SciComp with Py

Learning and Evaluating Decision Trees

Vladimir Kulyukin
Department of Computer Science
Utah State University

Outline

- Review
- Decision Tree Structure
- Learning and Evaluating Decision Trees for the IRIS Dataset
- Confusion Matrices
- Precision, Recall, F1
- DIGITS Dataset

Review

Entropy

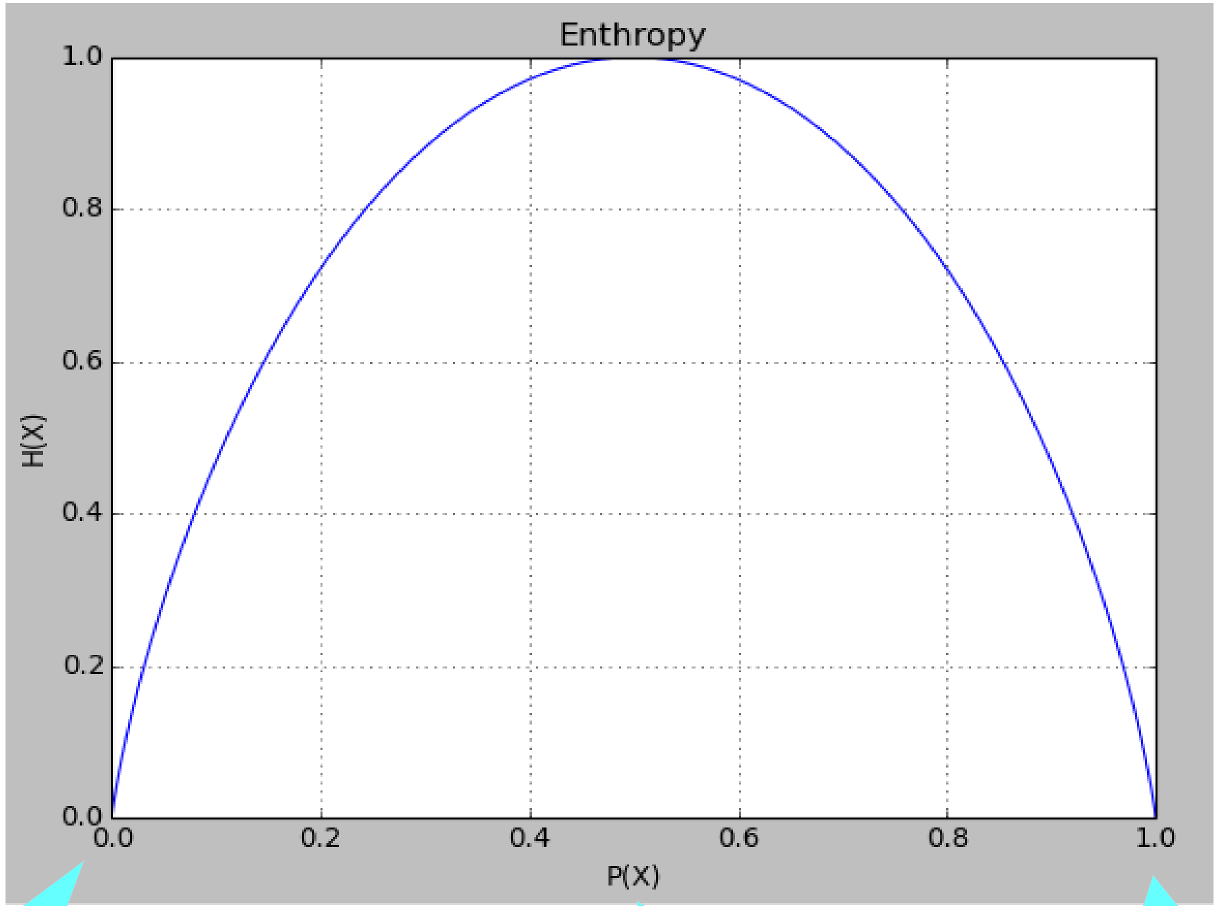
- How does one select the best attribute to split a set of samples?
- This selection is based on entropy, a function that measures the diversity of a set
- The more diverse a set, the higher its entropy
- Put another way, the more diverse a set, the more information it contains

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.

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

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

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

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

Set of samples

Attribute

Number of elements in S for which A=v

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

Entropy of S

Set of all possible values of A

Entropy of S_v

Number of elements in S

Decision Tree Learning 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.

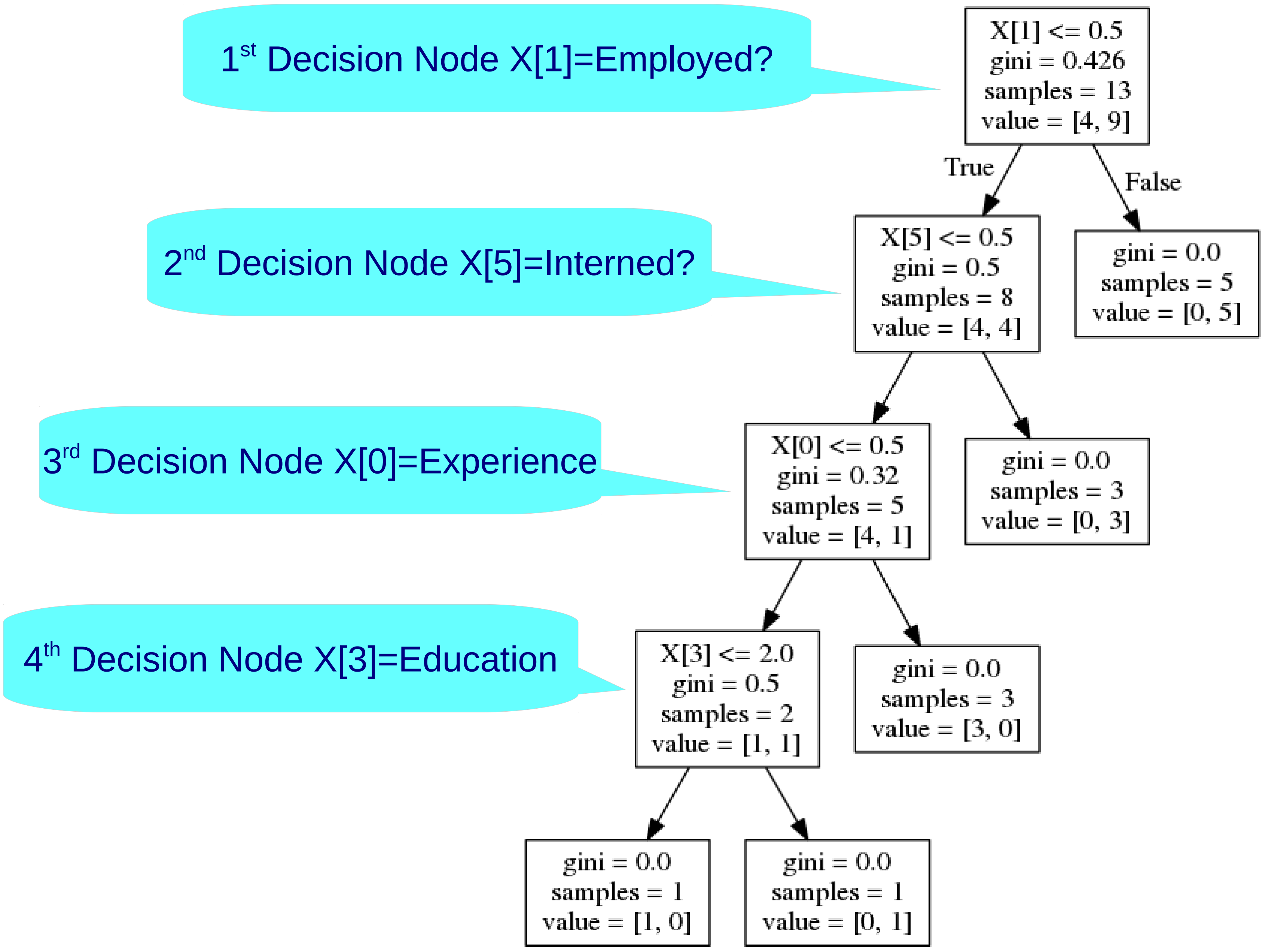
Problem

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.

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

Learned Hiring Decision Tree



Learning and Evaluating Decision Trees for IRIS Dataset

Review: Iris Dataset Details

- The dataset consists of 150 flowers (data items)
- Data items are iris plants of three species: setosa, versicolor, and virgnica
- Each plant is characterized by four numerical attributes (**features**): sepal length (cm), sepal width (cm), petal length (cm), petal width (cm): many thanks to all those meticulous botanists!
- Each plant is labeled by its species (in sklearn lingo, the class/species label is called a **target**)

Review: Iris Flower Species

Iris Setosa

Iris Versicolor

Iris Virginica

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')
```

Construct a decision tree classifier

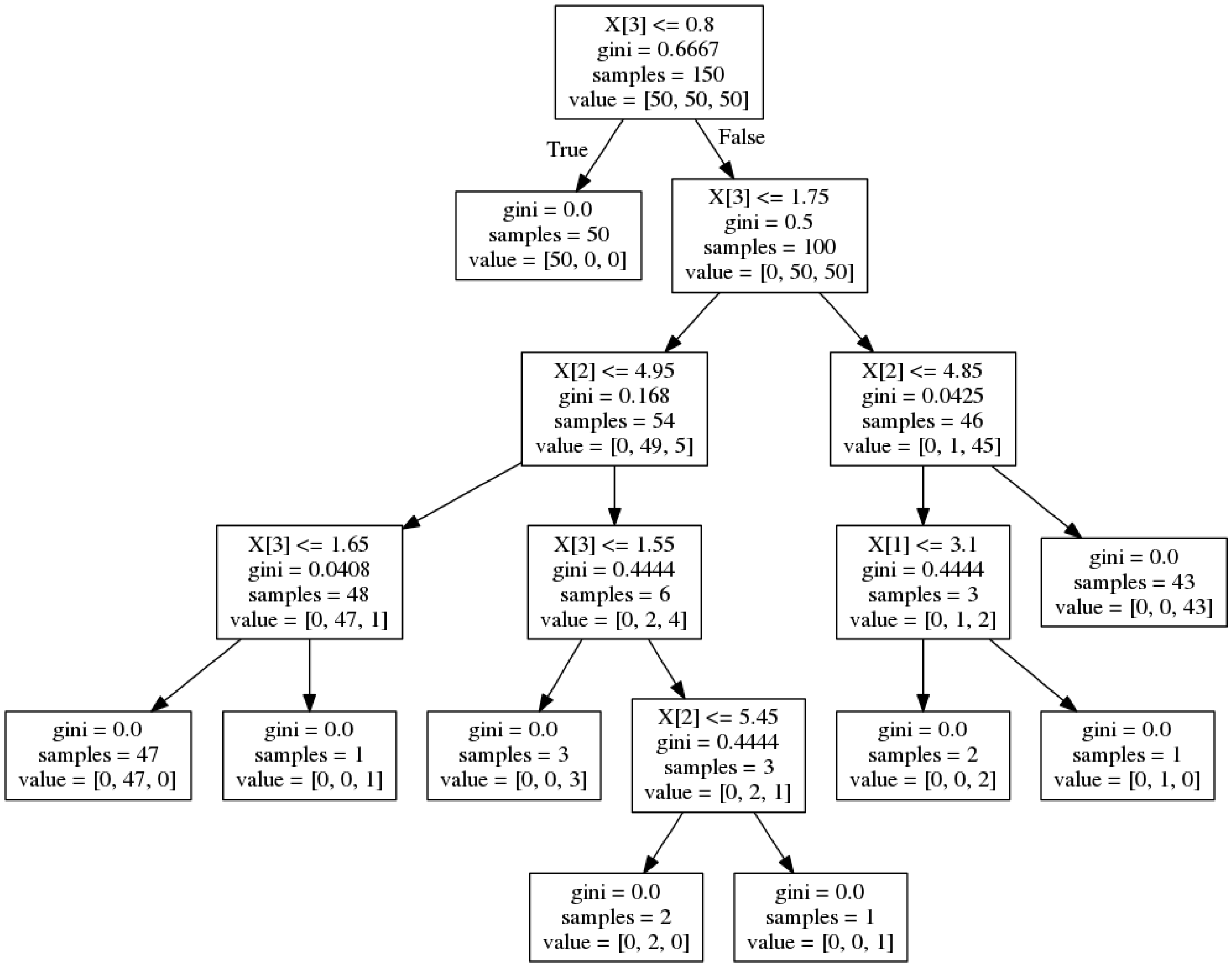
Train a decision tree on the iris data

source in iris_decision_tree.py

Feature Name Array

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

Learned IRIS Decision Tree



Testing/Train Split

- Estimating accuracy of a models *only on the training data* gives over optimistic results
- What we really want is testing how a model performs on new instances (i.e., instances it has not seen before)
- Data items are split into two non-overlapping sets: training data and testing data
- Models are trained on training data and evaluated on testing data

Train/Test Split

```
from sklearn.model_selection import train_test_split

def run_train_test_split(classifier, n, test_size):
    for i in xrange(n):
        train_data, test_data, train_target, test_target = \
            train_test_split(data_items, target,
                test_size=test_size, random_state=randint(0, 1000))
        dt = classifier.fit(train_data, train_target)
        print(sum(dt.predict(test_data) == test_target)/float(len(test_target)))
```

Run train/test split n times

% of data used for testing

Split the data into training and testing sets

Train a decision tree

Count accuracy

source in iris_decision_tree.py

Sample Run

Run train/test split 10 times

Take 30% of data for testing

```
>>> run_train_test_split(clf, 10, 0.3)
0.955555555556
0.933333333333
0.977777777778
0.933333333333
0.933333333333
0.911111111111
0.955555555556
0.866666666667
0.955555555556
0.955555555556
```

K-Fold Cross-Validation

- Cross-validation is another model evaluation technique
- Given a data set, split it into some number (K > 1) of subsets (folds)
- Randomly select one fold for testing and use the remaining K-1 folds for training
- Repeat the previous step multiple times to select different folds for testing and training

Cross-Validation

```
from sklearn.model_selection import cross_val_predict
```

Decision tree classifier

# of times cross-validation is executed

```
def run_cross_validation(dtr, n):
    for i in xrange(n):
        ## cv specifies the number of folds data is split into
        for cv in xrange(5, 16):
            cross_val = cross_val_predict(dtr, data_items, target, cv=cv)
            acc = sum(cross_val==target)/float(len(target))
            print cv, acc
```

source in iris_decision_tree.py

Sample Run

Decision tree classifier

Run 1 cross validation

```
>>> run_cross_validation(dtr, 1)
cross-validation run 0
num_folders 5, accuracy = 0.784641
----------------------------------------
num_folders 6, accuracy = 0.799666
----------------------------------------
num_folders 7, accuracy = 0.798553
----------------------------------------
...
----------------------------------------
num_folders 14, accuracy = 0.795771
----------------------------------------
num_folders 15, accuracy = 0.813022
----------------------------------------
```

Confusion Matrices

Confusion Matrix

- Confusion matrix is another commonly used evaluation technique for classification models
- If a trained model classifies into C1, …, Cn classes, the confusion matrix (CM) is an n x n matrix where an entry [i, j] tells us how frequently samples of class Ci were classified as samples of class Cj during testing
- An ideal confusion matrix has 1's on its main diagonal and 0's everywhere else

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
from matplotlib import pylab

def compute_and_plot_cm(classifier, test_size):
    train_data, test_data, train_target, test_target = \
                train_test_split(data_items, target,
                                 test_size=test_size, random_state=randint(0, 1000))
    dt = classifier.fit(train_data, train_target)
    test_predict = dt.predict(test_data)
    cm = confusion_matrix(test_target, test_predict)
    plot_cm(cm, ['setosa', 'versicolor', 'virginica'], 'IRIS DT CM')
```

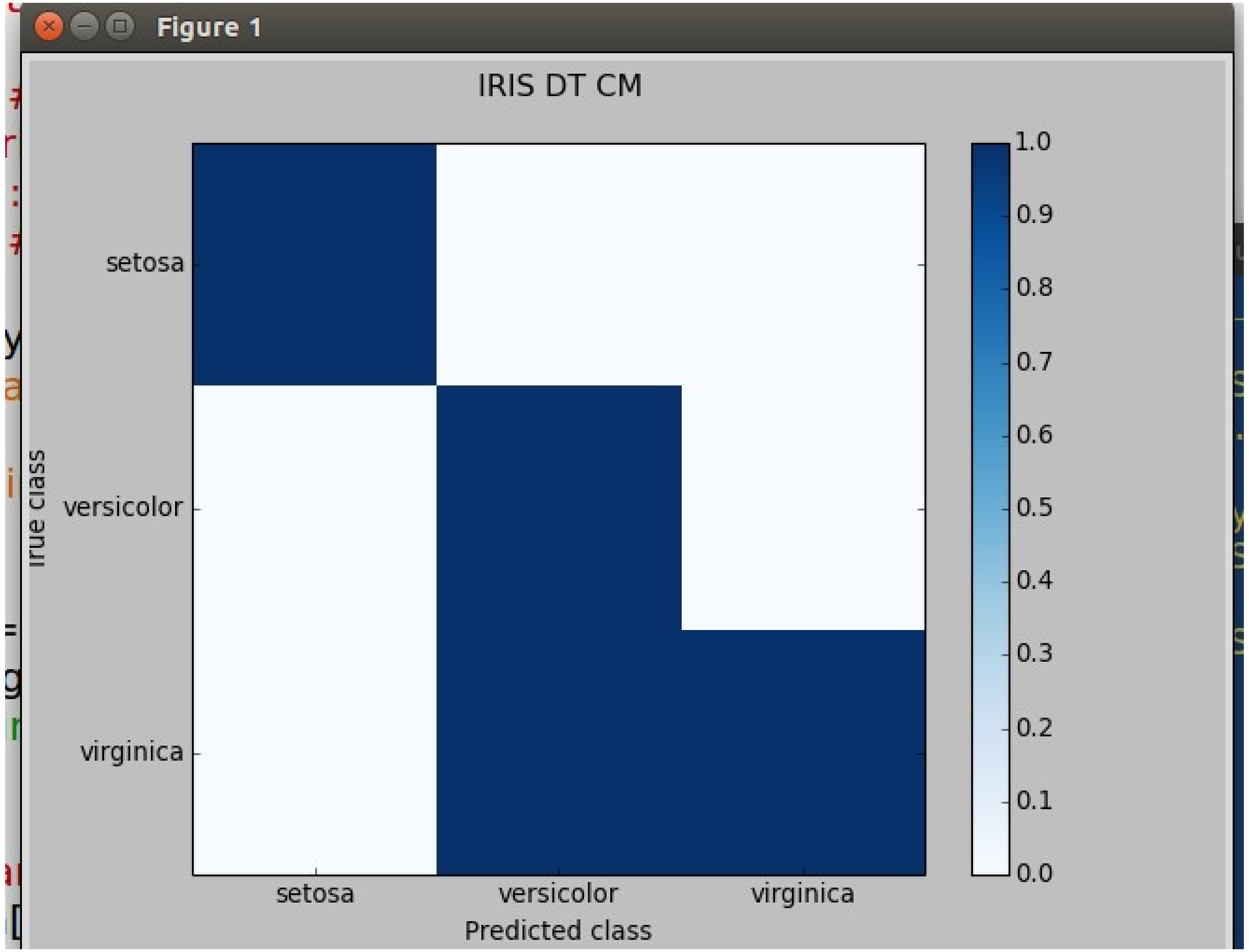
source in iris_decision_tree.py

Confusion Matrix for IRIS Decision Tree

Confusion Matrix

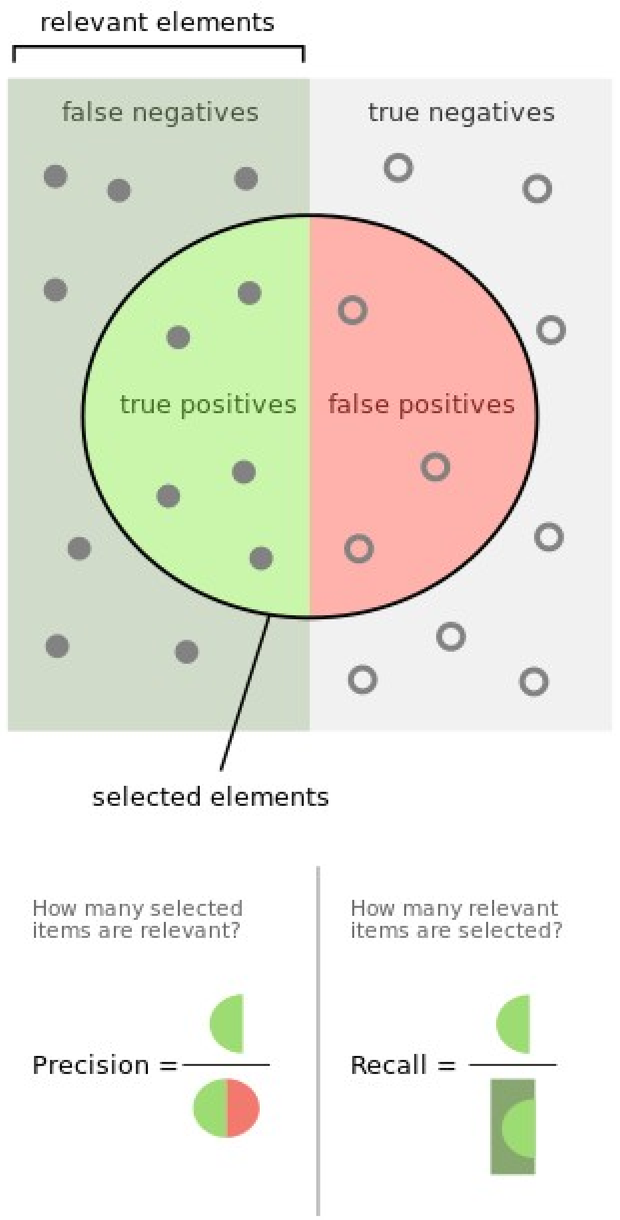
```
[[13  0  0]
 [ 0 15  0]
 [ 0  2 15]]
```

Confusion Matrix Plot



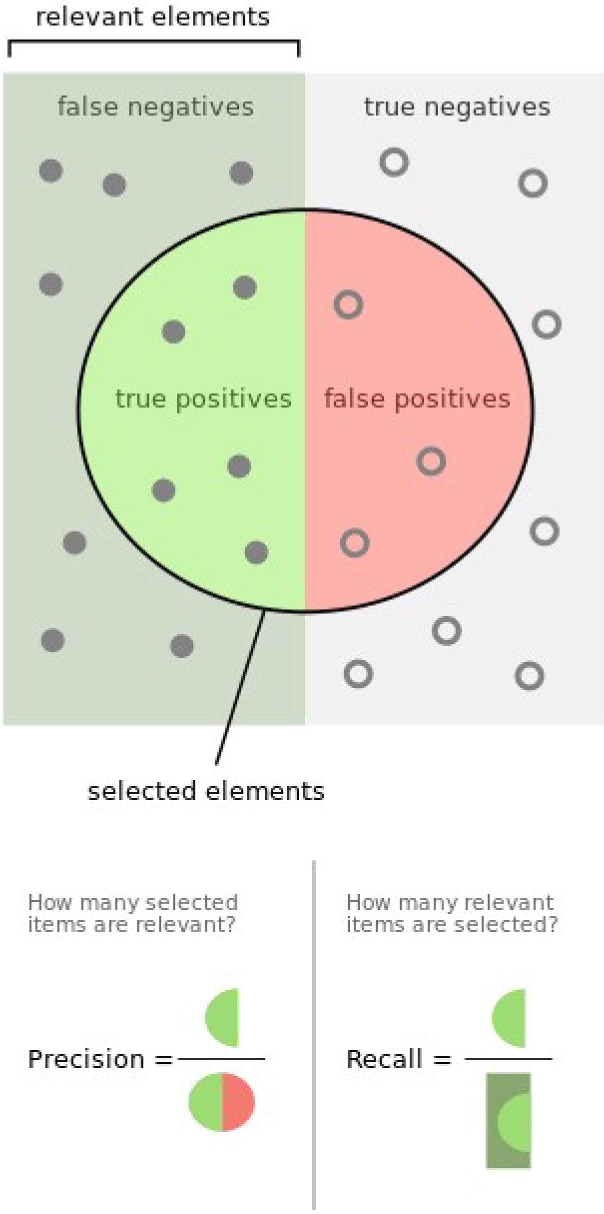
Recall, Precision, F1

Precision and Recall



- False negatives – relevant elements classified as irrelevant
- False positives – irrelevant elements classified as relevant
- True negatives – irrelevant elements classified as irrelevant
- True positives – relevant elements classified as relevant

Precision and Recall

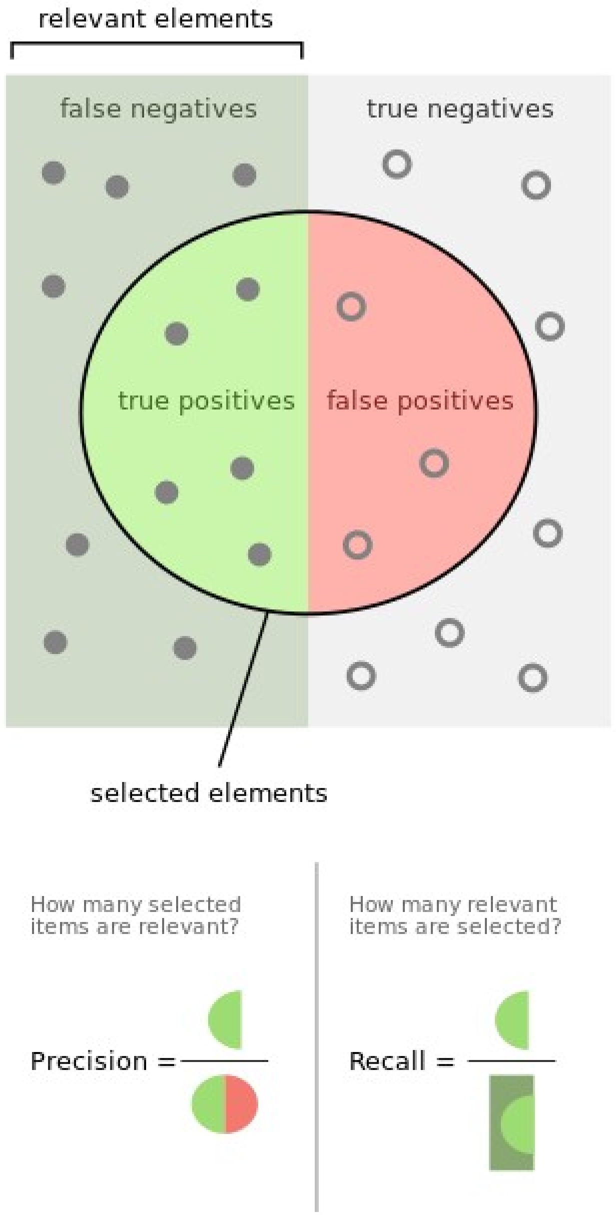


$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

- tp – # of true positives
- fp – # of false positives
- fn - # of false negatives

Combining Precision and Recall



$$F_1 = 2 \cdot \frac{1}{\frac{1}{\mathrm{recall}} + \frac{1}{\mathrm{precision}}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

Classification Report Example

```
>>> y_true = [0, 1, 2, 2, 2]
>>> y_pred = [0, 0, 2, 2, 1]
>>> target_names = ['A', 'B', 'C']
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_true, y_pred, target_names=target_names))
              precision    recall  f1-score   support

          A       0.50      1.00      0.67         1
          B       0.00      0.00      0.00         1
          C       1.00      0.67      0.80         3

avg / total       0.70      0.60      0.61         5
```

# of items of each class: 1 item of A; 1 item of B; 3 items of B

total # of items

weighted average: (0.5*1 + 0.0*1 + 1.0*3)/5 = 0.70

Classification Report for IRIS Data Set

```
>>> compute_cm_cr(dtr, 0.3)
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        18
          1       0.88      1.00      0.94        15
          2       1.00      0.83      0.91        12

avg / total       0.96      0.96      0.95        45


Confusion matrix:
[[18  0  0]
 [ 0 15  0]
 [ 0  2 10]]
```

Classification Report for IRIS

Confusion Matrix for IRIS

Learning and Evaluating Decision Trees for DIGITS Dataset

Digits Dataset

- Digits is one of the sklearn image datasets
- This dataset consists of 1,797 images of handwritten characters
- Each image is 8 x 8, i.e., has 64 pixels.
- In numpy terms, each image is a 64-element array of floats
- The target vector for this dataset is [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Digits Dataset

```
from sklearn.datasets import load_digits


digits_data = load_digits()
data_items = digits_data.data
target = digits_data.target
```

source in digits_decision_tree.py

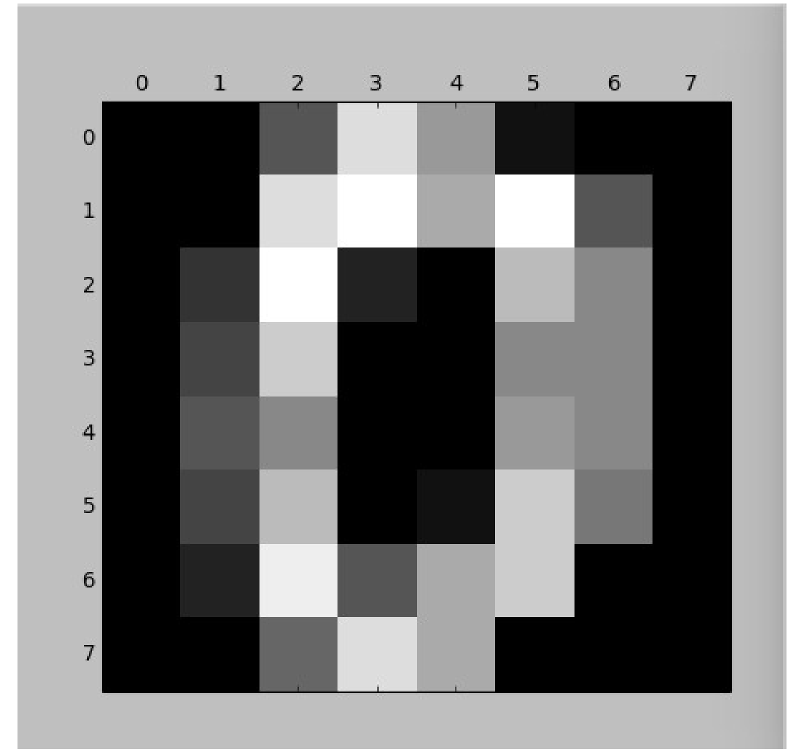
Digits Dataset

```
print(data_items[0])

plt.gray()

plt.matshow(digits_data.images[0])

plt.show()
```




```
[  0.   0.   5.  13.   9.   1.   0.   0.   0.   0.  13.  15.
10.  15.   5. 0.   0.   3.  15.   2.   0.  11.   8.   0.   0.
4.  12.   0.   0.   8. 8.   0.   0.   5.   8.   0.   0.   9.   8.
0.   0.   4.  11.   0.   1. 12.   7.   0.   0.   2.  14.   5.
10.  12.   0.   0.   0.   0.   6.  13. 10.   0.   0.   0.]
```

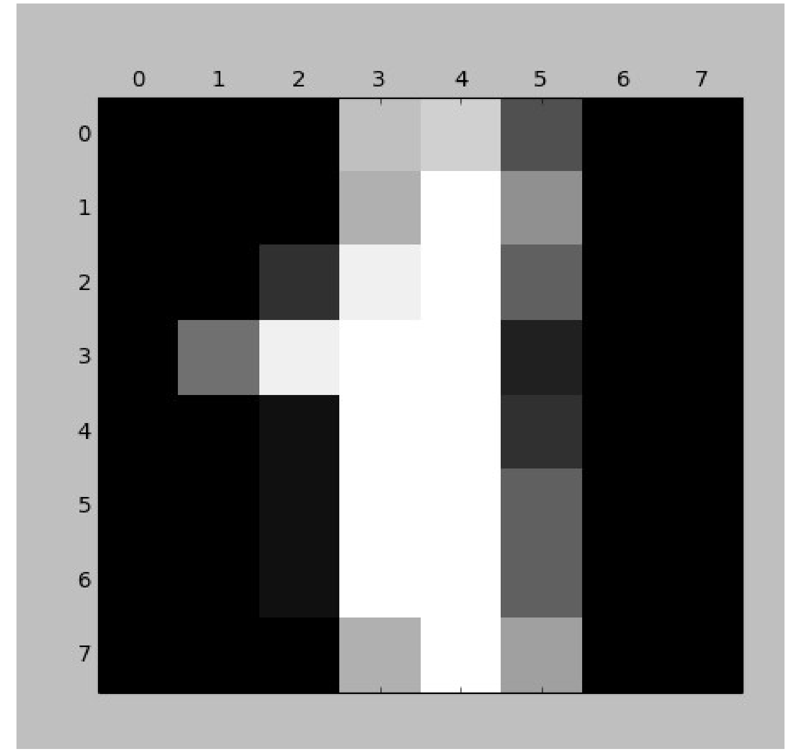
Digits Dataset

```
print(data_items[1])

plt.gray()

plt.matshow(digits_data.images[1])

plt.show()
```



[0. 0. 0. 12. 13. 5. 0. 0. 0. 0. 0. 11.
16. 9. 0. 0. 0. 0. 3. 15. 16. 6. 0. 0. 0. 7.
15. 16. 16. 2. 0. 0. 0. 0. 1. 16. 16. 3. 0.
0. 0. 0. 1. 16. 16. 6. 0. 0. 0. 0. 1. 16. 16.
6. 0. 0. 0. 0. 0. 11. 16. 10. 0. 0.]

Digits Dataset

	precision	recall	f1-score	support
0	1.00	0.95	0.98	44
1	0.71	0.78	0.74	58
2	0.91	0.83	0.87	48
3	0.77	0.78	0.77	59
4	0.76	0.84	0.80	57
5	0.91	0.88	0.90	49
6	0.94	0.85	0.89	59
7	0.85	0.87	0.86	52
8	0.73	0.77	0.75	47
9	0.82	0.81	0.81	67
avg / total	0.84	0.83	0.83	540

Confusion matrix:

```
[[42  0  0  0  1  0  0  0  1  0]
 [ 0 45  0  3  3  0  0  1  3  3]
 [ 0  4 40  0  0  0  1  0  3  0]
 [ 0  0  1 46  2  1  0  1  3  5]
 [ 0  3  0  0 48  1  1  2  0  2]
 [ 0  0  1  0  2 43  1  1  1  0]
 [ 0  1  0  0  5  1 50  0  1  1]
 [ 0  2  0  3  2  0  0 45  0  0]
 [ 0  5  2  2  0  0  0  1 36  1]
 [ 0  3  0  6  0  1  0  2  1 54]]
```

References

- http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html
- http://scikit-learn.org/stable/auto_examples/datasets/plot_digits_last_image.html
- http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html
- https://en.wikipedia.org/wiki/Precision_and_recall