COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

Week 12
Machine learning — Decision Trees and Clustering

Applied Data Science

Columbia University - Columbia Engineering

Course Agenda

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

- Week 1: Python Basics: How to Translate Procedures into Codes
- Week 2: Intermediate Python — Data structures for Your Analysis
- Week 3: Relational Databases — Where Big Data is Typically Stored
- Week 4: SQL — Ubiquitous Database Format/Language
- Week 5: Statistical Distributions — The Shape of Data
- Week 6: Sampling — When You Can't or Won't Have ALL the Data
- Week 7:Hypothesis Testing — Answering Questions about Your Data
- Week 8: Data Analysis and Visualization — Using Python's NumPy for Analysis
- Week 9: Data analysis and visualization — Using Python's Pandas for Data Wrangling
- Week 10: Text Mining — Automatic Understanding of Text
- Week 11: Machine learning — Basic Regression and Classification
- **Week 12: Machine learning — Decision Trees and Clustering**

Decision Trees

- Decision trees are tree structures containing rules
- The leaf nodes of the tree are the "learned" categories (or threshold values)
- A path from the root to a leaf node represents a rule

Example: A decision tree with rules on deciding who survived or died on the titanic

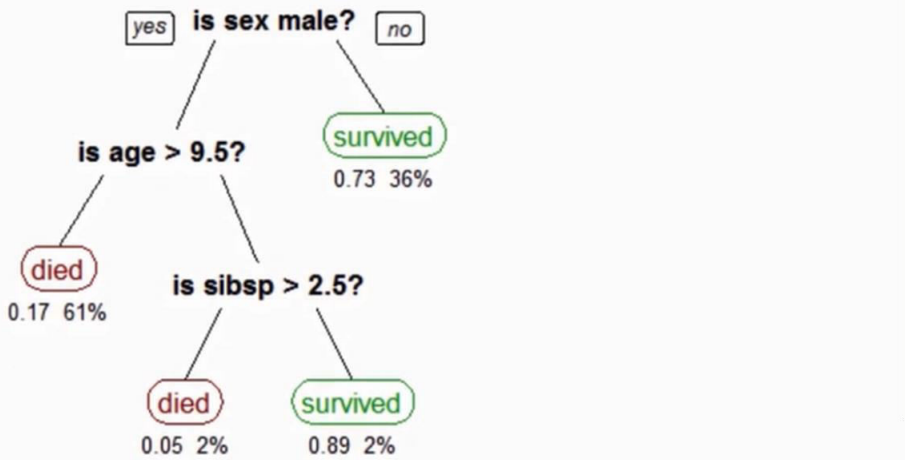
Source: https://en.wikipedia.org/wiki/Decision_tree_learning#/media/File:CART_tree_titanic_survivors.png

- To use the tree, enter a person-data object and you'll get an answer
- Ex: ("John Brown","Male","30 years old", "3 siblings") Ans: Survived (89% probability)
- Ex: ("Jill Jones", "Female","7 years old", "no siblings") Ans: Survived (73% probability)
- Ex: ("Hercules Mulligan", "Male", "2 years old", "20 siblings") Ans: Died (17% probability)
- Note that the 17% probability doesn't mean that there is an 83% chance that Mulligan survived!

Decision Trees


```
In [1]: from IPython.display import Image
        Image(filename = "CART_tree_titanic_survivors.png", width=400, height=400)
```

Out[1]:



Types of Decision Tree

- **Classification trees**: Uses rules to classify cases into two or more categories (Rocks vs Mines)
  - Classification trees recursively split the data on a feature value
  - Each split minimizes the cost (also known as the impurity)
  - Cost is commonly measured using the GINI cost function (a measure of the probability of misclassification or 'purity')
- **Regression trees**: Uses rules to group data into target variable ranges (Wine Quality)
  - Also split the data on feature values
  - Minimize cost (impurity). Usually the mean squared error

Predicting Wine Quality

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

Import the data

In [ ]:
```
url = "http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv"
import pandas as pd
from pandas import DataFrame
w_df = pd.read_csv(url,header=0,sep=';')
w_df.describe()
```

Build train and test samples

In [ ]:
```
from sklearn.model_selection import train_test_split
train, test = train_test_split(w_df, test_size = 0.3)
x_train = train.iloc[0:,0:11]
y_train = train[['quality']]
x_test = test.iloc[0:,0:11]
y_test = test[['quality']]

#Use all data for cross validation
x_data = w_df.iloc[0:,0:11]
y_data = w_df[['quality']]
#x_data
y_test
```

Classifier Vs. Regressor

Classifiers vs Regressors

- Decision tree regressors are used when the target variable is continuous and ordered (wine quality from 0 to 10)
- Classifiers are used when the target variable is a set of unordered categories (rocks or mines)

For wine quality, we need a regressor

In [4]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn import tree

model = DecisionTreeRegressor(max_depth = 3)
model.fit(x_train,y_train)
```

Out[4]:
```
DecisionTreeRegressor(criterion='mse', max_depth=3, max_features=None,
           max_leaf_nodes=None, min_impurity_split=1e-07,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, presort=False, random_state=None,
           splitter='best')
```

Details: http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

In [5]:
```
#Get the R-Square for the predicted vs actuals on the text sample
print("Training R-Square",model.score(x_train,y_train))
print("Testing R-Square",model.score(x_test,y_test))
```

```
Training R-Square 0.352621759288
Testing R-Square 0.281274761523
```

Classifier Vs. Regressor

Download and install graphviz https://graphviz.gitlab.io/download/

If you are having issues using Graphviz in Windows, then try the following steps:

1. 1. Install Graphviz
2. 2. After installing graphviz, add it to the Computer's Path.
   - Go to PC properties
   - Click environment variables in the advanced settings section
   - Add C:\Program Files (x86)\Graphviz2.38\bin\ to the PATH and click Apply
3. Install Pydotplus. Note that you will always have to install pydot after graphviz as Pydot is Graphviz's dot language and needs Graphviz for reference

Install pydotplus (using pip): Install graphviz before you install pydotplus!

In [ ]:
```
#!pip install pydotplus
```

In [ ]:
```
import pydotplus
feature_names = [key for key in w_df]
dot_data = tree.export_graphviz(model.tree_, out_file=None,feature_names=feature_names)
graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_pdf("wines.pdf")
#The tree will be saved to wines.pdf in your current directory
```

Entropy Minimizers

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

Decision trees are Entropy minimizers

- **Entropy**: a measure of uncertainty in the data

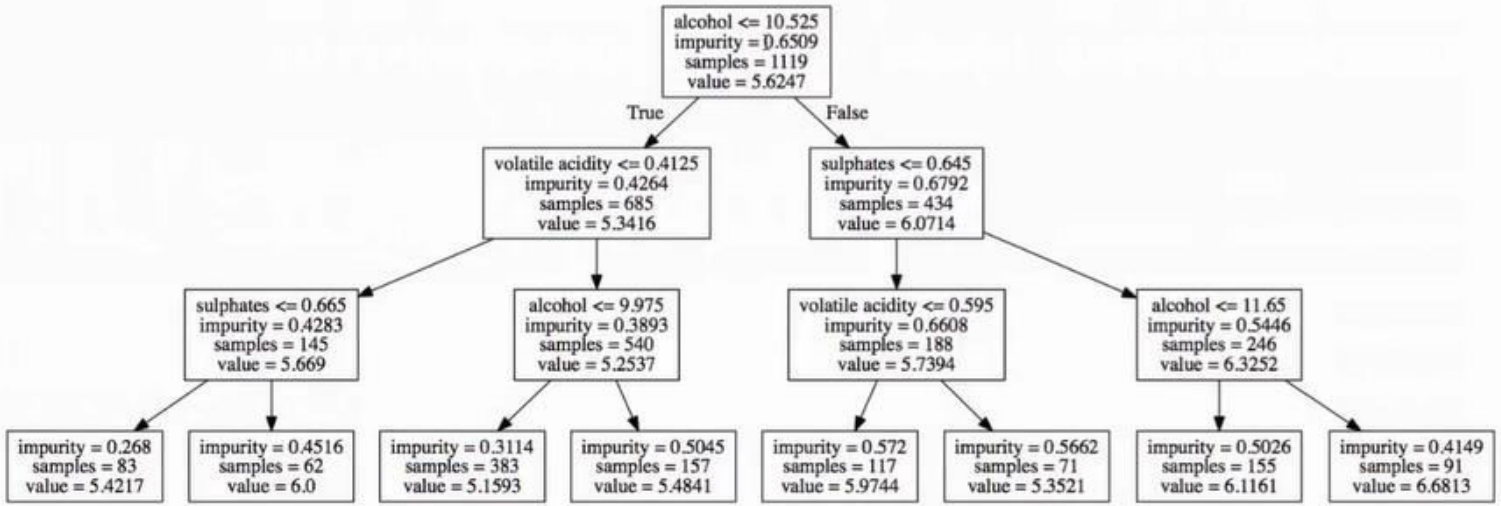
what is the uncertainty in color when you draw a marble from a box of 100 blue marbles?

what is the uncertainty when you draw a marble from a box with 50 blue and 50 red marbles?

- Entropy minimization: decision tree algorithms seek to partition the data on features in the way that total entropy is minimized

Regression trees

- Run regressions for each X to the dependent variable
- Pick the variable with the most explanatory power and split it at several points
- Calculate the Mean Square Error of each of the two halves for each split
- Pick the split point that gives the lowest mse (combined)



Cross Validation

Cross validation is required to understand how robust is the model.

- Split the training set into k smaller sets (aka folds)
- Train the data on k-1 folds
- Validate the results on fold k
- Repeat this holding out each of the k folds in turn
- Report the average of all tests as the performance metric
- http://scikit-learn.org/stable/modules/cross_validation.html

Purpose of cross-validation

- Not to generate a tree (it generates many trees!)
- But to provide an estimate of the average error of the model
- Roughly, the idea is to see how the model performance varies with different training sets
- To generate the tree, use the entire training set as before

In [8]:
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
```

In [9]:
```
#from sklearn.cross_validation import cross_val_score
#from sklearn.cross_validation import KFold
crossvalidation = KFold(n_splits=5,shuffle=True, random_state=1)
```

In [10]:
```
from sklearn import tree
import numpy as np
for depth in range(1,10):
    model = tree.DecisionTreeRegressor(
    max_depth=depth, random_state=0)
    if model.fit(x_data,y_data).tree_.max_depth < depth:
        break
    score = np.mean(cross_val_score(model, x_data, y_data,scoring='neg_mean_squared_error', cv=crossvalidation, n_jobs=
    print ('Depth: %i Accuracy: %.3f' % (depth,score))
```

```
Depth: 1 Accuracy: -0.548
Depth: 2 Accuracy: -0.512
Depth: 3 Accuracy: -0.482
Depth: 4 Accuracy: -0.482
Depth: 5 Accuracy: -0.480
Depth: 6 Accuracy: -0.493
Depth: 7 Accuracy: -0.535
Depth: 8 Accuracy: -0.573
Depth: 9 Accuracy: -0.600
```

Classification Tree

Classification trees are used when dealing with categorical dependent variables

- Pick a variable and a split point so that the misclassification cost is the lowest.

Rocks and Mines Data Set

In [11]:
```
import pandas as pd
from pandas import DataFrame
url="https://archive.ics.uci.edu/ml/machine-learning-databases/undocumented/connectionist-bench/sonar/sonar.all-data"
df = pd.read_csv(url,header=None)
df[60]=np.where(df[60]=='R',0,1)
df.describe()
```

In [12]:
```
from sklearn.model_selection import train_test_split
train, test = train_test_split(df, test_size = 0.3)
x_train = train.iloc[0:,0:60]
y_train = train[[60]]
x_test = test.iloc[0:,0:60]
y_test = test[[60]]
y_train
```

Out[12]:

	60
78	0
133	1
12	0
163	1
95	0

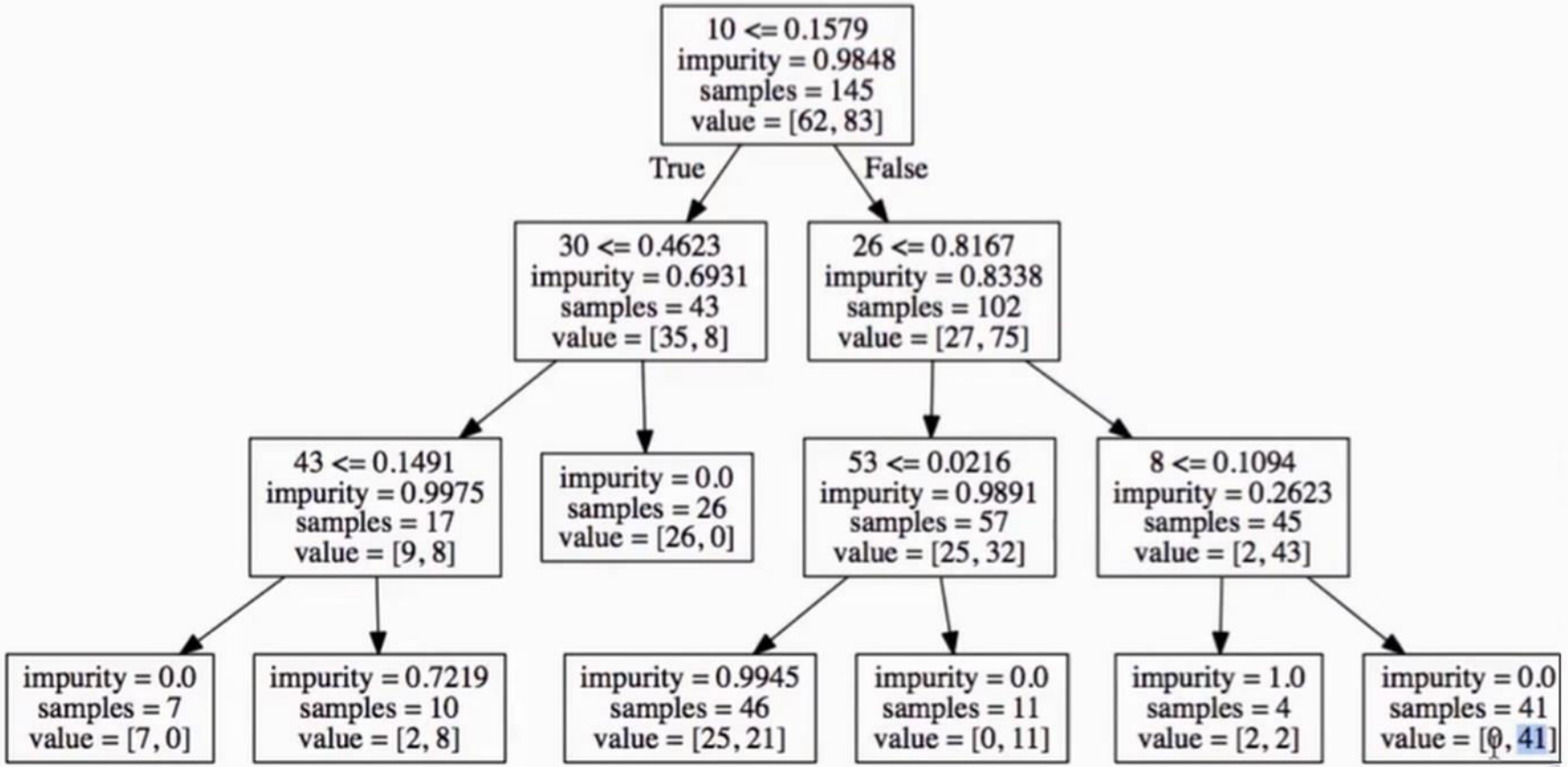
In [13]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

model = DecisionTreeClassifier(max_depth = 3,criterion='entropy')
model.fit(x_train,y_train)
```

Out[13]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=3,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
```

In [14]:
```
import pydotplus
feature_names = [key for key in df]
dot_data = tree.export_graphviz(model.tree_, out_file=None,feature_names=feature_names)
graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_pdf("mines.pdf")
#The tree will be saved to mines.pdf in your current directory
```

Out[14]: True

Classification Tree



Confusion Matrix

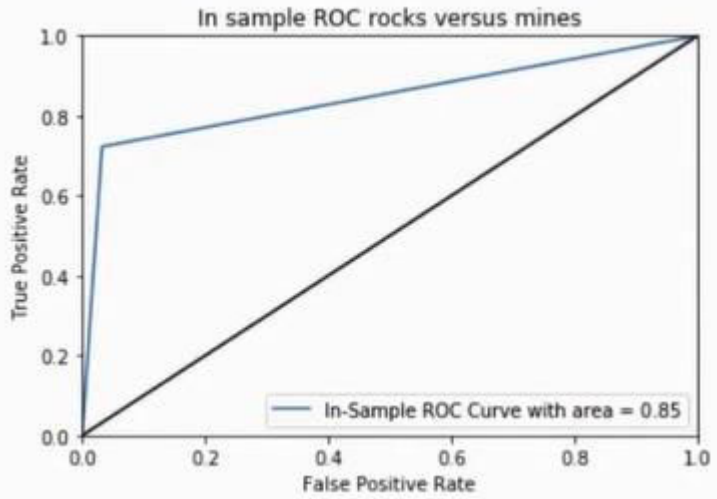
COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

In [15]:
```
def confusion_matrix(predicted, actual, threshold):
    if len(predicted) != len(actual): return -1
    tp = 0.0
    fp = 0.0
    tn = 0.0
    fn = 0.0
    for i in range(len(actual)):
        if actual[i] > 0.5: #labels that are 1.0  (positive examples)
            if predicted[i] > threshold:
                tp += 1.0 #correctly predicted positive
            else:
                fn += 1.0 #incorrectly predicted negative
        else:              #labels that are 0.0 (negative examples)
            if predicted[i] < threshold:
                tn += 1.0 #correctly predicted negative
            else:
                fp += 1.0 #incorrectly predicted positive
    rtn = [tp, fn, fp, tn]

    return rtn
```

In [16]:
```
p_train=model.predict(x_train)
p_test = model.predict(x_test)
print(confusion_matrix(p_train,np.array(y_train),.5))
print(confusion_matrix(p_test,np.array(y_test),.5))
```

```
[60.0, 23.0, 2.0, 60.0]
[18.0, 10.0, 3.0, 32.0]
```

In [17]:
```
from sklearn.metrics import roc_curve,auc
import pylab as pl
%matplotlib inline
(fpr, tpr, thresholds) = roc_curve(y_train,p_train)
area = auc(fpr,tpr)
pl.clf() #Clear the current figure
pl.plot(fpr,tpr,label="In-Sample ROC Curve with area = %1.2f"%area)

pl.plot([0, 1], [0, 1], 'k') #This plots the random (equal probability line)
pl.xlim([0.0, 1.0])
pl.ylim([0.0, 1.0])
pl.xlabel('False Positive Rate')
pl.ylabel('True Positive Rate')
pl.title('In sample ROC rocks versus mines')
pl.legend(loc="lower right")
pl.show()
```

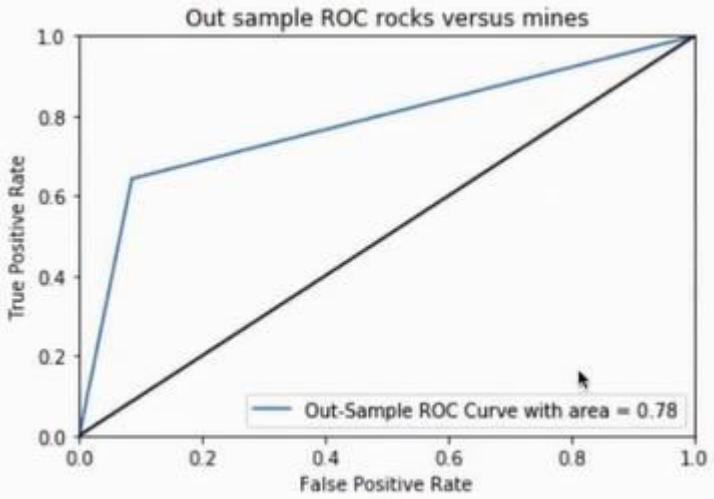


In [18]:
```
fpr,tpr,thresholds
```

Out[18]:
```
(array([ 0.        ,  0.03225806,  1.        ]),
 array([ 0.        ,  0.72289157,  1.        ]),
 array([2, 1, 0]))
```

Confusion Matrix


```
In [19]: from sklearn.metrics import roc_curve,auc
         import pylab as pl
         %matplotlib inline
         (fpr, tpr, thresholds) = roc_curve(y_test,p_test)
         area = auc(fpr,tpr)
         pl.clf() #Clear the current figure
         pl.plot(fpr,tpr,label="Out-Sample ROC Curve with area = %1.2f"%area)

         pl.plot([0, 1], [0, 1], 'k') #This plots the random (equal probability line)
         pl.xlim([0.0, 1.0])
         pl.ylim([0.0, 1.0])
         pl.xlabel('False Positive Rate')
         pl.ylabel('True Positive Rate')
         pl.title('Out sample ROC rocks versus mines')
         pl.legend(loc="lower right")
         pl.show()
```



We can pick a model by looking at the threshold, the cost, and decide which methodology is better. Cross validation can be helpful to identify other methods to improve decision tree's results.

Clustering Algorithms

Unsupervised learning

- The algorithm tries to group similar data together (clusters)
- Using the values of the feature space

K-Means Clusterng

- partitions the dataspace into clusters
- minimizes distance between the mean of a cluster and the data points
- the desired number of clusters must be known in advance

Image recognition dataset

- Digits 0-9 pixelated into 64 quadrants
- Each value represents the area that is shaded

Do imports

```
In [2]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
        from sklearn.datasets import load_digits
        from sklearn.preprocessing import scale
```

Load data

```
In [3]: digits = load_digits()
        digits
```

```
Out[3]: {'DESCR': "Optical Recognition of Handwritten
        ======================\n\nNotes\n-----\nData
        s: 5620\n    :Number of Attributes: 64\n    :
        xels in the range 0..16.\n    :Missing Attrib
        lpaydin '@' boun.edu.tr)\n    :Date: July: 19
```

scale the data to normal distribution

```
In [4]: data = scale(digits.data)
```

```
In [5]: data
```

```
Out[5]: array([[ 0.        , -0.33501649, -0.04308102, ..., -1.14664746,
                -0.5056698 , -0.19600752],
               [ 0.        , -0.33501649, -1.09493684, ...,  0.54856067,
                -0.5056698 , -0.19600752],
               [ 0.        , -0.33501649, -1.09493684, ...,  1.56568555,
                 1.6951369 , -0.19600752],
               ...,
               [ 0.        , -0.33501649, -0.88456568, ..., -0.12952258,
                -0.5056698 , -0.19600752],
               [ 0.        , -0.33501649, -0.67419451, ...,  0.8876023 ,
                -0.5056698 , -0.19600752],
               [ 0.        , -0.33501649,  1.00877481, ...,  0.8876023 ,
                -0.26113572, -0.19600752]])
```

Render Digit Images

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

Render the digit images and their associated values

In [6]:
```
def print_digits(images,y,max_n=10):
    # set up the figure size in inches
    fig = plt.figure(figsize=(12, 12))
    fig.subplots_adjust(left=0, right=1, bottom=0, top=1,
            hspace=.05, wspace=.5)
    i = 0
    while i <max_n and i <images.shape[0]:
        # plot the images in a matrix of 20x20
        p = fig.add_subplot(20, 20, i + 1, xticks=[],
              yticks=[])
        p.imshow(images[i], cmap=plt.cm.bone)
        # label the image with the target value
        p.text(0, 14, str(y[i]))
        i = i + 1
print_digits(digits.images, digits.target, max_n=10)
```

Training and Testing Samples

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

In [9]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test, images_train,images_test = train_test_split(
        data, digits.target, digits.images,  test_size=0.25,
          random_state=42)

n_samples, n_features = X_train.shape
n_digits = len(np.unique(y_train))
labels = y_train
labels
```

Out[9]:
```
array([5, 2, 0, ..., 2, 7, 1])
```

In [10]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test, images_train,images_test = train_test_split(
        data, digits.target, digits.images,  test_size=0.25,
          random_state=42)

n_samples, n_features = X_train.shape
n_digits = len(np.unique(y_train))
labels = y_train
X_train
```

Out[10]:
```
array([[ 0.        , -0.33501649, -0.67419451, ..., -1.14664746,
        -0.5056698 , -0.19600752],
       [ 0.        ,  5.17802955,  2.2710018 , ..., -0.12952258,
        -0.26113572, -0.19600752],
       [ 0.        , -0.33501649, -0.25345218, ..., -0.80760583,
```

In [11]:
```
len(np.unique(y_train))
```

Out[11]: 10

Create the model and fit the data

In [12]:
```
from sklearn import cluster
clf = cluster.KMeans(init='k-means++',n_clusters=10, random_state=42)
clf.fit(X_train)
```

Out[12]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=10, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=42, tol=0.0001, verbose=0)
```

k-means++ runs an initializer before using the k-means algorithm

In [ ]:
```
images_train
```

Return Labels

Returned labels are cluster numbers

```
In [14]: print_digits(images_train, clf.labels_, max_n=20)
```

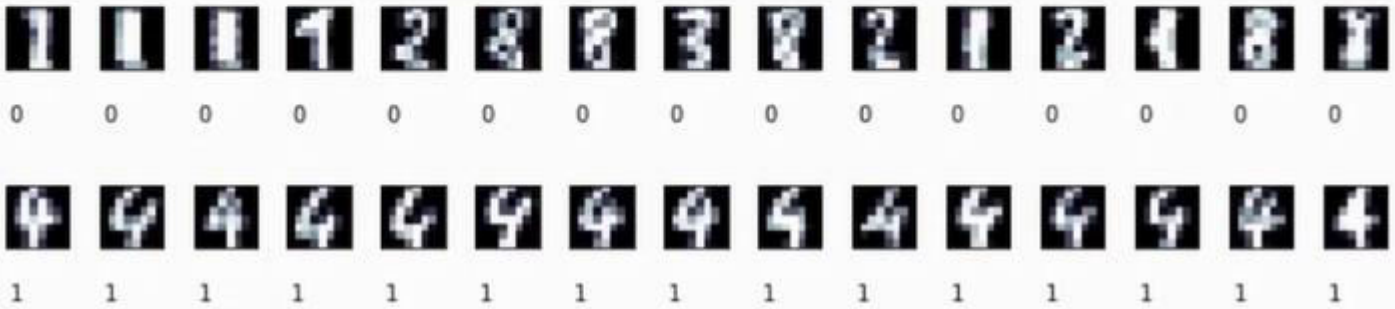
Use test sample to generate predictions

```
In [16]: y_pred=clf.predict(X_test)
         y_pred
```

```
Out[16]: array([4, 8, 8, 9, 3, 3, 5, 8, 5, 3, 0, 7, 1, 2, 1, 3, 8, 6, 8, 8, 1, 5, 8,
                6, 5, 4, 8, 5, 4, 8, 1, 8, 3, 1, 1, 4, 8, 1, 6, 4, 4, 8, 0, 8, 4, 7,
                8, 2, 4, 5, 5, 0, 8, 5, 4, 2, 8, 2, 2, 7, 2, 1, 5, 3, 1, 5, 6, 2, 6,
                8, 8, 8, 8, 6, 6, 2, 1, 5, 8, 8, 8, 2, 3, 8, 8, 2, 4, 1, 1, 8, 0, 3,
                7, 8, 8, 3, 8, 2, 1, 1, 1, 8, 5, 8, 7, 6, 5, 7, 1, 3, 6, 6, 1, 1, 7,
                8, 0, 6, 8, 6, 3, 4, 8, 7, 2, 6, 3, 6, 5, 0, 5, 5, 6, 7, 2, 4, 4, 1,
                3, 0, 2, 8, 1, 4, 8, 7, 4, 8, 2, 8, 8, 4, 4, 2, 4, 1, 3, 8, 8, 6, 6,
                3, 8, 2, 1, 5, 0, 4, 5, 6, 8, 0, 1, 0, 0, 8, 6, 6, 3, 3, 8, 8, 8, 2,
                8, 0, 3, 5, 4, 8, 8, 1, 0, 5, 9, 3, 8, 4, 1, 0, 8, 8, 5, 6, 0, 8, 1,
                0, 0, 5, 1, 1, 8, 4, 0, 8, 4, 2, 1, 5, 3, 6, 7, 4, 1, 5, 4, 2, 8, 3,
                8, 4, 6, 7, 8, 3, 1, 6, 4, 5, 0, 8, 5, 0, 2, 3, 0, 0, 6, 5, 6, 4, 3,
```

Return Labels


```
In [*]: def print_cluster(images, y_pred, cluster_number):
            images = images[y_pred==cluster_number]
            y_pred = y_pred[y_pred==cluster_number]
            print_digits(images, y_pred,max_n=15)
        for i in range(10):
              print_cluster(images_test, y_pred, i)
```



Evaluating the model

- Adjusted rand index: A measure of the similarity between two groups
- We'll use it to see how similar the y_test actuals and predicted groupings are
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html
- 0.0 indicates that there is no similarity and any overlap is explainable as totally random
- 1.0 indicates that the two groups are identical

In [18]:
```
from sklearn import metrics
print("Adjusted rand score: {0:2}".format(metrics.adjusted_rand_score(y_test, y_pred)))
```

```
Adjusted rand score: 0.5674467844660916
```

Confusion matrix

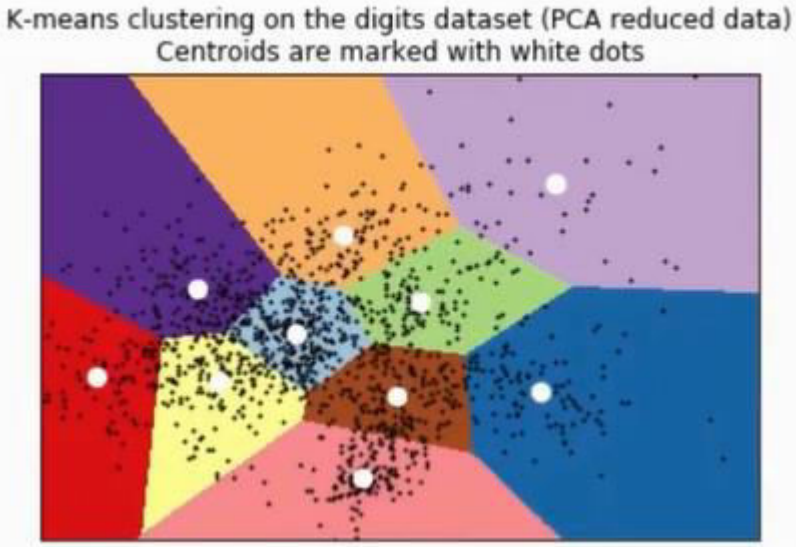
- Each row corresponds to a number (y_test)
- Each column to y_pred (the cluster number)
- Data is the number of times y_test was assigned to the corresponding y_pred
- For example, 0 is fully assigned to cluster 2 (Row 0, Column 2)
- 8 is assigned to cluster 0 21 times (Row 8, Column 0)
- 7, which is cluster 6 is assigned to cluster 6 34 times (Row 7, Column 6)

```
In [19]: print(metrics.confusion_matrix(y_test, y_pred))

[[ 0  0 43  0  0  0  0  0  0  0]
 [20  0  0  7  0  0  0 10  0  0]
 [ 5  0  0 31  0  0  0  1  1  0]
 [ 1  0  0  1  0  1  4  0 39  0]
 [ 1 50  0  0  0  0  1  2  0  1]
 [ 1  0  0  0  1 41  0  0 16  0]
 [ 0  0  1  0 44  0  0  0  0  0]
 [ 0  0  0  0  0  1 34  1  0  5]
 [21  0  0  0  0  3  1  2 11  0]
 [ 0  0  0  0  0  2  3  3 40  0]]
```

Graphical Views of the Clusters

- First reduce the x dimensions to 2 using principle component analysis
- https://en.wikipedia.org/wiki/Principal_componenent_analysis
- Then figure out the range of values and define the grid
- Run k-means on the reduced (2 component) data set
- Draw a color map and plot the pca points on this map
- Find the cluster centroids and plot them on the color map

In [ ]:
```
from sklearn import decomposition
pca = decomposition.PCA(n_components=2).fit(X_train)
reduced_X_train = pca.transform(X_train)
# Step size of the mesh.
h = .01
# point in the mesh [x_min, m_max]x[y_min, y_max].
x_min, x_max = reduced_X_train[:, 0].min() + 1, reduced_X_train[:, 0].max() - 1
y_min, y_max = reduced_X_train[:, 1].min() + 1, reduced_X_train[:, 1].max() - 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
    np.arange(y_min, y_max, h))
kmeans = cluster.KMeans(init='k-means++', n_clusters=n_digits,
    n_init=10)
kmeans.fit(reduced_X_train)
Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])
# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1)
plt.clf()
plt.imshow(Z, interpolation='nearest', extent=(xx.min(), xx.max(), yy.min(), yy.max
plt.plot(reduced_X_train[:, 0], reduced_X_train[:, 1], 'k.',
    markersize=2)
# Plot the centroids as a white X
centroids = kmeans.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1],marker='.',
    s=169, linewidths=3, color='w', zorder=10)
plt.title('K-means clustering on the digits dataset (PCA reduced data)\nCentroids a
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
plt.show()
```



www.emeritus.org