In [ ]:

```
#  Ebnable HTML/CSS
from IPython.core.display import HTML
HTML("<link href='https://fonts.googleapis.com/css?family=Passion+One' rel='stylesheet' type='text/css'><style>div.attn { font-family: 'Helvetica Neue'; font-size: 30px; line-height: 40px; color: #FFFFFF; text-align: center;
```

Enter Team Member Names here (double click to edit):

- Name 1: Andre Mauldin
- Name 2: Ben Goodwin

Live Session Assignment Two

In the following assignment you will be asked to fill in python code and derivations for a number of different problems. Please read all instructions carefully and turn in the rendered notebook (.ipynb file, remember to save it!!) or HTML of the rendered notebook before the end of class.

Contents

Back to Top

Loading the Classification Data

Please run the following code to read in the "digits" dataset from sklearn's data loading module. This is identical to the first in class assignment for loading the data into matrices. `ds.data` is a matrix of feature values and `ds.target` is a column vector of the class output (in our case, the hand written digit we want to classify). Each class is a number (0 through 9) that we want to classify as one of ten hand written digits.

In [2]:

```
from sklearn.datasets import load_digits
import numpy as np
from __future__ import print_function

ds = load_digits()

# this holds the continuous feature data
print('features shape:', ds.data.shape) # there are 1797 instances and 64 features per instance
print('target shape:', ds.target.shape )
print('range of target:', np.min(ds.target),np.max(ds.target))
```

```
features shape: (1797, 64)
target shape: (1797,)
range of target: 0 9
```

Back to Top

Using Decision Trees

In the videos, we talked about the splitting conditions for different attributes. Specifically, we discussed the number of ways in which it is possible to split a node, depending on the attribute types. To understand the possible splits, we need to understand the attributes. For the question below, you might find the description in the `ds['DESCR']` field to be useful. You can see the field using `print(ds['DESCR'])`

Question 1: For the digits dataset, what are the type(s) of the attributes? How many attributes are there? What do they represent?

In [36]:

```
## Enter your comments here


print(ds['DESCR'])

## Enter comments here
#For this dataset we have: 64 Integer data attributes.
#This data contains images of hand-written digits
#The attributes are hand-written 8x8 int pixels between 0 and 1
```

```
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 1797
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.

.. topic:: References

  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of
    Graduate Studies in Science and Engineering, Bogazici University.
  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
    Linear dimensionalityreduction using relevance weighted LDA. School of
    Electrical and Electronic Engineering Nanyang Technological University.
    2005.
  - Claudio Gentile. A New Approximate Maximal Margin Classification
    Algorithm. NIPS. 2000.
```

Using the gini coefficient

We talked about the gini index in the videos. The gini coefficient for a **given split** is given by:

$$Gini = \sum_{t=1}^{T} \frac{n_t}{N} gini(t)$$

where T is the total number of splits (2 for binary attributes), n_t is the number of instances in node t after splitting, and N is the total number of instances in the parent node. $gini(t)$ is the **gini index for each individual node that is created by the split** and is given by:

$$gini(t) = 1 - \sum_{j=0}^{C-1} p(j|t)^2$$

where C is the total number of possible classes and $p(j|t)$ is the probability of class j in node t (i.e., $n_j ==$ the count of instances belonging to class j in node t, normalized by the total number of instances in node t).

$$p(j|t) = \frac{n_j}{n_t}$$

For the given dataset, $gini(t)$ has been programmed for you in the function `gini_index`.

- `def gini_index(classes_in_split):`
  - To use the function, pass in a `numpy` array of the class labels for a node as (i.e., pass in the rows from `ds.target` that make up a node in the tree) and the gini will be returned for that node.

In [3]:

```
# compute the gini of several examples for the starting dataset
# This function "gini_index" is written for you. Once you run this block, you
#   will have access to the function for the notebook. You do not need to know
#   how this function works--only what it returns
# This function returns the gini index for an array of classes in a node.
def gini_index(classes_in_split):
    # pay no attention to this code in the function-- it just computes the gini for a given split
    classes_in_split = np.reshape(classes_in_split,(len(classes_in_split),-1))
    unique_classes = np.unique(classes_in_split)
    gini = 1
    for c in unique_classes:
        gini -= (np.sum(classes_in_split==c) / float(len(classes_in_split)))**2

    return gini
```

In the example below, the function is used calculate the gini for splitting the dataset on feature 28, with value 2.5. In this example, we need to create two separate tree nodes: the first node has all the `ds.target` labels when feature 28 is greater than 2.5, the second node has all the rows from `ds.target` where feature 28 is less than 2.5. The steps are outlined below. **Read this carefully to understand what the code does below in the block following this.**

- Feature 28 is saved into a separate variable `feature28 = ds.data[:,28]`
- First all the target classes for the first node are calculated using `numpy` indexing `ds.target[feature28>2.5]`
  - Note: this grabs all the rows in `ds.target` (the classes) which have feature 28 greater than 2.5 (similar to indexing in pandas)
- Second, those classes are passed into the function to get the gini for the right node in this split (i.e., feature 28 being greater than the threshold 2.5).
  - `gini_r = gini_index(ds.target[feature28>2.5])`
- Third, the gini is calculated for the left node in the tree. This grabs only the rows in `ds.target` where feature 28 is less than 2.5.
  - `gini_l = gini_index(ds.target[feature28<=2.5])`
- Combining the gini indices is left as an exercise in the next section

In [4]:

```
#==========================Use the gini_index Example===============
# get the value for this feature as a column vector
# (this is like grabbing one column of the record table)
feature28 = ds.data[:,28]

# if we split on the value of 2.5, then this is the gini for each resulting node:
gini_r = gini_index(ds.target[feature28>2.5]) # just like in pandas, we are sending in the rows where feature28>2.5
gini_l = gini_index(ds.target[feature28<=2.5]) # and sending the rows where feature28<=2.5

# compute gini example. This splits on attribute '28' with a value of 2.5
print('gini for right node of split:', gini_r)
print('gini for left node of split:', gini_l)
```

```
gini for right node of split: 0.8845857867667073
gini for left node of split: 0.7115407566535388
```

Question 2: Now, using the above values `gini_r` and `gini_l`. Calculate the combined Gini for the entire split. You will need to write the weighted summation (based upon the number of instances inside each node). To count the number of instances greater than a value using numpy, you can use broadcasting, which is a special way of indexing into a numpy array. For example, the code `some_array>5` will return a new numpy array of true/false elements. It is the same size as `some_array` and is marked true where the array is greater than `5`, and false otherwise. By taking the `sum` of this array, we can count how many times `some_array` is greater than `5`.

`counts = sum(some_array>5)`

You will need to use this syntax to count the values in each node as a result of splitting.

In [5]:

```
## Enter your code here

# we need to make a weighted sum of the gini indices
num_instances_r1 = float(sum(feature28>2.5))
num_instances_l1 = float(sum(feature28<=2.5))
N = float(len(ds.target))

gini_total = (num_instances_r1*gini_r + num_instances_l1*gini_l) / N

## Enter your code here
print('The total gini of the split for a threshold of 2.5 is:',gini_total)
```

```
The total gini of the split for a threshold of 2.5 is: 0.8461634345045179
```

Start of Live Session Coding

Question 3: Now we want to know which is a better split:

- `feature28` split on a value of `2.5`
- `feature28` split on a value of `10`.

Enter your code to find the total gini of splitting on the threshold of 10 and compare it to the total gini of splitting on threshold of 2.5 (for feature 28 only). According to gini, which threshold is better for spliting on feature 28, `threshold=2.5` or `threshold=10.0`?

In [6]:

```
# Enter your code here
#As per the hint
T=10
num_instances_r = sum(feature28>T)
num_instances_l = sum(feature28<=T)
#Time to sum the instances
count_r=sum(feature28>T)
count_l=sum(feature28<=T)
#Combine the gini
giniT1 = (num_instances_r*gini_r)
giniT2 = (num_instances_l*gini_l)
gini_tot= (giniT1+giniT2)/N
# Enter your code here

print('The total gini of the split for a threshold of 10 is:',gini_tot)
print('This is better than the split on 2.5')
```

```
The total gini of the split for a threshold of 10 is: 0.8119781336196127
This is better than the split on 2.5
```

Back to Top

Entropy based splitting

We discussed entropy as well in the video as another means of splitting. We calculated entropy for a node t by:

$$Entropy(t) = -\sum p(j|t) \log p(j|t)$$

where $p(j|t)$ is the same as above. To combine Entropy measures from a set of nodes, t = {1,...,T} we use:

$$Entropy_{split} = \sum_{t=1}^{T} \frac{n_t}{N} Entropy(t)$$

where n_t and N are the same as defined above for the $Gini$. Information gain is calculated by subtracting the Entropy of the split from the Entropy of the parent node before splitting:

$$InfoGain = Entropy(p) - Entropy_{split}$$

where p is the parent node before splitting. You are given an equation for calculating the $Entropy(t)$ of node t. It works exactly like the `gini_index` function above, but is named `entropy_value` and returns the entropy for a node. You simply send in an array of the feature values for the node you want to calculate the entropy value for.

In [7]:

```
def entropy_value(classes_in_split):
    # pay no attention to this code -- it just computes the gini for a given split
    classes_in_split = np.reshape(classes_in_split,(len(classes_in_split),-1))
    unique_classes = np.unique(classes_in_split)
    ent = 0
    for c in unique_classes:
        p = (np.sum(classes_in_split==c) / float(len(classes_in_split)))
        ent += p * np.log(p)

    return -ent
```

In [8]:

```
ent_r = entropy_value(ds.target[feature28>2.5])
ent_l = entropy_value(ds.target[feature28<=2.5])

ent_r1= entropy_value(ds.target[feature28>T])
ent_l1= entropy_value(ds.target[feature28<=T])


# compute entropy example. This splits on attribute '28' with a value of 2.5
print('entropy for right node of split:', ent_r)
print('entropy for left node of split:', ent_l)
```

```
entropy for right node of split: 2.1836975378213057
entropy for left node of split: 1.4898881412786364
```

Question 4: Calculate the **information gain** of the split when the threshold is 2.5 on `feature28`. What is the value of the information gain?

In [9]:

```
# Enter your code here

#Find parents entropy
parentEntropy = entropy_value(ds.target)
#print(parentEntropy)

other = ((num_instances_r1*ent_r)+(num_instances_l1*ent_l))/(N)
infoGain1 = (parentEntropy)-other



# Enter your code here

print('The information gain of the split for threshold of 2.5:',infoGain1)
```

```
The information gain of the split for threshold of 2.5: 0.27283285132716273
```

Question 5: What is the information gain if the threshold is 10.0 on `feature28`? According to information gain, is it better to split on a threshold of 2.5 or 10? Does entropy give the same decision as gini for this example?

In [10]:

```
# Enter your code here
# Enter your code here

#Find parents entropy
parentEntropy = entropy_value(ds.target)
#print(parentEntropy)

other = ((num_instances_r*ent_r1)+(num_instances_l*ent_l1))/(float(len(ds.target)))
infoGain = (parentEntropy)-other



# Enter your code here

print('The information gain of the split for threshold of 2.5:',infoGain1)



# Enter your code here

print('The information gain of the split for threshold of 10:',infoGain)
print('This is not better than the split on 2.5')
print('This is the same as gini')
```

```
The information gain of the split for threshold of 2.5: 0.27283285132716273
The information gain of the split for threshold of 10: 0.20955137704371163
This is not better than the split on 2.5
This is the same as gini
```

In [ ]:

Back to Top

Information gain and multi-way splitting

Now assume that we can use not just a binary split, but a three way split.

Question 6 What is the information gain if we split feature28 on two thesholds (three separate nodes corresponding to three branches from one node)

- node left: `feature28<2.5`,
- node middle: `2.5<=feature28<10`, and
- node right: `10<=feature28`?

Is the information gain better?

Note: You can index into a `numpy` array for the middle node with the following notation: `some_array[(2.5<=feature28) & (feature28<10.0)]`

In [35]:

```
# Enter your code here
nun_instances_m = len(ds.target)-num_instances_r-num_instances_l
#print(nun_instances_m)
nun_instances_l = len(ds.target)-num_instances_r-nun_instances_m
#print(nun_instances_l)
#print((nun_instances_m))
#Entropy
nodeL=entropy_value(ds.target[(feature28<2.5)])
#print(nodeL)
threeWl= sum(feature28<2.5)
#print(threeWl)
nodeM=entropy_value(ds.target[(2.5<=feature28) & (feature28<10)])
#print(nodeM)
threeWm= sum((feature28>=2.5)& (feature28<10))
#print(threeWm)
nodeR=entropy_value(ds.target[(10<=feature28)])
#print(nodeR)
threeWr= sum(10<=feature28)
#print(threeWr)


tot3way=(((nodeL*threeWl)+(nodeM*threeWm)+(nodeR*threeWr)))/(float(len(ds.target)))
#print(tot3way)
totTot = parentEntropy-tot3way
#print(parentEntropy-tot3way)

# Enter your code here
print('The information gain of the three way split is:',totTot)
```

```
The information gain of the three way split is: 0.3171890999123379
```

In [ ]:

Question 7: Should we normalize the quantity that we just calculated if we want to compare it to the information gain of a binary split? Why or Why not?

In [12]:

```
# Enter your comments here
#Yes, we need to normalize the quantity to compare it to the information gain of a binary split. The information gain prefers larger, purer splits.




# Enter your comments here
```

Back to Top

Decision Trees in scikit-learn

Scikit-learn also has an implementation of decision trees. Its available here:

- http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier

Question 8: What algorithm does scikit-learn use for creating decision trees (i.e., ID3, C4.5, C5.0, CART, MARS, CHAID, etc.)?

In [13]:

```
# Enter your answer here

#scikit-learn uses CART

# Enter your answer here
```

Question 9: Using the documentation, use scikit-learn to train a decision tree on the digits data. Calculate the accuracy on the training data. What is the accuracy? Did you expect the decision tree to have this kind of accuracy? Why or Why not?

In [30]:

```
# use scikit learn to train a decision tree

##Borrowed this block from the course github repo##
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.metrics import accuracy_score

cv = StratifiedShuffleSplit(n_splits=1,train_size=0.5)

dt_clf = DecisionTreeClassifier()

# now get the training and testing
for train, test in cv.split(ds.data,ds.target):
    # train the decision tree algorithm
    dt_clf.fit(ds.data[train],ds.target[train])
    yhat = dt_clf.predict(ds.data[test])
    acc=accuracy_score(ds.target[test],yhat)
    #print ('accuracy:',accuracy_score(ds.target[test],yhat))
print('accuracy:', acc)
print('I did not expect the accuracy to be as high as it is, considering the un-readability of the data when actually visualized.  I guess this shows the power of entropy and the gini and all the other things we covered in t
```

```
accuracy: 0.8075639599555061
I did not expect the accuracy to be as high as it is, considering the un-readability of the data when actually visualized.  I guess this shows the power of entropy and the gini and all the other things we covered in this las
t week.  Plus decision trees are easy to implement. Afer disucssing with Andre, and your comment on "dont be surprised" if its 100% makes me question if 81% is a good score.
```

In [ ]:

Question 10: Look at the other input parameters to the function `DecisionTreeClassifier` could any of them be used to help prevent the decision tree from overlearning on the data?

Which variables might we use to control overfitting and how (explain why it might help to stop overfitting)?

In [34]:

```
# Enter your answer here
#After reading up on the documentation it seems like many of the parameters that stop the tree from growing can help in the case of overfitting.
#I think that min_samples_split is another interesting parameter (defined as: "The minimum number of samples required to split an internal node"), this will make the model more specific and require more "evidence" before spl
#Min_impurity_decrease is also interesting (defined as: "A node will be split if this split induces a decrease of the impurity greater than or equal to this value"))
#This handles splitting the nodes and decreases the impurity, thus making the nodes more pure and gets us closer to the results we desire.
# Enter your answer here
```

That's all! Please **upload your rendered notebook** and please include **team member names** in the notebook submission. Also please remember to save the notebook before uploading.

In [ ]: