SciComp with Py

Decision Trees

Vladimir Kulyukin
Department of Computer Science
Utah State University

Outline

- Introduction
- Entropy
- Information Gain
- Learning Decision Trees

Introduction

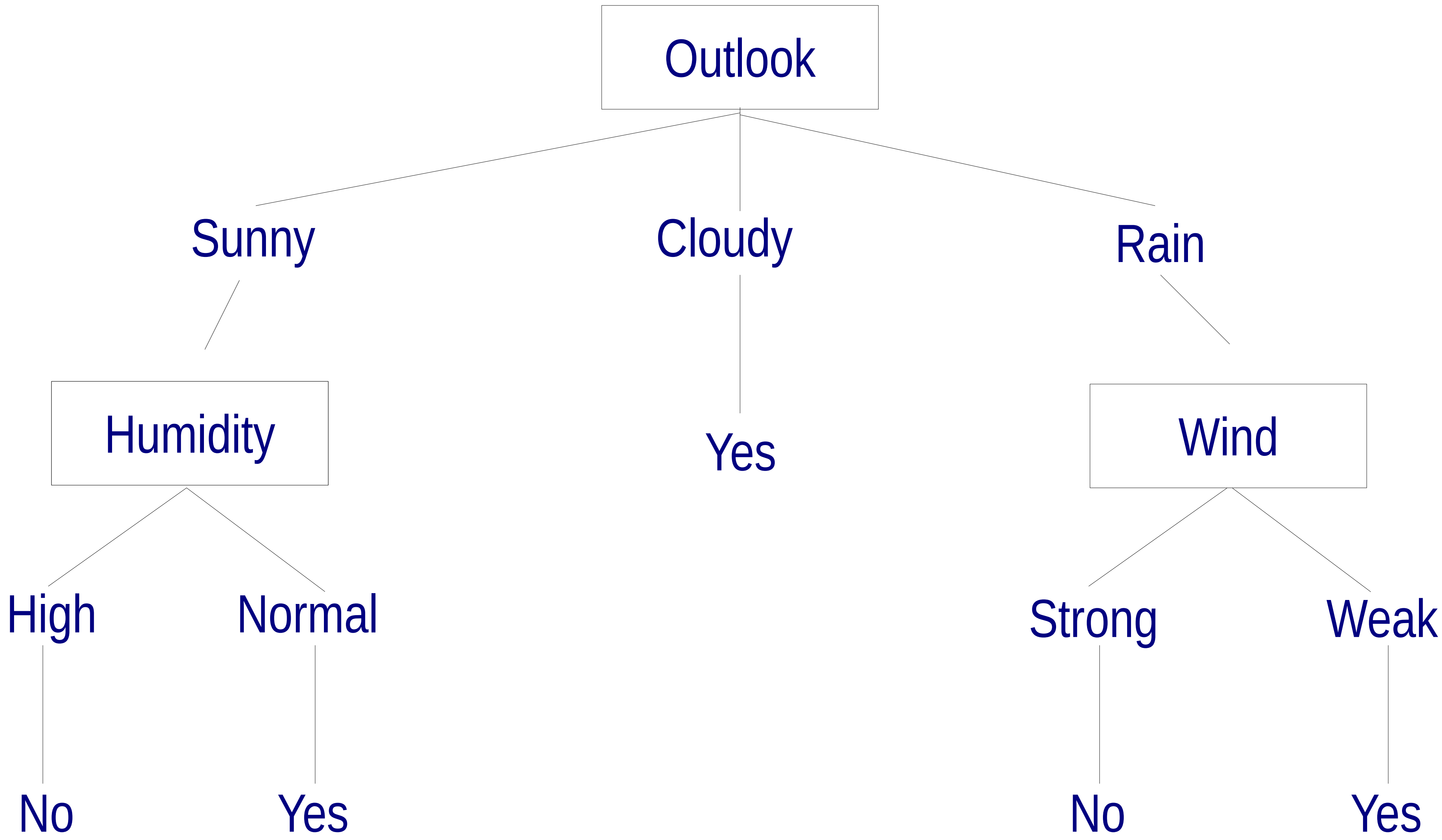
Decision Trees

- Decision tree learning is a method for classifying discrete-valued data
- Decision tree learning is appropriate when samples are represented by attribute-value pairs: each sample is described by a finite number of attributes each of which can take on a finite number of values
- Samples must be classified into a finite number of classes

Typical Decision Tree Problems

- Classify medical patients by their symptoms
- Classify equipment malfunctions by their descriptions
- Classify loan applications by their likelihood of defaulting on payments
- Classify resumes by their likelihood of being hired

Example: Decision Tree to Go On Jeep Trail



Example: Learning the Concept GoOnJeepTrail

The tree on the previous slide encodes a decision procedure for when one should go on a Jeep trail. Specifically, go on a Jeep trail:

- when outlook = sunny and humidity = normal or
- when outlook = cloudy or
- when outlook = rain and wind = weak

Core Decision Tree Learning Algorithm

All decision tree learning algorithms (e.g., ID3 or C4.5) are based on the same idea: choose the best attribute that splits the samples into subsets on the basis of the attribute's values and then recursively descend into each subset and find the best attribute for it and so on. Recursion stops when a set cannot be partitioned.

How do we choose the best attribute though? Let's find out.

Entropy

Entropy

- How does one select the best attribute to split a set of samples?
- This selection is based on entropy, a function that measures the diversity of a set
- The more diverse a set, the higher its entropy
- Put another way, the more diverse a set, the more information it contains

Entropy Formula of a Binary (2-Valued) Attribute

Let S be a set of samples classified as positive and negative with respect to some attribute (e.g., Hired). Let p_1 be the proportion of positive samples in S and p_0 be the proportion negative samples in S. Then the entropy of S relative to that attribute is computed as follows.

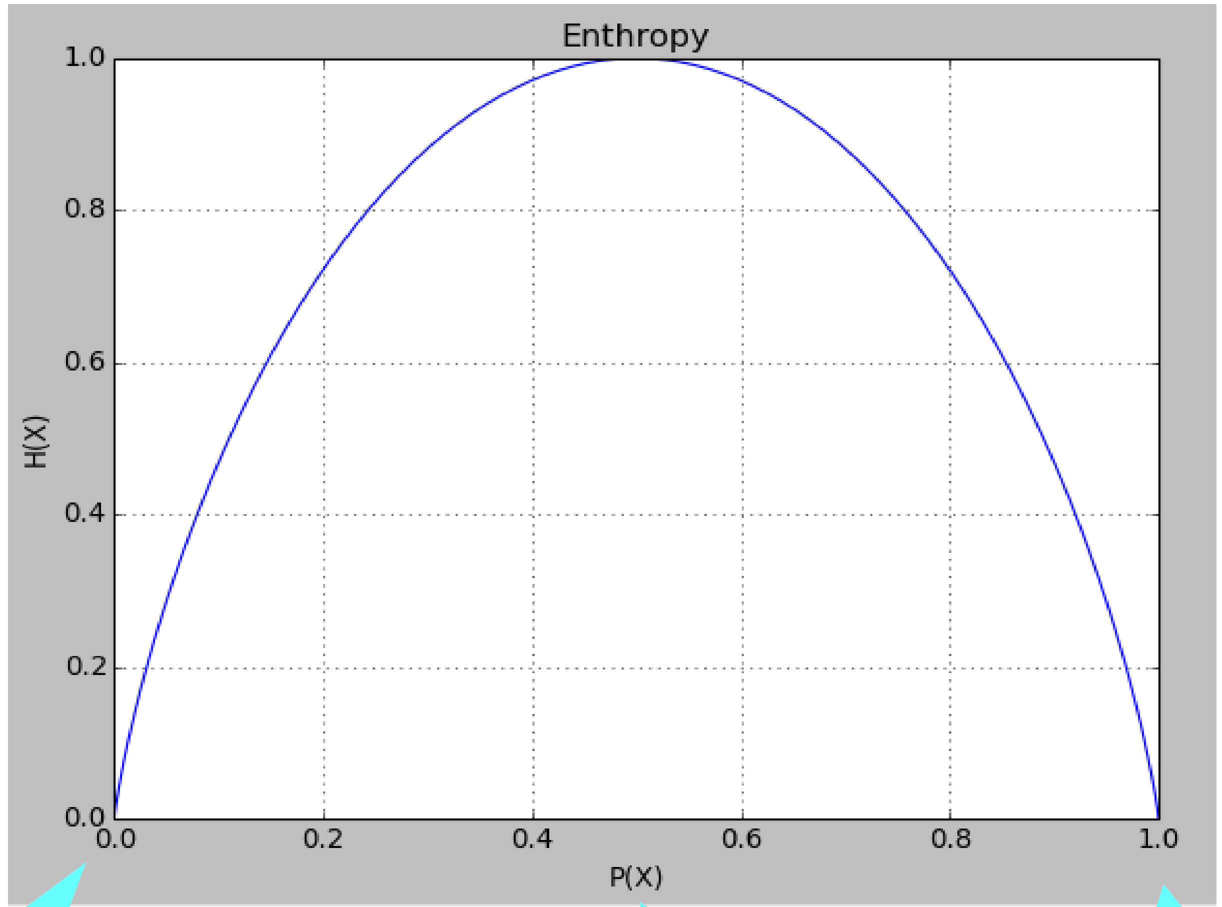
$$H(S) = -p_1 log_2(p_1) + -p_0 log_2(p_0)$$

Example

Suppose that S has 9 positive and 5 negative examples. Then its entropy is computed as follows.

$$H(S) = -\frac{9}{14} log_2(\frac{9}{14}) + -\frac{5}{14} log_2(\frac{5}{14}) = 0.94$$

Binary Entropy Plot



When probability of positive samples is 0, then there is no entropy.

When probability of positive samples is 0.5, the entropy is the highest.

When probability of positive samples is 1, then there is no entropy.

Entropy Formula of a C-Valued Attribute

If an attribute takes on *c* possible values then H(S) is the entropy of S relative to *c*-wise classification

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

Entropy in Py

```
def h(p):
   if p == 0:
        return 0
    else:
        return -p*math.log(p, 2)


def H(s):
    return sum(h(pi) for pi in s)
```

entropy.py

Information Gain

Information Gain

- The objective is to classify a set of samples into a set of classes
- Within each class, the entropy should be as small as possible
- The best attribute is the one that gives us the largest expected reduction of entropy
- This expected reduction of entropy is called information gain

Information Gain

Number of elements in S for which A=v

Set of samples

Attribute

$$Gain(S,A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_v)$$

Entropy of S

Set of all possible values of A

Entropy of S_v

Number of elements in S

Example

There are 14 samples (days) for the GoOnJeepTrail concept. Let us compute Gain(S, Wind).

Day	Outlook	Temperature	Humidity	Wind	GoOnJeepTrail
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Cloudy	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Cloudy	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Cloudy	Mild	High	Strong	Yes
D13	Cloudy	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Example

There are 14 samples (days) for the GoOnJeepTrail concept. Let us compute Gain(S, Wind).

$$Gain(S, Wind) = H(S) - \sum_{v \in \{Weak, Strong\}} \frac{|S_v|}{|S|} H(S_v) =$$
$$H(S) - \frac{8}{14} H(S_{Weak}) - \frac{6}{14} H(S_{Strong}) =$$
$$0.94 - \frac{8}{14} 0.811 - \frac{6}{14} 1.00 = 0.048$$

$$H(S) = -\frac{9}{14} log_2(\frac{9}{14}) + -\frac{5}{14} log_2(\frac{5}{14}) = 0.94$$

Humidity vs. Wind

Which attribute is better – Humidity or Wind?

$$Gain(S, Wind) = H(S) - \sum_{v \in \{Weak, Strong\}} \frac{|S_v|}{|S|} H(S_v) =$$
$$H(S) - \frac{8}{14} H(S_{Weak}) - \frac{6}{14} H(S_{Strong}) =$$
$$0.94 - \frac{8}{14} 0.811 - \frac{6}{14} 1.00 = 0.048$$

$$Gain(S, Humidity) = H(S) - \sum_{v \in \{Normal, High\}} \frac{|S_v|}{|S|} H(S_v) =$$
$$H(S) - \frac{7}{14} H(S_{Normal}) - \frac{7}{14} H(S_{High}) =$$
$$0.94 - \frac{7}{14} 0.592 - \frac{6}{14} 0.985 = 0.151$$

Humidity vs. Wind

Humidity is a better attribute, because it gives us higher information gain: 0.151 vs. 0.048.

Core Decision Tree Principle

Always choose to split the set on the attribute with the highest information gain, i.e., the attribute that results in the greatest reduction in entropy.

Learning Decision Trees

Problem

We have a database of applications and hiring decisions (Yes/No) made for each application by some company X. Each application is described in terms of a finite set of attributes and values. We need to learn a decision tree for predicting the hiring decisions of incoming applications.

Hiring Data

	Years Experience	Employed?	Previous Employers	Level of Education	Top-tier School?	Interned?	Hired?
1)	10	Y	4	BS	N	N	Y
2)	0	N	0	BS	Y	Y	Y
3)	7	N	6	BS	N	N	N
4)	2	Y	1	MS	Y	N	Y
5)	20	N	2	PhD	Y	N	N
6)	0	N	0	PhD	Y	Y	Y
7)	5	Y	2	MS	N	Y	Y
8)	3	N	1	BS	N	Y	Y
9)	15	Y	5	BS	N	N	Y
10)	0	N	0	BS	N	N	N
11)	1	N	1	PhD	Y	N	N
12)	4	Y	1	MS	N	Y	Y
13)	0	N	0	PhD	Y	N	Y

Attribute-Value Encoding

The first step is to assign numbers to attributes and values:

X[0] = 'Years of Experience'
X[1] = 'Employed?'
X[2] = 'Previous Employers'
X[3] = 'Level of Education'
X[4] = 'Top-tier School?'
X[5] = 'Interned?'
X[6] = 'Hired?'
BS = 1; MS = 2; PhD = 3
N = 0; Y = 1

Encoded Table

	X[0]	X[1]	X[2]	X[3]	X[4]	X[5]	Hired?
1)	10	1	4	1	0	0	1
2)	0	0	0	1	1	1	1
3)	7	0	6	1	0	0	0
4)	2	1	1	2	1	0	1
5)	20	0	2	3	1	0	0
6)	0	0	0	3	1	1	1
7)	5	1	2	2	0	1	1
8)	3	0	1	1	0	1	1
9)	15	1	5	1	0	0	1
10)	0	0	0	1	0	0	0
11)	1	0	1	3	1	0	0
12)	4	1	1	2	0	1	1
13)	0	0	0	3	1	0	1

HiringData.tsv

10	1	4	1	0	0	1
0	0	0	1	1	1	1
7	0	6	1	0	0	0
2	1	1	2	1	0	1
20	0	2	3	1	0	0
0	0	0	3	1	1	1
5	1	2	2	0	1	1
3	0	1	1	0	1	1
15	1	5	1	0	0	1
0	0	0	1	0	0	0
1	0	1	3	1	0	0
4	1	1	2	0	1	1
0	0	0	3	1	0	1

Displaying Hiring Data

hiring_decision_tree.py

```
import numpy as np
import scipy as sp
import sys
from sklearn import tree

data = None
if __name__ == '__main__':
    data = sp.genfromtxt(sys.argv[1], delimiter='\t')
    print('Original data:')
    print(data)
```

output

10	1	4	1	0	0	1
0	0	0	1	1	1	1
7	0	6	1	0	0	0
2	1	1	2	1	0	1
20	0	2	3	1	0	0
0	0	0	3	1	1	1
5	1	2	2	0	1	1
3	0	1	1	0	1	1
15	1	5	1	0	0	1
0	0	0	1	0	0	0
1	0	1	3	1	0	0
4	1	1	2	0	1	1
0	0	0	3	1	0	1

Preparing Hiring Data

hiring_decision_tree.py

```
## get the target vector
target = data[:,6]
print('Target: %s' % str(target))
## convert the target vector to integers
vf = np.vectorize(lambda x: int(x))
target = vf(target)
print('Target: %s' % str(target))
# delete the Hired? column
data = np.delete(data, [6], 1)
print('Prepared data:')
print(data)
```

```
Target: [1 1 0 1 0 1 1 1 1 0 0 1 1]
```

```
[[ 10.   1.   4.   1.   0.   0.]
 [  0.   0.   0.   1.   1.   1.]
 [  7.   0.   6.   1.   0.   0.]
 [  2.   1.   1.   2.   1.   0.]
 [ 20.   0.   2.   3.   1.   0.]
 [  0.   0.   0.   3.   1.   1.]
 [  5.   1.   2.   2.   0.   1.]
 [  3.   0.   1.   1.   0.   1.]
 [ 15.   1.   5.   1.   0.   0.]
 [  0.   0.   0.   1.   0.   0.]
 [  1.   0.   1.   3.   1.   0.]
 [  4.   1.   1.   2.   0.   1.]
 [  0.   0.   0.   3.   1.   0.]]
```

Learning Hiring Decision Tree

hiring_decision_tree.py

```
## training and saving the tree
clf = tree.DecisionTreeClassifier(random_state=0)
dtr = clf.fit(data, target)
tree.export_graphviz(dtr, out_file='hiring_dtr.dot')
## testing the decision tree
test_case = np.array([1, 1, 0, 0, 0, 0])
print(dtr.predict(test_case))
```

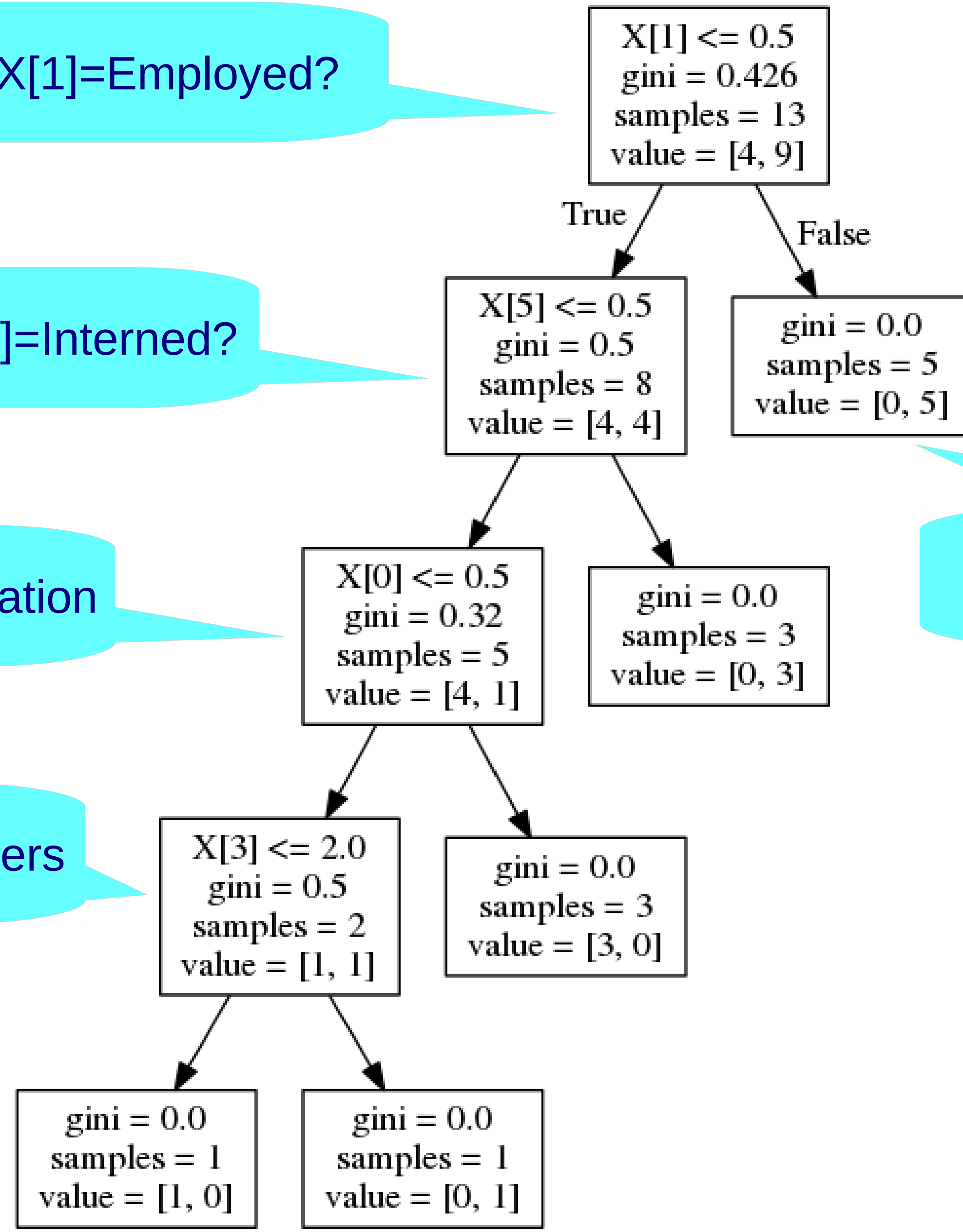
Learned Hiring Decision Tree

1st Decision Node X[1]=Employed?

2nd Decision Node X[5]=Interned?

3rd Decision Node X[0]=Education

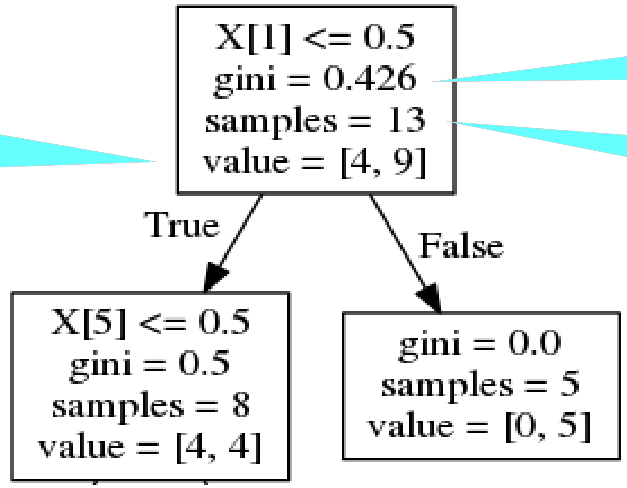
4th Decision Node X[3]=Employers



value [0, 5] means that 0 negative samples and 5 positive samples.

1st Decision Node: X[1] = Employed?

Of the 13 samples, 4 are not hired and 9 are hired



Information Gain

There are 13 samples

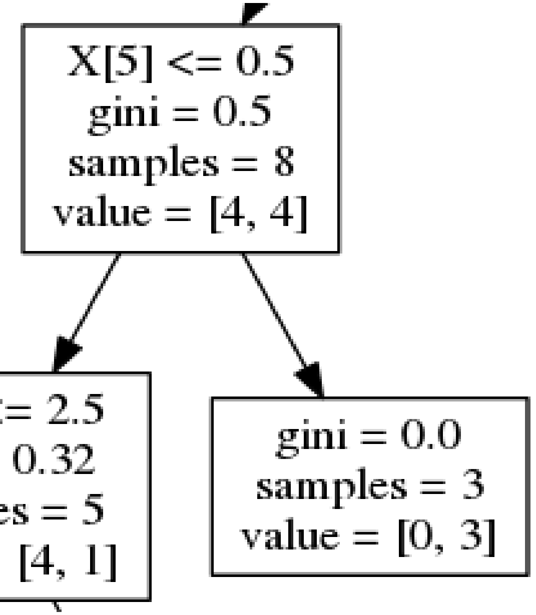
	X[0]	X[1]	X[2]	X[3]	X[4]	X[5]	Hired?
2)	0	0	0	1	1	1	1
3)	7	0	6	1	0	0	0
5)	20	0	2	3	1	0	0
6)	0	0	0	3	1	1	1
8)	3	0	1	1	0	1	1
10)	0	0	0	1	0	0	0
11)	1	0	1	3	1	0	0
13)	0	0	0	3	1	0	1

[4, 4] – 4 not hired, 4 hired

	X[0]	X[1]	X[2]	X[3]	X[4]	X[5]	Hired?
1)	10	1	4	1	0	0	1
4)	2	1	1	2	1	0	1
7)	5	1	2	2	0	1	1
9)	15	1	5	1	0	0	1
12)	4	1	1	2	0	1	1

[0, 5] – 0 not hired, 5 hired

2nd Decision Node: X[5] = Interned?



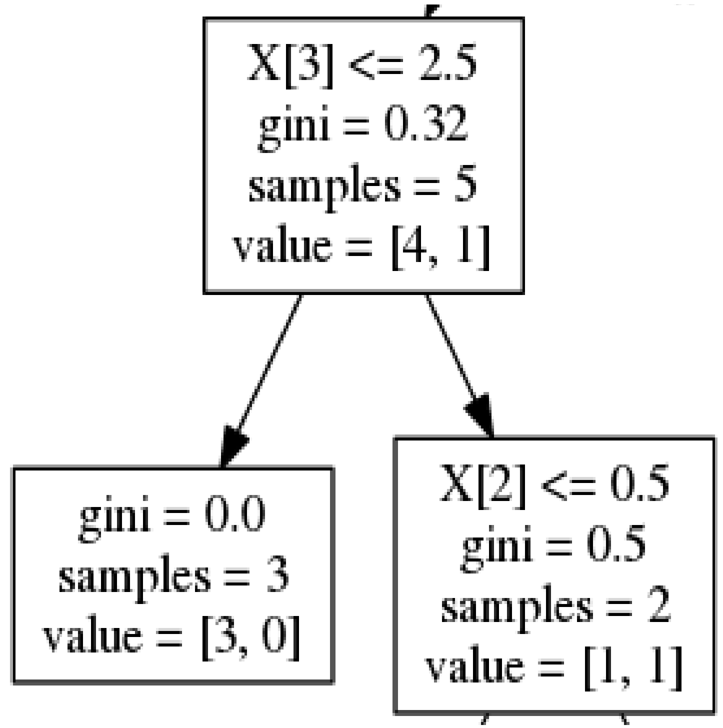
	X[0]	X[1]	X[2]	X[3]	X[4]	X[5]	Hired?
3)	7	0	6	1	0	0	0
5)	20	0	2	3	1	0	0
10)	0	0	0	1	0	0	0
11)	1	0	1	3	1	0	0
13)	0	0	0	3	1	0	1

[4, 1] – 4 not hired, 1 hired

	X[0]	X[1]	X[2]	X[3]	X[4]	X[5]	Hired?
2)	0	0	0	1	1	1	1
6)	0	0	0	3	1	1	1
8)	3	0	1	1	0	1	1

[0, 3] – 0 not hired, 3 hired

3rd Decision Node: X[3] = Level of Education



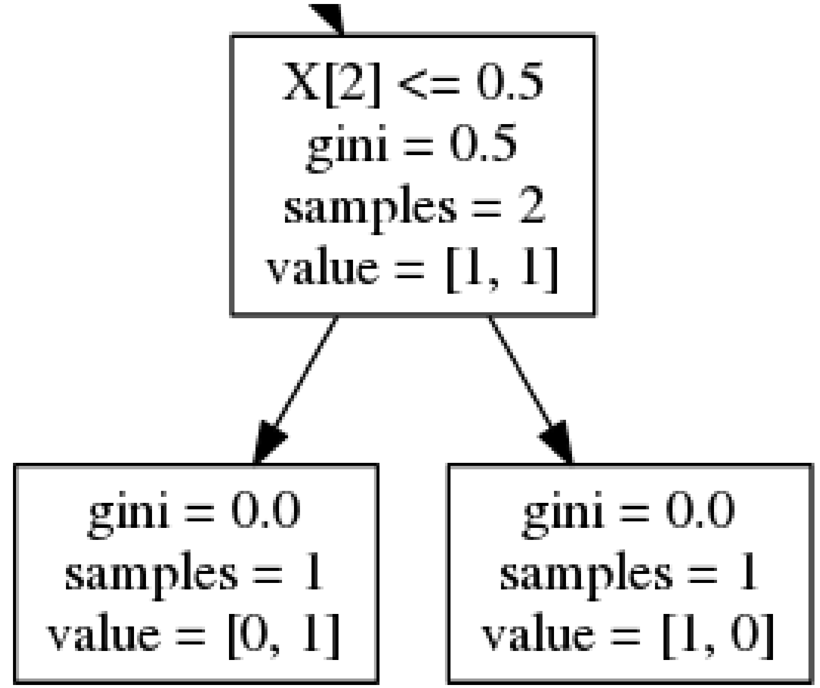
	X[0]	X[1]	X[2]	X[3]	X[4]	X[5]	Hired?
3)	7	0	6	1	0	0	0
5)	20	0	2	3	1	0	0
10)	0	0	0	1	0	0	0

[3, 0] – 3 not hired, 0 hired

	X[0]	X[1]	X[2]	X[3]	X[4]	X[5]	Hired?
11)	1	0	1	3	1	0	0
13)	0	0	0	3	1	0	1

[1, 1] – 1 not hired, 1 hired

4th Decision Node: Previous Employers



	X[0]	X[1]	X[2]	X[3]	X[4]	X[5]	Hired?
13)	0	0	0	3	1	0	1

	X[0]	X[1]	X[2]	X[3]	X[4]	X[5]	Hired?
11)	1	0	1	3	1	0	0

[0, 1] – 0 not hired, 1 hired

[1, 0] – 1 not hired, 0 hired

Problem

Let's learn a decision tree for the IRIS dataset.

IRIS Features

['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

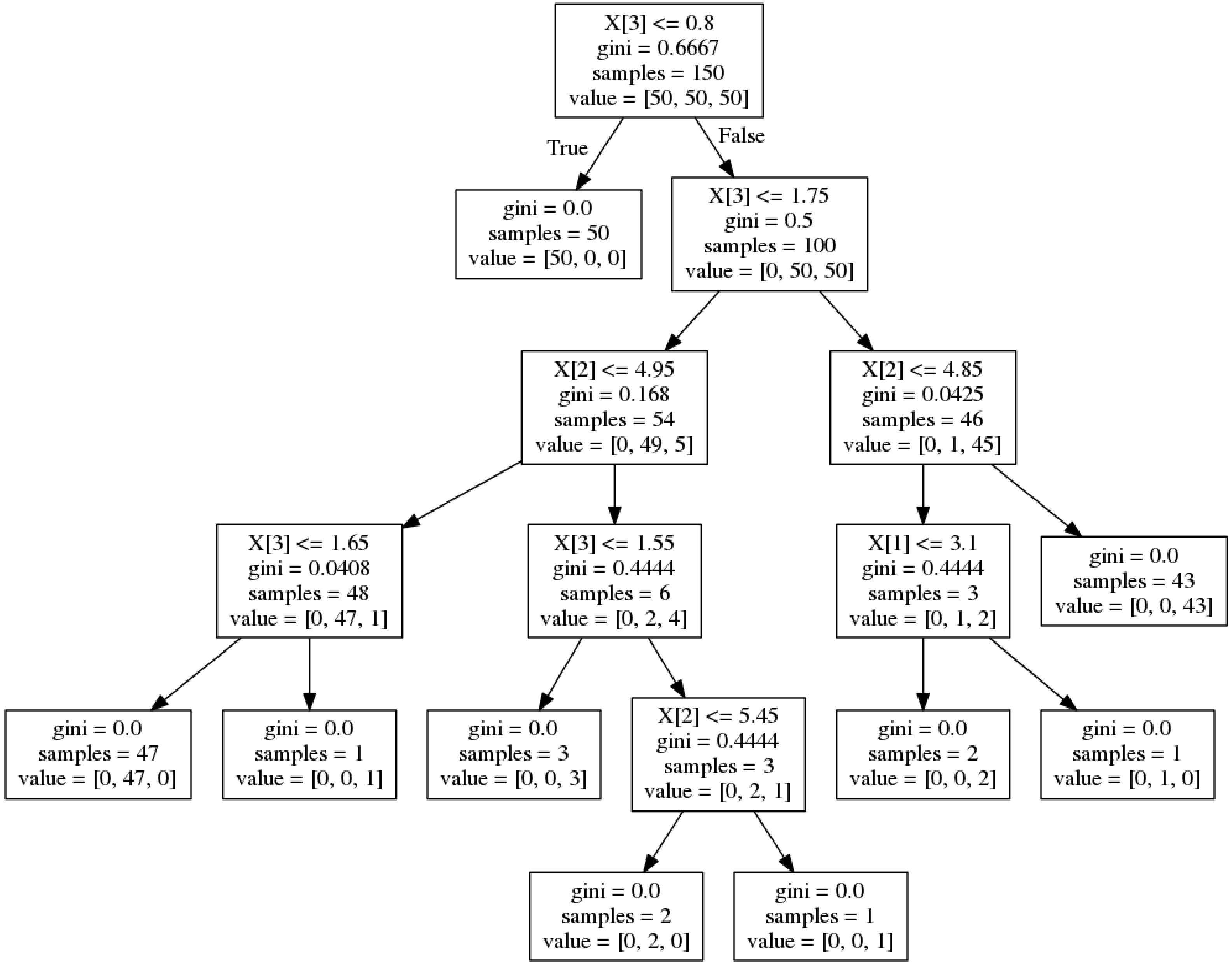
X[0] = 'sepal length (cm)'
X[1] = 'sepal width (cm)'
X[2] = 'petal length (cm)'
X[3] = 'petal width (cm)'

Solution

```
from sklearn.datasets import load_iris
from sklearn import tree

iris_data     = load_iris()
data_items = iris_data.data
target          = iris_data.target
clf = tree.DecisionTreeClassifier(random_state=0)
dtr = clf.fit(data_items, target)
tree.export_graphviz(dtr, out_file='iris_tree.dot')
```

Learned IRIS Decision Tree



Doing Train/Test Split

```
from sklearn.model_selection import train_test_split

def run_train_test_split_once(classifier):

    X_train, X_test, y_train, y_test = \

            train_test_split(data_items, target,

                    test_size=0.33,

                    random_state=randint(0,1000))

    dt = clf.fit(X_train, y_train)

    print(X_test.shape)

    print(y_test.shape)

    print(sum(dt.predict(X_test) == y_test)/(len(y_test)*1.0))
```

Split data and labels into train and test.

Train a classifier on train data (X_train) and train labels (y_train)

Compute model's accuracy

Running K-Fold Cross Validation

```
from sklearn.model_selection import cross_val_predict

def run_cross_validation(classifier, n):

    for i in xrange(n):

        ## cv specifies the number of folds data

        for cv in xrange(5, 11):

            cross_val = cross_val_predict(classifier, data_items, target, cv=cv)

            acc = sum(cross_val==target)/float(len(target))

            print cv, acc

        print '-----------------'
```

source code in iris_decision_tree.py