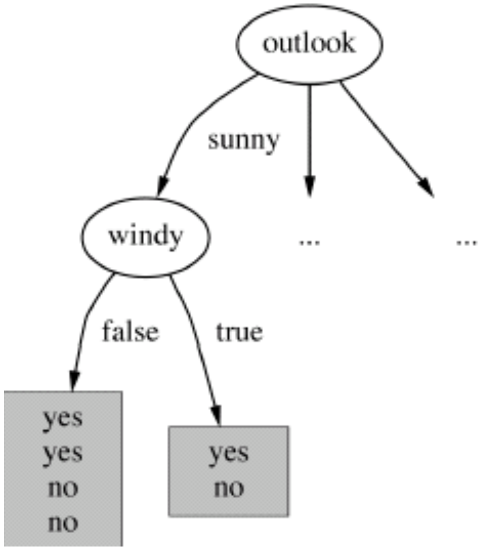
$$\begin{aligned}\text{gain}(\textit{Outlook}) &= \text{info}([9,5]) - \text{info}([2,3],[4,0],[3,2]) \\ &= 0.940 - 0.693 \\ &= 0.247 \text{ bits}\end{aligned}$$

- Information gain for attributes from weather data:

$$\begin{aligned}\text{gain}(\textit{Outlook}) \quad & \boxed{= 0.247 \text{ bits}} \\ \text{gain}(\textit{Temperature}) \quad & = 0.029 \text{ bits} \\ \text{gain}(\textit{Humidity}) \quad & = 0.152 \text{ bits} \\ \text{gain}(\textit{Windy}) \quad & = 0.048 \text{ bits}\end{aligned}$$

We see that the attribute “Outlook” provides the most gain. Thus, it is the preferred attribute to split the data.



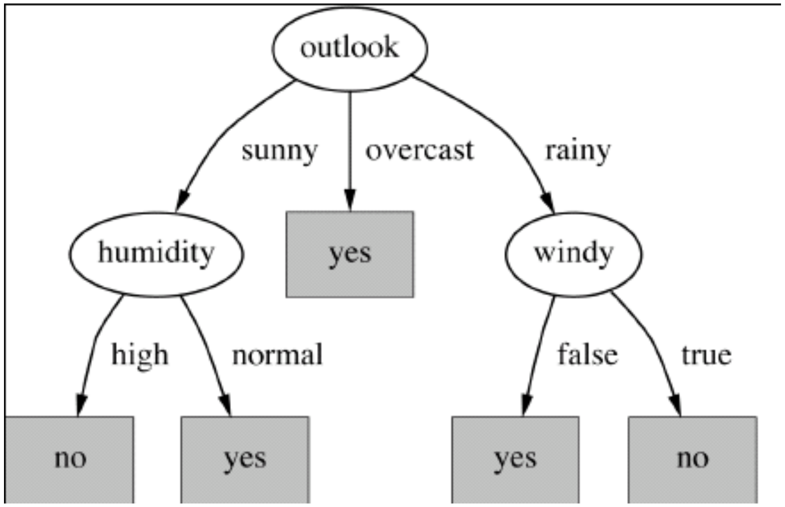


Now let’s look at three attributes for splitting at the leftmost descendent node after using “Outlook” at the previous node.

$$
\begin{aligned}
\text{gain}(\mathit{Temperature}) &= 0.571 \text{ bits}\\
\text{gain}(\mathit{Humidity}) &= 0.971 \text{ bits}\\
\text{gain}(\mathit{Windy}) &= 0.020 \text{ bits}
\end{aligned}
$$

Continuing with similar calculations at the other two descendent nodes, we arrive at the final decision tree shown in the next slide.

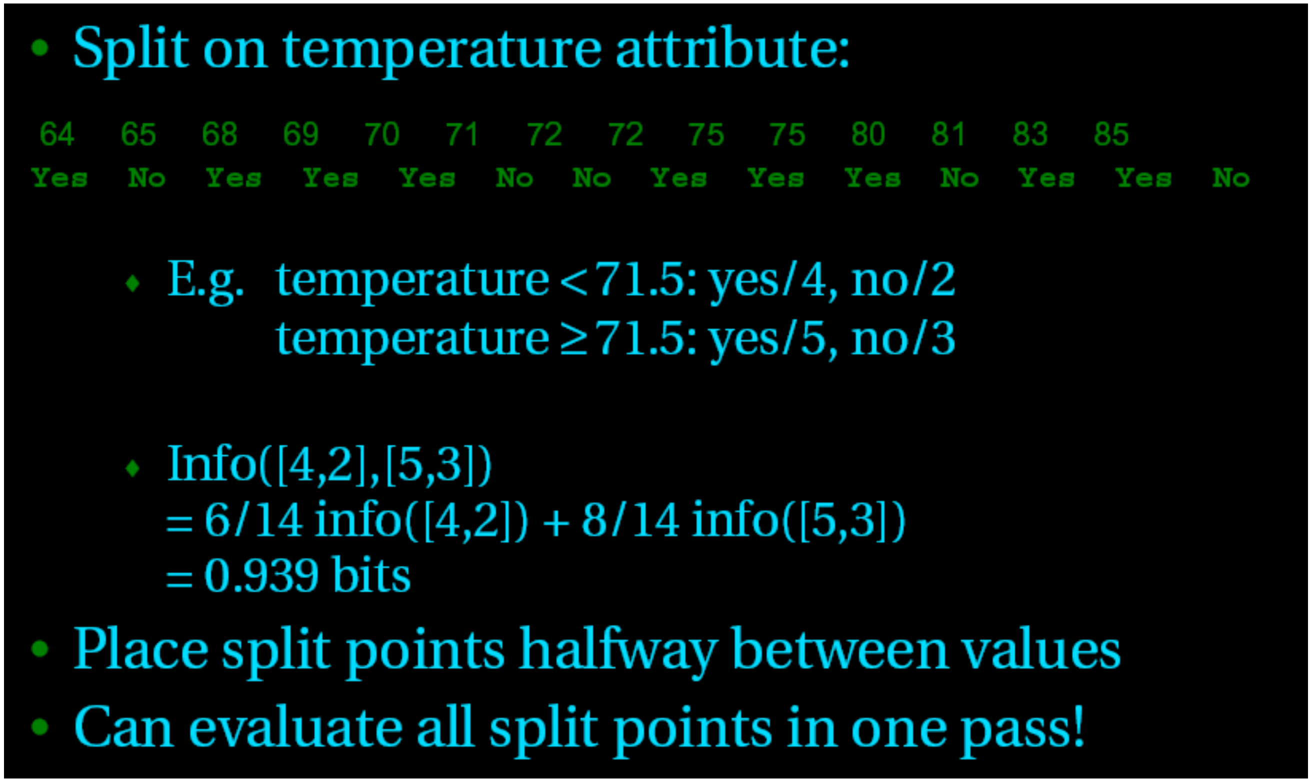
Final Decision Tree



Note: not all leaves need to be pure; sometimes identical instances have different classes

⇒ Splitting stops when data can't be split any further

Finding Splits on Numerical Attributes



- Split on temperature attribute:

64	65	68	69	70	71	72	72	75	75	80	81	83	85
Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	Yes	Yes	No

  - E.g. temperature < 71.5: yes/4, no/2
    temperature ≥ 71.5: yes/5, no/3
  - Info([4,2],[5,3])
    = 6/14 info([4,2]) + 8/14 info([5,3])
    = 0.939 bits
- Place split points halfway between values
- Can evaluate all split points in one pass!

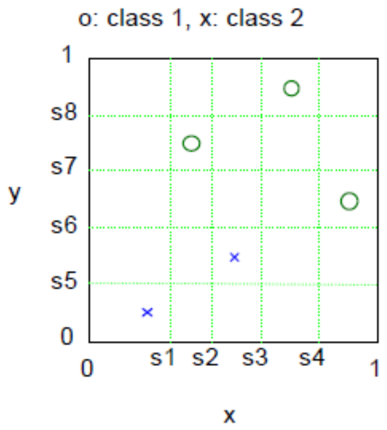
When working with continuous attributes, we sort the attribute values and create an array of associated labels.

Entropy Measure for Splits' Quality: An Example with Continuous Attributes

$$E = -\sum_{j=1}^{M} p_j \ln p_j$$

$$\Delta E(s) = E - p_{left} E_{left} - p_{right} E_{right}$$

$$E = -\frac{2}{5}\ln(\frac{2}{5}) - \frac{3}{5}\ln(\frac{3}{5}) = 0.6730$$



To see which split should be preferred, let us calculate the change in impurity due to each split. We show these calculations for split s_2.

$$p_{left} = \frac{2}{5},$$

$$E_{left} = -\frac{1}{2}\ln(\frac{1}{2}) - \frac{1}{2}\ln(\frac{1}{2}) = 0.6983,$$

$$p_{right} = \frac{3}{5},$$

$$E_{right} = -\frac{1}{3}\ln(\frac{1}{3}) - \frac{2}{3}\ln(\frac{2}{3}) = 0.6365$$

$$\Delta E(s_1) = 0.2231,$$
$$\Delta E(s_3) = 0.2911,$$
$$\Delta E(s_4) = 0.1185,$$
$$\Delta E(s_5) = 0.2231,$$
$$\Delta E(s_6) = 0.6730,$$
$$\Delta E(s_7) = 0.2911,$$
$$\Delta E(s_8) = 0.1185.$$

Best split: S_6

Therefore, the change in impurity due to split s_2 is given as

$$\Delta E(s_2) = 0.6730 - 0.4 * 0.6983 - 0.6 * 0.6365$$
$$= 0.0138$$

GINI Measure of Impurity

- Another popular measure is Gini Index for a given node t :

$$GINI(t) = 1 - \sum_{j} [p(j \mid t)]^2$$

(NOTE: $p(j \mid t)$ is the relative frequency of class j at node t).

  - Maximum ($1 - 1/n_c$) when records are equally distributed among all classes, implying least interesting information
  - Minimum (0.0) when all records belong to one class, implying most interesting information

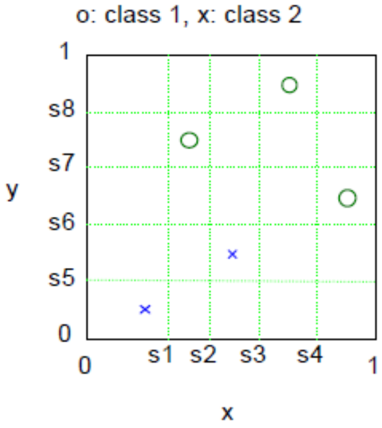
The Gini index, or Gini coefficient, is a measure of the distribution of income across a population developed by the Italian statistician Corrado Gini in 1912. It is often used as a gauge of economic inequality, measuring income distribution or, less commonly, wealth distribution among a population.

GINI Measure of Impurity

$$G = 1 - \sum_{j=1}^{M} p_j^2.$$

$$\Delta G(s) = G - p_{left} G_{left} - p_{right} G_{right}$$

$$G = 1 - (\frac{2}{5})^2 - (\frac{3}{5})^2 = 0.480.$$



$$\Delta G(s_1) = 0.180,$$
$$\Delta G(s_2) = 0.427,$$
$$\Delta G(s_3) = 0.214,$$
$$\Delta G(s_4) = 0.080,$$
$$\Delta G(s_5) = 0.180,$$
$$\Delta G(s_6) = 0.480,$$
$$\Delta G(s_7) = 0.214,$$
$$\Delta G(s_8) = 0.080.$$

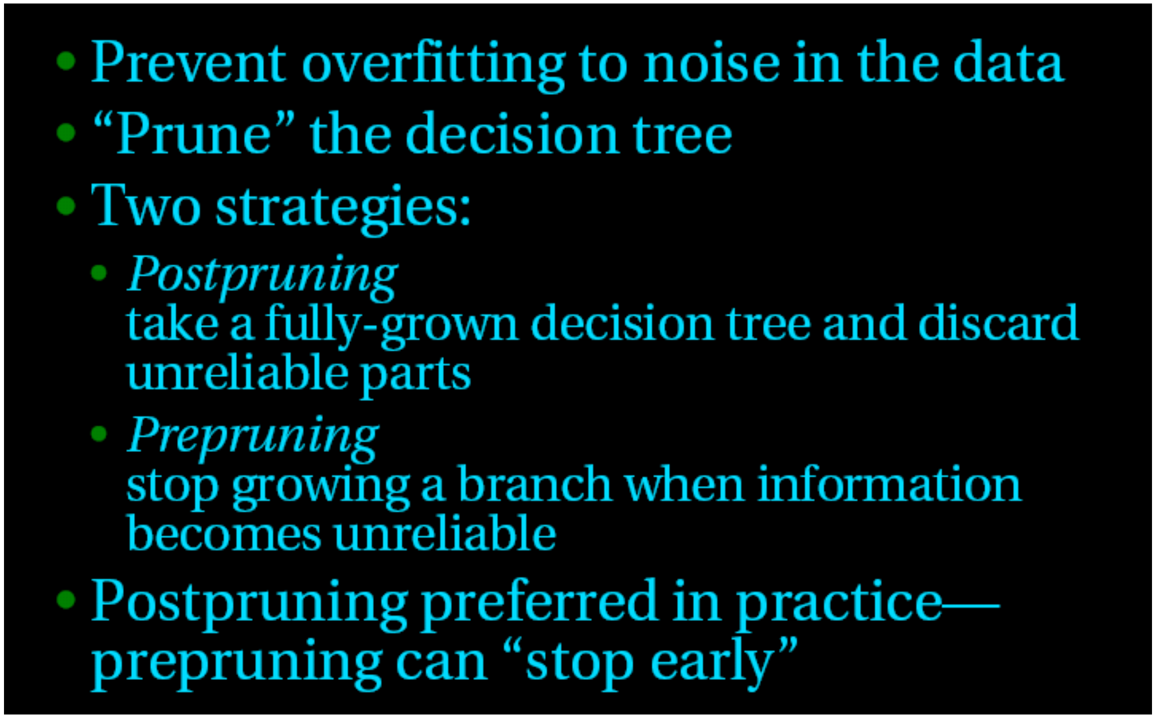
Not surprisingly, the best split is s_6.

Stopping Criteria for Tree Induction

- Stop expanding a node when all the records belong to the same class

- Stop expanding a node when all the records have similar attribute values

Pruning for Right Size Tree

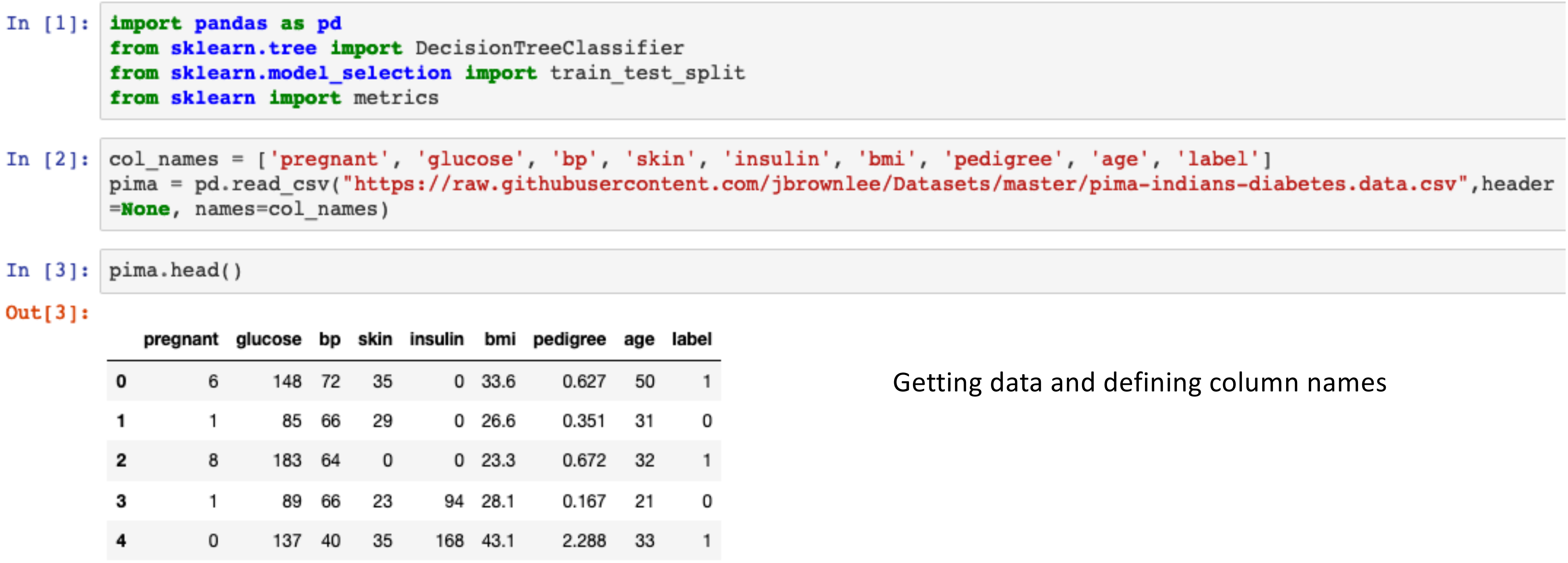


- Prevent overfitting to noise in the data
- "Prune" the decision tree
- Two strategies:
  - *Postpruning*
    take a fully-grown decision tree and discard unreliable parts
  - *Prepruning*
    stop growing a branch when information becomes unreliable
- Postpruning preferred in practice— prepruning can "stop early"

If we keep splitting and growing the tree, we are likely to end with a decision tree customized for that particular dataset that is being used to grow the tree. This means overfitting and likely poor performance in practice.

DT Classification Example

Pima Indian Diabetes dataset consisting of 8 features and 2 classes. 768 examples with 268 in Class 1



```
In [1]: import pandas as pd
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.model_selection import train_test_split
        from sklearn import metrics
```

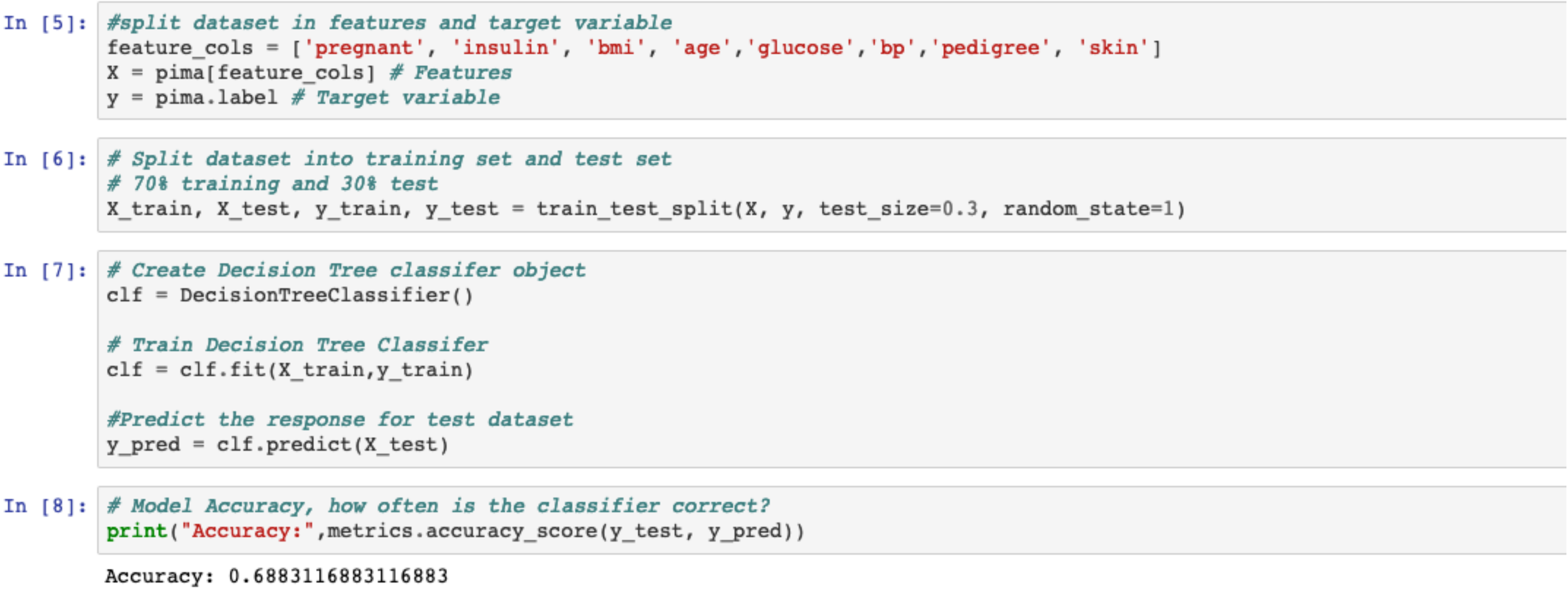
```
In [2]: col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
        pima = pd.read_csv("https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv",header
        =None, names=col_names)
```

```
In [3]: pima.head()
```

Out[3]:

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Getting data and defining column names



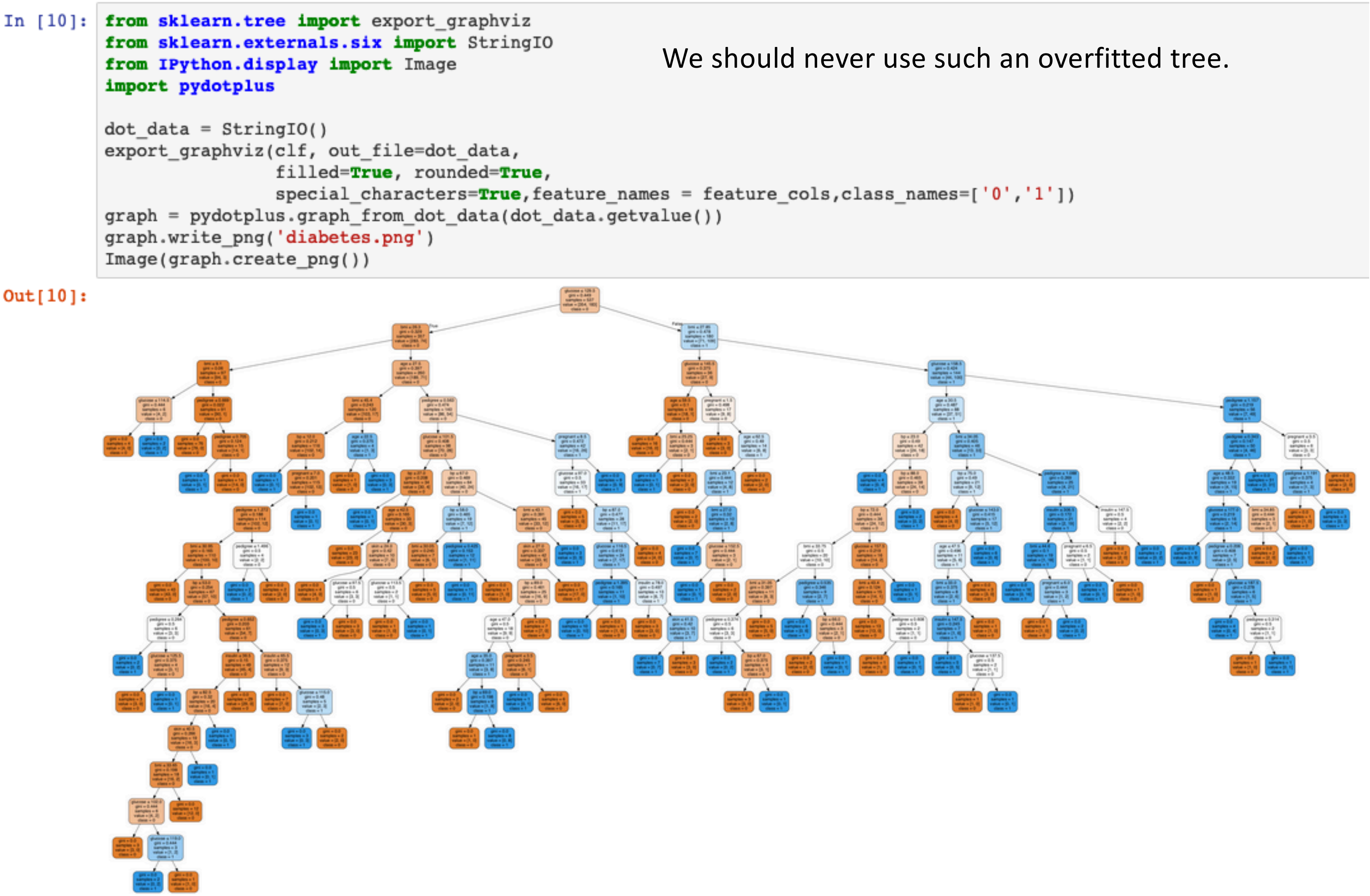
```
In [5]: #split dataset in features and target variable
        feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree', 'skin']
        X = pima[feature_cols] # Features
        y = pima.label # Target variable
```

```
In [6]: # Split dataset into training set and test set
        # 70% training and 30% test
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

```
In [7]: # Create Decision Tree classifer object
        clf = DecisionTreeClassifier()

        # Train Decision Tree Classifer
        clf = clf.fit(X_train,y_train)

        #Predict the response for test dataset
        y_pred = clf.predict(X_test)
```

```
In [8]: # Model Accuracy, how often is the classifier correct?
        print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

        Accuracy: 0.6883116883116883
```

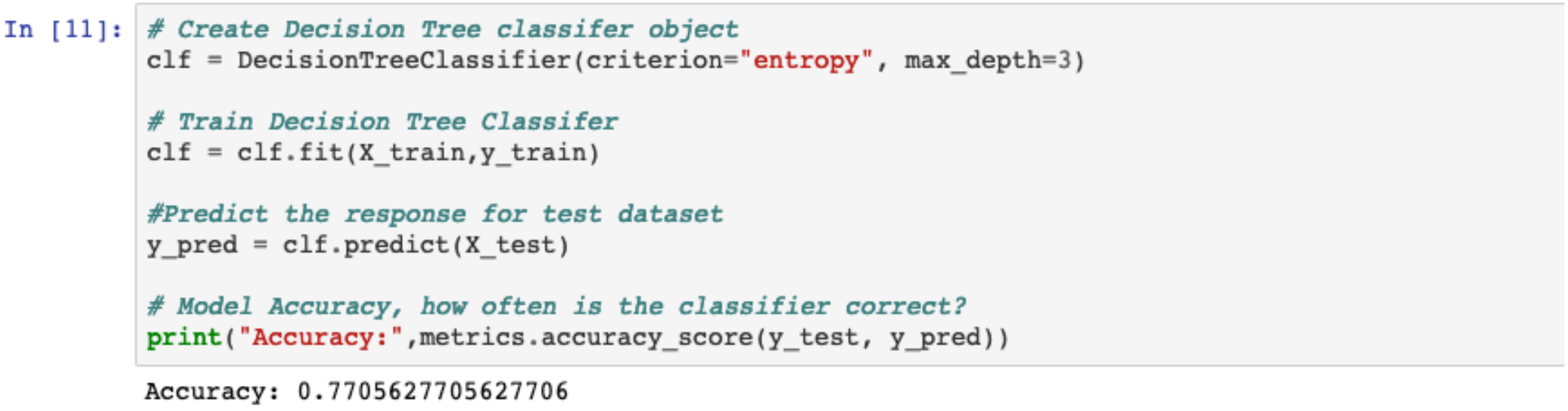
This is the accuracy of the DT built using default parameters in sklearn. This tree is fully grown (shown in the next slide) and overfitted.



```
In [10]: from sklearn.tree import export_graphviz
         from sklearn.externals.six import StringIO
         from IPython.display import Image
         import pydotplus

         dot_data = StringIO()
         export_graphviz(clf, out_file=dot_data,
                         filled=True, rounded=True,
                         special_characters=True,feature_names = feature_cols,class_names=['0','1'])
         graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
         graph.write_png('diabetes.png')
         Image(graph.create_png())

Out[10]:
```

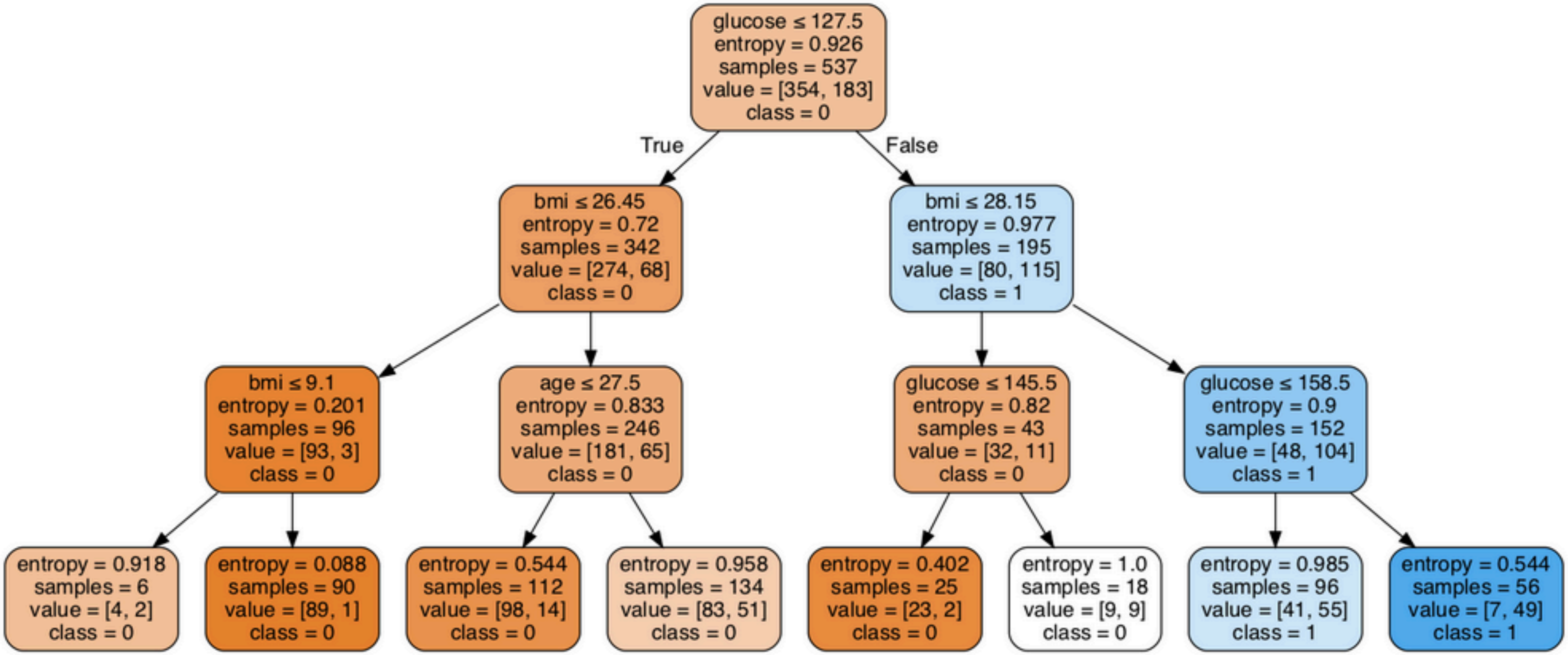
We should never use such an overfitted tree.

Growing a tree with built-in pruning by restricting the tree depth



```
In [11]: # Create Decision Tree classifer object
         clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)

         # Train Decision Tree Classifer
         clf = clf.fit(X_train,y_train)

         #Predict the response for test dataset
         y_pred = clf.predict(X_test)

         # Model Accuracy, how often is the classifier correct?
         print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

         Accuracy: 0.7705627705627706
```

This creates a smaller tree (shown in next slide) with maximum depth of 3, meaning at most 7 internal nodes. You can see this tree performs better on the test data.


glucose ≤ 127.5
entropy = 0.926
samples = 537
value = [354, 183]
class = 0
True
False
bmi ≤ 26.45
entropy = 0.72
samples = 342
value = [274, 68]
class = 0
bmi ≤ 28.15
entropy = 0.977
samples = 195
value = [80, 115]
class = 1
bmi ≤ 9.1
entropy = 0.201
samples = 96
value = [93, 3]
class = 0
age ≤ 27.5
entropy = 0.833
samples = 246
value = [181, 65]
class = 0
glucose ≤ 145.5
entropy = 0.82
samples = 43
value = [32, 11]
class = 0
glucose ≤ 158.5
entropy = 0.9
samples = 152
value = [48, 104]
class = 1
entropy = 0.918
samples = 6
value = [4, 2]
class = 0
entropy = 0.088
samples = 90
value = [89, 1]
class = 0
entropy = 0.544
samples = 112
value = [98, 14]
class = 0
entropy = 0.958
samples = 134
value = [83, 51]
class = 0
entropy = 0.402
samples = 25
value = [23, 2]
class = 0
entropy = 1.0
samples = 18
value = [9, 9]
class = 0
entropy = 0.985
samples = 96
value = [41, 55]
class = 1
entropy = 0.544
samples = 56
value = [7, 49]
class = 1

Multifeature Splits

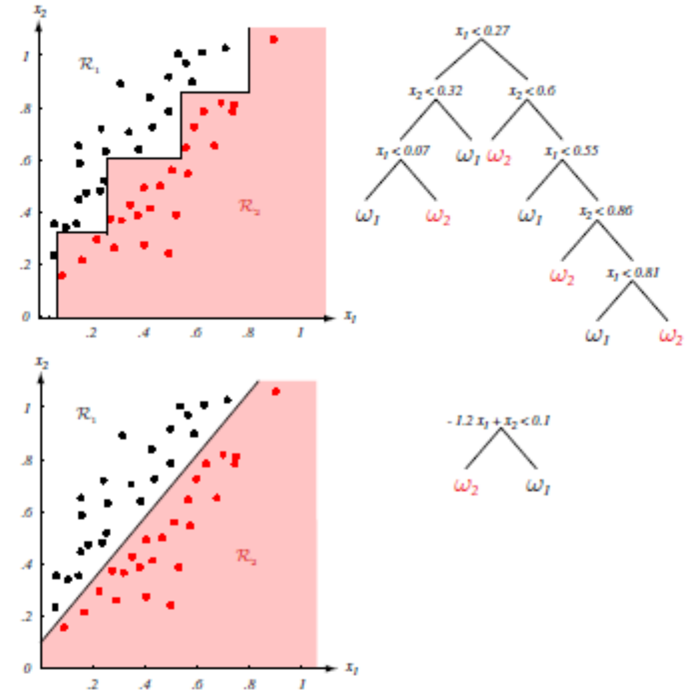


FIGURE 8.5. If the class of node decisions does not match the form of the training data, a very complicated decision tree will result, as shown at the top. Here decisions are parallel to the axes while in fact the data is better split by boundaries along another direction. If, however, "proper" decision forms are used (here, linear combinations of the features), the tree can be quite simple, as shown at the bottom. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

DTC Strengths

- Intuitive, easy to comprehend
- Fast classification
- Can deal with continuous and categorical attributes
- Built in attribute selection [All attributes may not appear in the final tree]

DTC Weaknesses

- Prone to errors due to noise in attributes
- May not yield good performance with single feature splits
- Can over-fit if not pruned properly
- May mask important relationships because of greedy top-down search while building the tree [https://iksinc.online/2013/04/06/best-split-doesnt-necessarily-produce-the-best-decision-tree/]