# CS 3430: SciComp with Py
# Assignment 12
# Learning and Testing Decision Trees to Classify Handwritten Digits

Vladimir Kulyukin
Department of Computer Science
Utah State University

April 15, 2017

## 1 Learning Objectives

1. Decision Trees
2. Train/Test Splits
3. Cross-Validation
4. Confusion Matrices

## 2 Introduction

In this assignment, you will learn how to train and evaluate decision trees to classify handwritten digits. The data for this assignment comes from the `digits` datasets, one of the standard `sklearn.datasets` datasets. The target vector for this dataset is `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`.

## 3 DIGITS Dataset

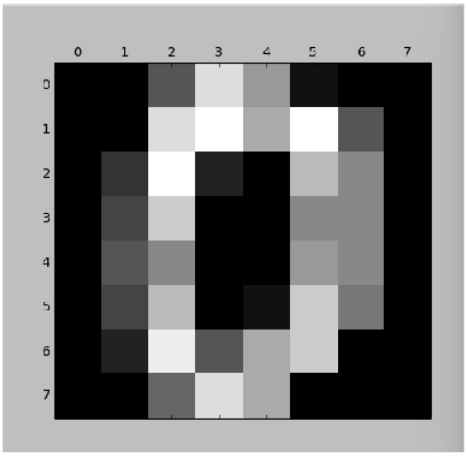


Figure 1: An 8x8 image of a handwritten character 0

This `DIGITS` dataset consists of 1797 images of handwritten digit characters. Each image is 8 x 8, i.e., it has 64 pixels. In numpy terms, each image is a 64-element array of floats. Figure 1 shows an 8x8 image of a handwritten character 0. The array below is the 64-element numpy array of floats corresponding to the image in Figure 1.

```
[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.]
```

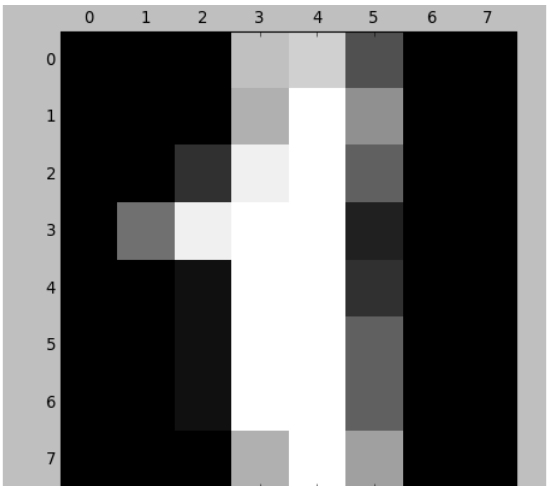


Figure 2: An 8x8 image of handwritten character 1

Figure 2 shows an 8x8 image of a handwritten character 1. The array below is the 64-element numpy array of floats corresponding to the image in Figure 2.

```
[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.]
```

# 4 Learning Decision Trees

Implement the program `digits_decision_tree.py` that trains and evaluates decision trees for the digits dataset. Start by loading the dataset and setting the data items and target vectors.

```
from sklearn.datasets import load_digits

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

Then implement two functions `run_train_test_split` that takes a classifier, a number of times `n` that the train/test split evaluation is run, and the `test_size` parameter indicating the percentage of the data used for testing (e.g., 0.3, 0.4, etc.). This function uses `train_test_split` function to split the data into `train_data`, `test_data`, `train_target`, and `test_target`, learns a decision tree on `train_data` and `train_target`, and computes and prints out the accuracy of the decision tree on `test_data` and `test_target`.

```
def run_train_test_split(classifier, n, test_size):
    ## your code
    pass
```

Here is a trial run.

```
>>> run_train_test_split(clf, 10, 0.3)
train/test run 0: accuracy = 0.850000
----------------------------------------
train/test run 1: accuracy = 0.835185
----------------------------------------
train/test run 2: accuracy = 0.848148
----------------------------------------
train/test run 3: accuracy = 0.840741
```

```
----------------------------------------
train/test run 4: accuracy = 0.850000
----------------------------------------
train/test run 5: accuracy = 0.859259
----------------------------------------
train/test run 6: accuracy = 0.831481
----------------------------------------
train/test run 7: accuracy = 0.861111
----------------------------------------
train/test run 8: accuracy = 0.850000
----------------------------------------
train/test run 9: accuracy = 0.850000
----------------------------------------
```

Now implement the function `run_cross_validation` that takes a learned decision tree `dtr` and the number of experiments `n` and then runs the cross-validation with a number of folders ranging from 5 to 16.

```
def run_cross_validation(dtr, n):
    ## your code
    pass
```

Here is a test run.

```
>>> run_cross_validation(dtr, 2)
cross-validation run 0
num_folders 5, accuracy = 0.784641
----------------------------------------
num_folders 6, accuracy = 0.799666
----------------------------------------
num_folders 7, accuracy = 0.798553
----------------------------------------
num_folders 8, accuracy = 0.815804
----------------------------------------
num_folders 9, accuracy = 0.828047
----------------------------------------
num_folders 10, accuracy = 0.830273
----------------------------------------
num_folders 11, accuracy = 0.810796
----------------------------------------
num_folders 12, accuracy = 0.823038
----------------------------------------
num_folders 13, accuracy = 0.819699
----------------------------------------
num_folders 14, accuracy = 0.795771
----------------------------------------
num_folders 15, accuracy = 0.813022
----------------------------------------
cross-validation run 1
num_folders 5, accuracy = 0.784641
----------------------------------------
num_folders 6, accuracy = 0.799666
----------------------------------------
num_folders 7, accuracy = 0.798553
----------------------------------------
num_folders 8, accuracy = 0.815804
----------------------------------------
num_folders 9, accuracy = 0.828047
----------------------------------------
num_folders 10, accuracy = 0.830273
----------------------------------------
num_folders 11, accuracy = 0.810796
----------------------------------------
num_folders 12, accuracy = 0.823038
----------------------------------------
num_folders 13, accuracy = 0.819699
```

```
----------------------------------------
num_folders 14, accuracy = 0.795771
----------------------------------------
num_folders 15, accuracy = 0.813022
----------------------------------------
```

Finally, write the function `compute_train_test_confusion` that takes a classifier and a test size percentage and does one train/test split on the data, learns a decision tree, and displays a confusion matrix in a plot.

```
def compute_train_test_confusion_matrix(classifier, test_size):
    ## your code
    pass
```

# 5 What To Submit

The zip archive `hw12.zip` contains `digits_decision_tree.py` with the starter code. Write your code in this file and submit it via Canvas.

Happy Hacking!