Machine Learning

Mitra Bokaei, Ph.D.

Assistant Professor of Computer Science

St. Mary's University

https://github.com/pfv610/ML-camp

ST. MARY'S UNIVERSITY

Review

Numpy Arrays

Pandas

DataFrames

Tuples

ST. MARY'S UNIVERSITY

Numpy Arrays

- Alternatives to Python Lists
- Perform calculations across the entire array instead of applying the calculations on every single element

- Let's create two separate lists for height and weight
- Create a third list containing the BMI for each element of height and weight

Tools

- Anaconda
  - A popular platform for data science and machine learning
- Jupyter Notebook
  - An interactive application that allows us to write, edit, and compile code through a web browser

BMI Calculation using Lists

```
height = [1.87, 1.87, 1.82, 1.91, 1.90, 1.85]
weight = [81.65, 97.52, 95.25, 92.98, 86.18, 88.45]

bmi = []

i = 0
while i < len(height):
        bmi.append(weight[i] /height[i] ** 2)
        print(bmi[i])
        i = i+1
```

BMI Calculation using Numpy Array

```
height = [1.87, 1.87, 1.82, 1.91, 1.90, 1.85]
weight = [81.65, 97.52, 95.25, 92.98, 86.18, 88.45]

# Import the numpy package as np
import numpy as np


np_height = np.array(height)
np_weight = np.array(weight)

# Calculate bmi
bmi = np_weight / np_height ** 2

# Print the result
print(bmi)
```

Element-wise Calculations using Numpy

- We calculated the BMI for six observations of height and weight only in one single statement.
- Imagine that you have 1000 observations of height and weight!
- This way your program faster and efficient!

ST. MARY'S UNIVERSITY

Sub-setting

- Which observations in our BMI array are related to underweight people?
- **If your BMI is less than 18.5**, it falls within the underweight range.
- **If your BMI is 18.5 to 24.9**, it falls within the normal or Healthy Weight range.
- **If your BMI is 25.0 to 29.9**, it falls within the overweight range.
- **If your BMI is 30.0 or higher**, it falls within the obese range.

Sub-setting Example Code

```
height = [1.87, 1.87, 1.82, 1.91, 1.90, 1.85]
weight = [81.65, 97.52, 95.25, 92.98, 86.18, 88.45]

import numpy as np

np_height = np.array(height)
np_weight = np.array(weight)

bmi = np_weight / np_height ** 2

print(bmi)

print (bmi[bmi < 25])
print (bmi[( bmi > 18.5) & (bmi <24.9)])
```

Exercise

- Ask ten of your classmates for their names, weight in pound, and height inches.
- Create three lists for names, weight, and height
- Create three Numpy arrays from name, weight, and height lists
- Transform weights from pound to kilogram (divide the mass value by 2.205)
- Transform height from inches to meters (divide the length value by 39.37)
- Print the name Numpy array
- Calculate and print the BMI

ST. MARY'S UNIVERSITY

Panda DataFrames

- Pandas is a high-level data manipulation tool
- It is built on the Numpy package and its key data structure is called the DataFrame.
- DataFrames allow you to store and manipulate tabular data in rows of observations and columns of variables.

	All Tests Performed**		Bexar County***	
Test Results*	Number	Percent	Number	Percent
Negative	131,421	91.6%	108,414	91.2%
Positive	11,834	8.2%	10,296	8.7%
Inconclusive	316	0.2%	101	0.1%
Total Tested	143,571	100.0%	118,811	100.0%

Posted on 6/24/2020 at 7:00 PM; Based on available data.

Creating Pandas

- There are several ways to create a DataFrame. One way way is to use a dictionary. For example:

```
dict = {"country": ["Brazil", "Russia", "India", "China", "South Africa"],
        "capital": ["Brasilia", "Moscow", "New Dehli", "Beijing", "Pretoria"],
         "population": [200.4, 143.5, 1252, 1357, 52.98] }

import pandas as pd
brics = pd.DataFrame(dict)
print(brics)
```

Creating Pandas – Cont.

- Another way to create a DataFrame is by importing a csv file using Pandas.
- https://www.learnpython.org/en/Pandas_Basics

```
# Import pandas as pd
import pandas as pd
# Import the cars.csv data: cars
cars = pd.read_csv('cars.csv')
# Print out cars
print(cars)
```

ST. MARY'S UNIVERSITY

Script Output

	Unnamed:0	Cars_per_cap	Country	Drives_right
0	US	809	United States	True
1	AUS	731	Australia	False
2	JAP	588	Japan	False
3	IN	18	India	False
4	RU	200	Russia	True
5	MOR	70	Morocco	True
6	EG	45	Egypt	True

Indexing DataFrames

- To extract specific columns, we can use square brackets as series or DataFrames (https://www.learnpython.org/en/Pandas_Basics)

```
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)

# Print out country column as Pandas Series
print(cars['cars_per_cap'])

# Print out country column as Pandas DataFrame
print(cars[['cars_per_cap']])

# Print out DataFrame with country and drives_right columns
print(cars[['cars_per_cap', 'country']])
```

Retrieving Rows from DataFrames

- Square brackets can also be used to access observations (rows) from a DataFrame. For example: (https://www.learnpython.org/en/Pandas_Basics)

```
# Import cars data
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)

# Print out first 4 observations
print(cars[0:4])

# Print out fifth and sixth observation
print(cars[4:6])
```

Function Zip() on Tuples

- **Tuple** is an immutable (unchangeable) collection of elements of different data types.
- It is an ordered collection, so it preserves the order of elements in which they were defined.
- **Tuples** are defined by enclosing elements in parentheses () , separated by a comma.
- The zip() function returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together etc.
- If the passed iterators have different lengths, the iterator with the least items decides the length of the new iterator.

Function Zip() on Tuples - Example

```
a = ("John", "Charles", "Mike")
b = ("Jenny", "Christy", "Monica")

x = zip(a, b)

#use the tuple() function to display a readable version of the result:

print(tuple(x))
```

ST. MARY'S UNIVERSITY

Machine Learning Introduction

What is programming?

- Breaking down a task into a series of steps that can be followed
- Learning to program is
  - Trial & error
  - Always experimental
  - Like learning a language
    - Building a vocabulary

What is machine learning?

- Machine learning involves building mathematical models to help understand data.
- Creating and using models that are learned from data.

Modeling

- What is a model?
- A specification of a mathematical (or probabilistic)relationship that exists between different variables.
- Example:

1. if you're trying to raise money for your social networking site,
   - Build a business model (likely in a spreadsheet) that takes inputs like:
     - "number of users"
     - "ad revenue per user"
     - "number of employees"
     - outputs the annual profit for the next several years.
2. A cookbook recipe entails a model that relates inputs like "number of eaters" and "hungriness" to quantities of ingredients needed.

Modeling (Cont.)

- The business model is probably based on simple mathematical relationships:
- profit is revenue minus expenses, revenue is units sold times average price, and so on.
- The recipe model is probably based on trial and error — someone went in a kitchen and tried different combinations of ingredients until they found one they liked.

Machine Learning' Goal

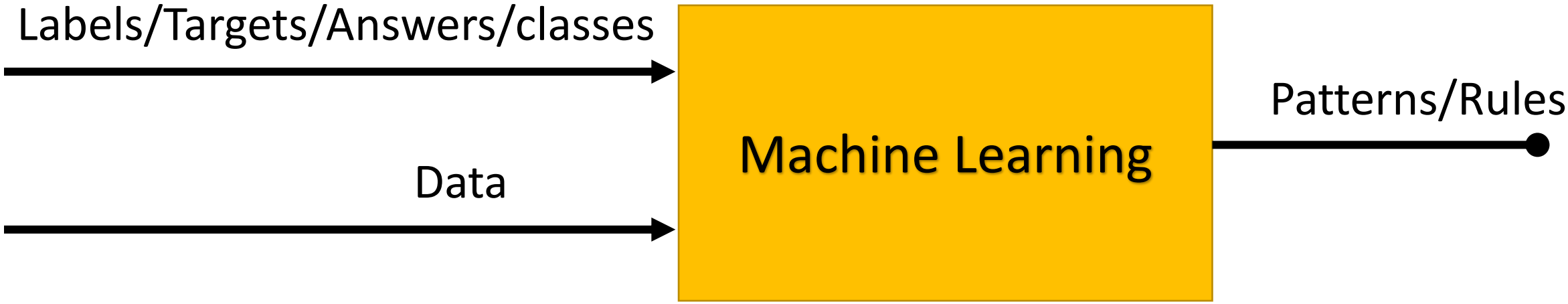
- To use existing data to develop models that we can use to predict various outcomes for new data
  - Predicting whether an email message is spam or not
  - Predicting whether a credit card transaction is fraudulent
  - Predicting which advertisement a shopper is most likely to click on
  - Predicting which football team is going to win the Super Bowl

Now can you give me some examples?

- What are different devices that you deal with during the day that are using machine learning?
  - Google Maps (maps in general)
  - Uber, Lyft
  - Gmail and email categorization
  - Siri, Alexa, Google home assistant
  - Tesla cars (self driving cars in general)
  - Netflix recommendation system
  - Google Translate– trained on examples of documents that have been manually translated.
  - The auto-suggest on my phone keyboard, that suggests what word I might want to write next – trained on examples of what I've typed before.
  - Credit card fraud detection – trained on my buying patterns to recognize a purchase that might not actually be me.
  - Disease Diagnosis

Review - Traditional Programming

Review – Machine Learning



ST. MARY'S UNIVERSITY

Spam Detector

Greetings to you my friend,

I know this will come to you as a surprise because you do not know me.
I am John Alison I work in Central Bank of Nigeria, packaging and courier department.

I got your contact among others from a search on the internet and I was inspired to seek your co-operation, I want you to help me clear this consignment that is already in theEurope which I shipped through our CBN accredited courier agent. The content of the package is $20,000,000.00 all in $100 bills, but the courier company does not know that the consignment contains money.

All I want you to do for me now is to give me your mailing address, your private phone and fax number, and I believe that at the end of the day you will have 50% and 50% will be for me. My identity must not be revealed to anybody.

If this arrangement is okay by you, you can call

Phone: +234 8028776685
Email:john_alison444@yahoo.com

ST. MARY'S UNIVERSITY

Nigerian Prince Scam Email

- Also called "**Nigerian** letter" **scams** or "foreign money exchanges," these typically start with an **email** from someone overseas who claims to be royalty.
- The fraudsters lure you in by offering a share of a huge investment opportunity or a fortune they can't get out of the country without your help.
- While it's closely associated with the early internet, the **Nigerian prince** first went global in the 1980s when West **African** fraudsters began snail-**mailing scam** letters around the world.

ST. MARY'S UNIVERSITY

Spam Detector – Cont.

Journal of Geography and Regional Planning Vol. 2(5), pp. 121-130, May, 2009
Available online at http://www.academicjournals.org/JGRP ISSN 2070-1845

Full Length Research Paper

Mapping as a complement of statistical/graphical approach to climate data analysis: Example from rainfall variability over the western Nigeria

A. O. Eludoyin[1]*, O. M. Akinbode[2], M. A. Oyinloye[3] and A. Fadejin[4]

[1]Department of Geography, Obafemi Awolowo University, Ile-Ife, Nigeria.
[2]Department of Geography and Planning Sciences, Adekunle Ajasin University, Akungba-Akoko, Nigeria.
[3]Department of Urban and Regional Planning, Federal University of Technology, Akure, Nigeria.
[4]Department of Geography, University of Ibadan, Nigeria.

Accepted 17 March, 2009

The challenge of insufficient network of meteorological data collection centre has been an age old problem in Africa. It is therefore necessary to encourage a technique that will ensure that data are generated for areas without them. This study is in this regard. 20 years rainfall data were collected from 5 stations in the Western region of Nigeria and analyses were carried out to detect the monthly variations in 2 decades (1985 – 1995 and 1995 – 2004). The study made use of geographic information system (GIS)'s mapping procedure and graphical methods to illustrate that mapping effectively complements the graphical/statistical method, apart from showing spatial distribution, values can be generated for areas without such data. The various maps can also be overlaid in a GIS environment to determine causes and effect of what is being studied among others. The study concluded that mapping could easily complement statistical/graphical approach to studies in spatial science, especially in modeling and data management.

Key words: GIS mapping, monthly and spatial variability, climate data, western Nigeria.

INTRODUCTION

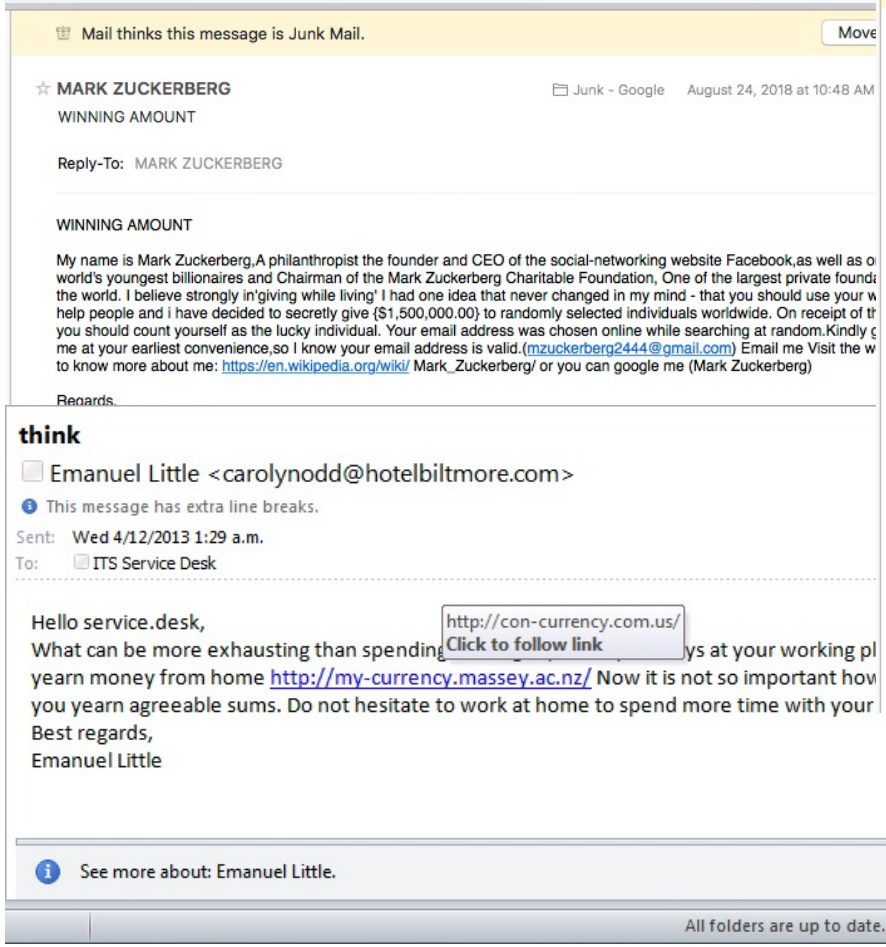
A spatial science typically seeks to describe and interpret the spatial objects from place to place of the earth as the world of man. A spatial object is defined as an object that is characterized by location (longitude, latitude and altitude) and attribute. The object could be represented as point, chains of points (lines) or polygon, depending on its form. This is the technique that has informed the use of geospatial analysis in geographical studies. In climate study, the problem of insufficient meteorological data has been recognized as an old age problem in Africa (Jagtap, 2008). This had probably been informed by the poor network of meteorological stations in most of the African countries. A way of solving this problem is to use the mapping technique inherent in geographical information system (GIS). A GIS is a computerized technique that involves data acquisition, storage, manipulation, analysis and information presentation aimed at solving an earth bound problem. GIS technology is not synonymous to automated mapping (AM) because it involves the creation of digital database that can be updated, analysed and synthesized to produce user required results (Burrough, 1985).

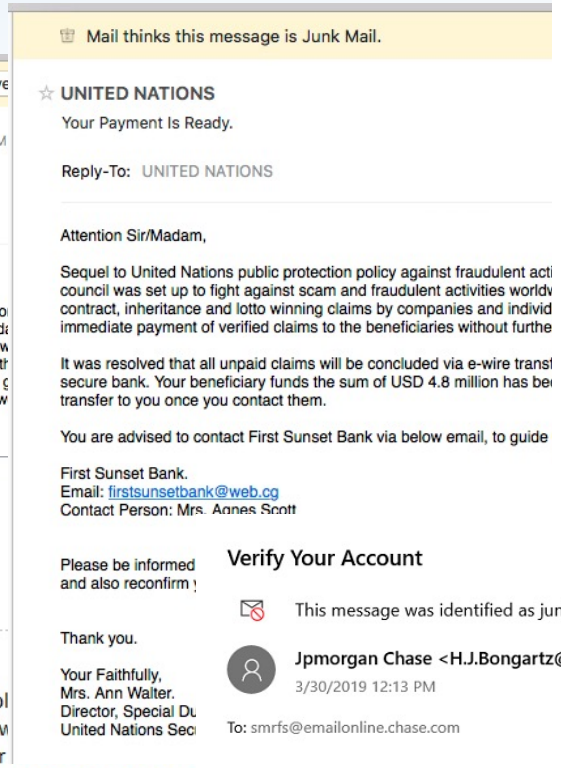
Mapping, in whatever form, usually involves interpolation which is commonly used in computer assisted mapping (CAM) and which assumed that the unknown value of a point is influenced more by nearer points than those farther away. This is quite important when the phenomena being studied (for example, rainfall as in this case) is continuous in space and are measured at points in space (at synoptic stations). Since the existence of such point is always limited in number for financial and other reasons, the mapping technique that uses interpolation becomes expedient. Statistical analyses on the other hands typically include the use of graphs and various parametric and non parametric tests. Whereas non parametric statistical tests such as Chi square, Kolmogorov–Smirnov tests, etc. derive statistics from ordinal or nominal (qualitative data) (Taylor, 1977) while parametric statistics require data that are of interval and ratio (quantitative) format. Data for parametric statistics

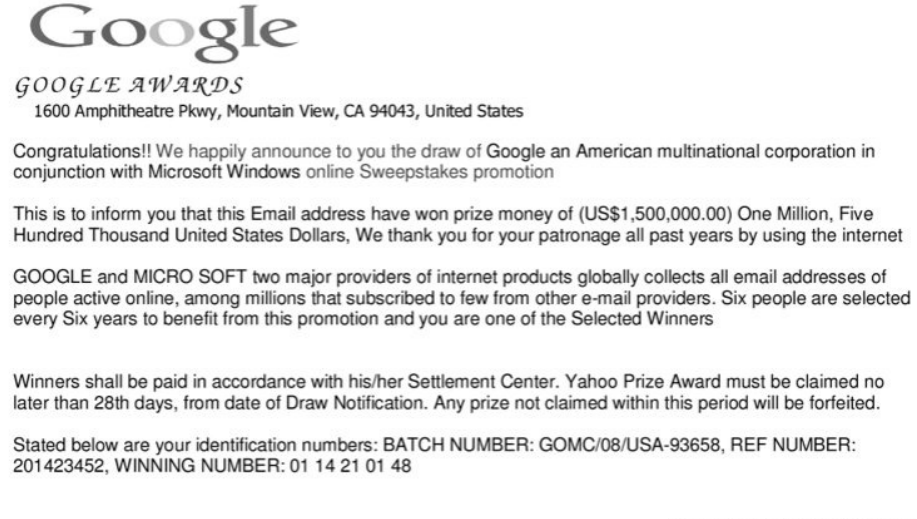
*Corresponding author. E-mail: oaeludoyin@oauife.edu.ng.

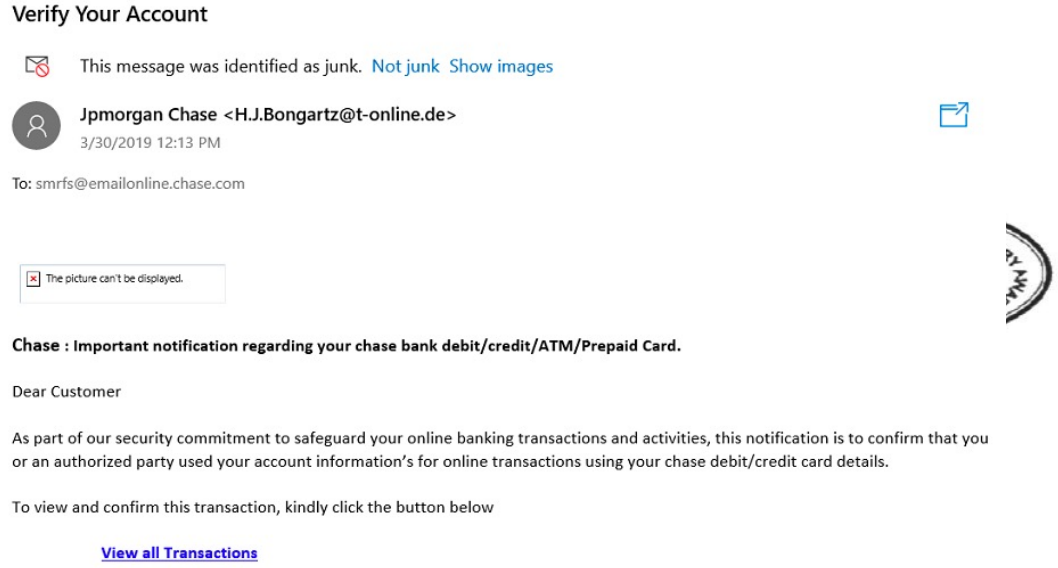
ST. MARY'S UNIVERSITY

Machine Learning for spam Detector







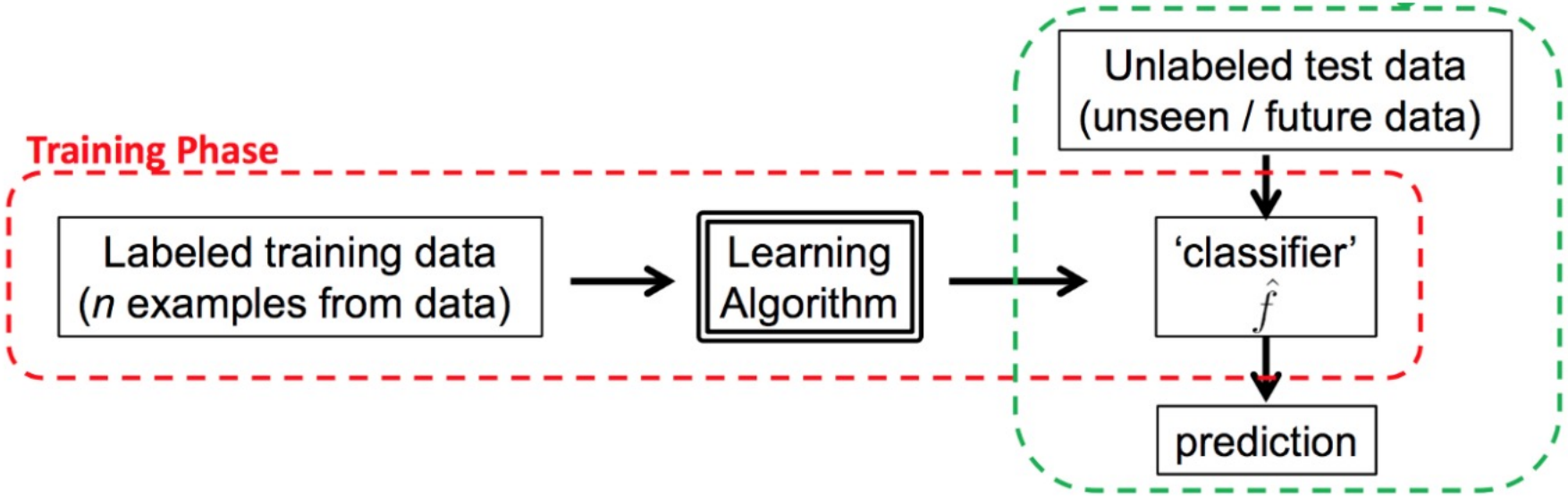


Machine Learning for spam Detector- Cont.

Spam emails pile

Non-spam emails pile

Machine Learning for spam Detector- Cont.



http://www.cs.columbia.edu/~verma/classes/ml/lec/lec1_intro_mle_bayes_naive_evaluation.pdf

A Simple Approach to Binary Classification (spam or non-spam email)

- We want to make a classification decision - spam(class 1) or non-spam(class 2)? - based on some evidence x, which we call email features, e.g., counts of important words or phrases (such as “prince”, “bill”, “money”, …)

$$Class(x) = \begin{cases} c_1 & \text{if } P(c_1|x) > P(c_2|x) \\ c_2 & \text{if } P(c_2|x) > P(c_1|x) \end{cases}$$

Naïve Bayes

- A **Naive Bayes Classifier** determines the probability that an example belongs to some class, calculating the probability that an event will occur given that some input event has occurred.
- When it does this calculation it is assumed that all the predictors of a class have the same effect on the outcome, that the predictors are independent.

Bayes Theorem

- Probability of an event, based on prior knowledge of conditions that might be related to the event.
- Risk of developing health problems is known to increase with age.

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

If $P(c_1|x) > P(c_2|x)$ then:

$$\frac{P(x|c_1)P(c_1)}{P(x)} > \frac{P(x|c_2)P(c_2)}{P(x)}$$

Categories of Machine Learning

- Supervised learning
- Unsupervised learning

Supervised learning

- Modeling the relationship between measured features of data and some label associated with the data;
- Once this model is determined, it can be used to apply labels to new, unknown data.
- Further subdivided into
  - classification tasks: the labels are discrete categories
  - Regression tasks: the labels are continuous quantities

Unsupervised learning

- Modeling the features of a dataset without reference to any label
- Clustering algorithms
  - Identify distinct groups of data

Classification as a Type of Supervised Learning

How would you differentiate a spam email from not spam?

Training a Classification Model

- Supervised learning means that the data fed to the model is already labeled, with the important features/attributes already separated into distinct categories beforehand.
- This means that the model learns which parts of the features are important, and there is also a target that the model can check itself against.
- The process of training a model is the process of feeding data into a model and letting it learn the patterns of the data.
- The training process takes in the data and pulls out the features of the dataset.
- During the training process for a classification task the model is passed both the features and the labels of the training data.
- However, during testing, the network is only fed features.

Testing a Classification Model

- The testing process is where the patterns that the model has learned are tested.
- The features are given to the network, and the network must predict the labels.
- The data for the model is divided into training and testing sets, two different sets of inputs.
- You do not test the classifier model on the same dataset you train it on, as the model has already learned the patterns of this set of data and it would be extreme bias.
- Instead, the dataset is split up into training and testing sets, a set the classifier trains on and a set the classifier has never seen before.

Classification Pipeline

- Preparing data
- Creating training/testing sets
- Instantiating the classification Model
- Training the classifier
- Making predictions
- Evaluating performance

Step 1: Preparing Data

- The first step to training a classifier on a dataset is to prepare the dataset - to get the data into the correct form for the classifier and handle any anomalies in the data.
- If there are missing values in the data, outliers in the data, or any other anomalies these data points should be handled, as they can negatively impact the performance of the classifier.
- This step is referred to as **data preprocessing**.

Step 2: Creating Training/Testing Sets

- Once the data has been preprocessed, the data must be split into training and testing sets.

Step 3: Instantiating the Classification Model

- As previously discussed, the classification model has to be instantiated and trained on the training data.

Step 4: Making Predictions

- After this, predictions can be made with the classification model.

Step 5: Evaluating Performance

- By comparing the predictions made by the classification model to the actual known values of the labels in the test data, we can get a measurement of how accurate the classification model is.
- There are various methods comparing the predicted labels to the actual labels and evaluating the classification model.
- Classification accuracy: the number of correct predictions divided by all predictions or a ratio of correct predictions to total predictions.

Step 5: Evaluating Performance (Cont.)

Confusion matrix

		Predicted	
		Negative (**N**) **-**	Positive (**P**) **+**
Actual	Negative **-**	True Negatives (T**N**)	False Positives (F**P**) **Type I error**
	Positive **+**	False Negatives (F**N**) **Type II error**	True Positives (T**P**)

Step 5: Evaluating Performance (Cont.)

- **Precision** for a class is the number of true positives (i.e. the number of items correctly labelled as belonging to the positive class) divided by the total number of elements labelled as belonging to the positive class (i.e. the sum of true positives and false positives, which are items incorrectly labelled as belonging to the class).
- **Recall** is defined as the number of true positives divided by the total number of elements that actually belong to the positive class (i.e. the sum of true positives and false negatives, which are items which were not labelled as belonging to the positive class but should have been).

Introducing Scikit-Learn

- Python library that provide solid implementations of a range of machine learning algorithms
- Open your command line (cmd on Windows and terminal on Mac) and type the following command

conda install -c conda-forge scikit-learn

Data Representation in Scikit-Learn

- Machine learning is about creating models from data.
- For that reason, we'll start by discussing how data can be represented in order to be understood by the computer.
- The best way to think about data within Scikit-Learn is in terms of tables of data.

Iris Data as a Table

- Famously analyzed by Ronald Fisher in 1936.
- We can download this dataset in the form of a Pandas DataFrame using the Seaborn library
- More about Iris data

ST. MARY'S UNIVERSITY

Iris Data as a Table - Code

In [1]:
```
import seaborn as sns
iris = sns.load_dataset('iris')
iris.head()
```

Out[1]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Iris Data

Out[1]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

- Each row of the data refers to a single observed flower, and the number of rows is the total number of flowers in the dataset. In general, we will refer to the rows of the matrix as samples , and the number of rows as n_samples .
- Each column of the data refers to a particular quantitative piece of information that describes each sample. In general, we will refer to the columns of the matrix as features , and the number of columns as n_features .

```
In [2]: iris.head(20)
```

Out[2]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa
10	5.4	3.7	1.5	0.2	setosa
11	4.8	3.4	1.6	0.2	setosa
12	4.8	3.0	1.4	0.1	setosa
13	4.3	3.0	1.1	0.1	setosa
14	5.8	4.0	1.2	0.2	setosa
15	5.7	4.4	1.5	0.4	setosa
16	5.4	3.9	1.3	0.4	setosa
17	5.1	3.5	1.4	0.3	setosa
18	5.7	3.8	1.7	0.3	setosa
19	5.1	3.8	1.5	0.3	setosa

ST. MARY'S UNIVERSITY

Basic Statistics of Iris Data

```
In [3]: # descriptions
        iris.describe()
```

Out[3]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Iris species

Iris Setosa

Iris versicolor

Iris virginica

Features matrix

- This table layout makes clear that the information can be thought of as a two-dimensional numerical array or matrix, which we will call the features matrix .
- By convention, this features matrix is often stored in a variable named x
- The features matrix is assumed to be two-dimensional, with shape [n_samples, n_features] , and is most often contained in a NumPy array or a Pandas DataFrame.

Target Array

- In addition to the feature matrix x, we also generally work with a label or target array, which by convention we will usually call y.
- The target array is usually one dimensional, with length n_samples, and is generally contained in a NumPy array or Pandas Series.
- The distinguishing feature of the target array is that it is usually the quantity we want to predict from the data: in statistical terms, it is the dependent variable.
- For example, in the preceding data we may wish to construct a model that can predict the species of flower based on the other measurements; in this case, the species column would be considered the feature.

Iris Data - Target Array

```
In [4]: #class distribution
        iris.groupby('species').size()

Out[4]: species
        setosa          50
        versicolor      50
        virginica       50
```

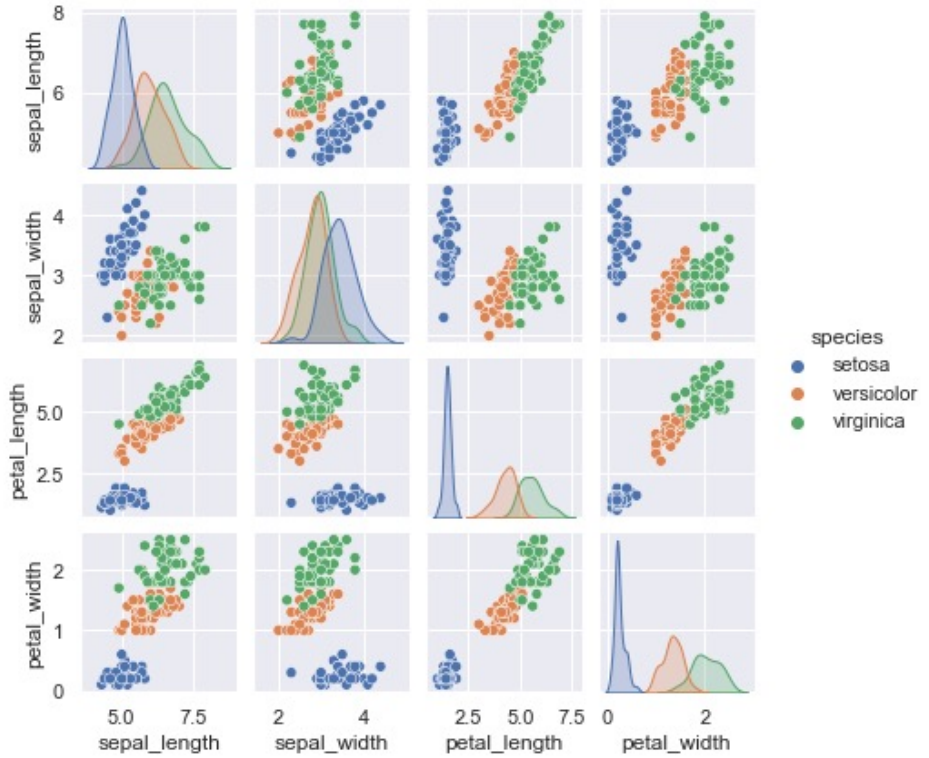
ST. MARY'S UNIVERSITY

Iris Data – Visualization using Seaborn

- Seaborn is an API that provides visualization on Pandas DataFrames.
- With target array in mind, we can use Seaborn to visualize Iris data.

```
In [5]: sns.set()
        sns.pairplot(iris,hue='species', size=1.5);
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/axisgrid.py:1969: UserWarning: The `size`
parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```



Iris Data – Classification Pipeline Code

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
X_iris= iris.drop('species', axis=1)
X_iris.shape
y_iris=iris['species']
y_iris.shape
Xtrain, Xtest, ytrain, ytest = train_test_split(X_iris,y_iris, random_state=1)
```

Iris Data – Classification Pipeline Code (Cont.)

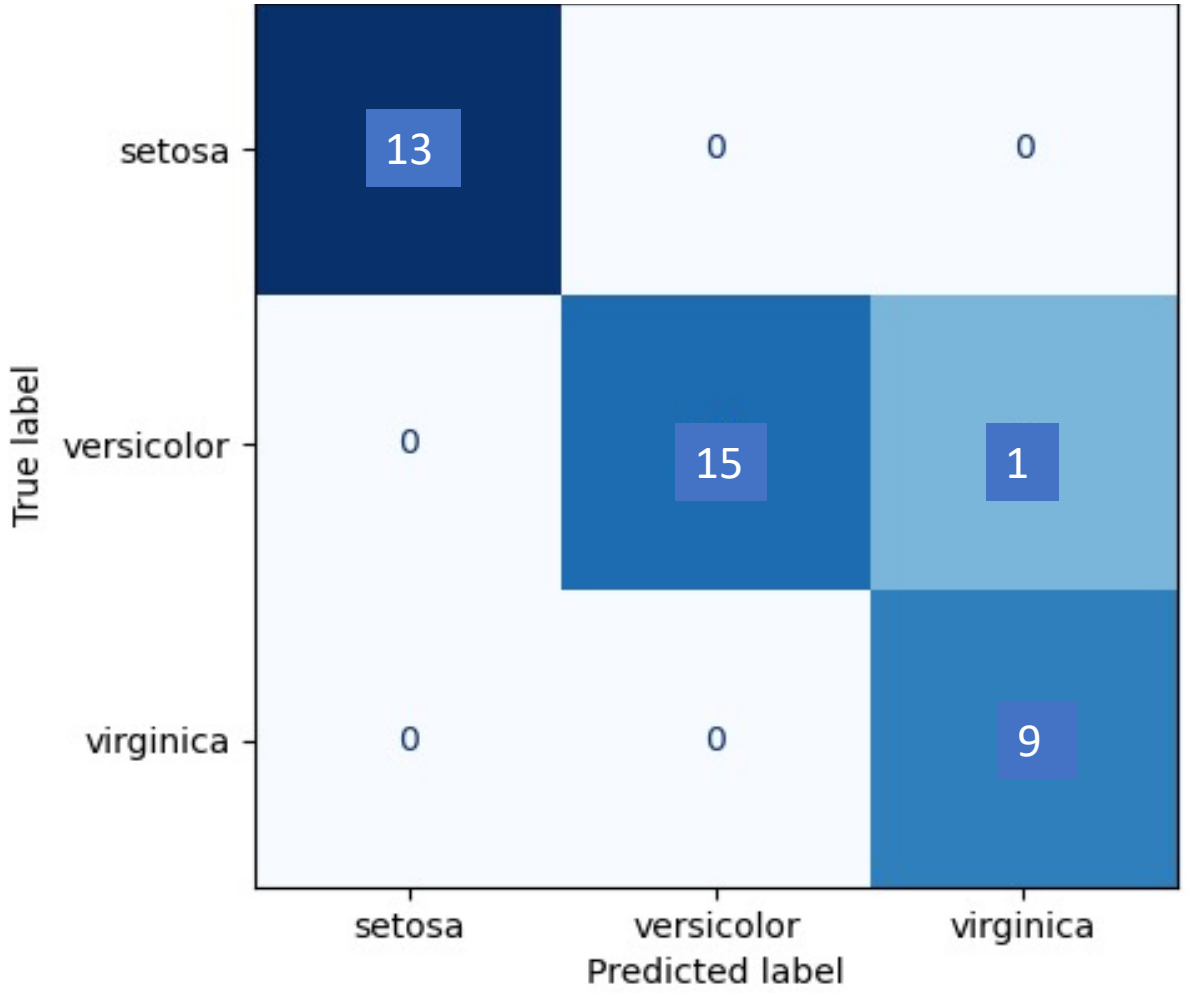
```
#Choose a model and fit it on training data (learning)
model= GaussianNB()
model.fit(Xtrain, ytrain)
#predict on the test data
yPredicition = model.predict(Xtest)
#Evaluate the preformance
print(accuracy_score(ytest, yPredicition))
print(confusion_matrix(ytest, yPredicition))
print(classification_report(ytest, yPredicition))
```

from sklearn.naive_bayes import GaussianNB

- In this classifier, the assumption is that data from each label is drawn from a simple Gaussian distribution

ST. MARY'S UNIVERSITY

Iris Data– Classification Confusion Matrix



Calculating Precision and Recall for Iris Data

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

TP = True positive

TN = True negative

FP = False positive

FN = False negative

Some Definitions

- **Support** is the number of actual occurrences of the class in the specified dataset.
- Macro avg
- Weighted avg

Handwritten Digits

- Optical character recognition problem: the identification of handwritten digits
- In the wild, this problem involves both locating and identifying characters in an image.
- Here we'll take a shortcut and use Scikit-Learn's set of preformatted digits, which is built into the library.

Loading the Digits Data

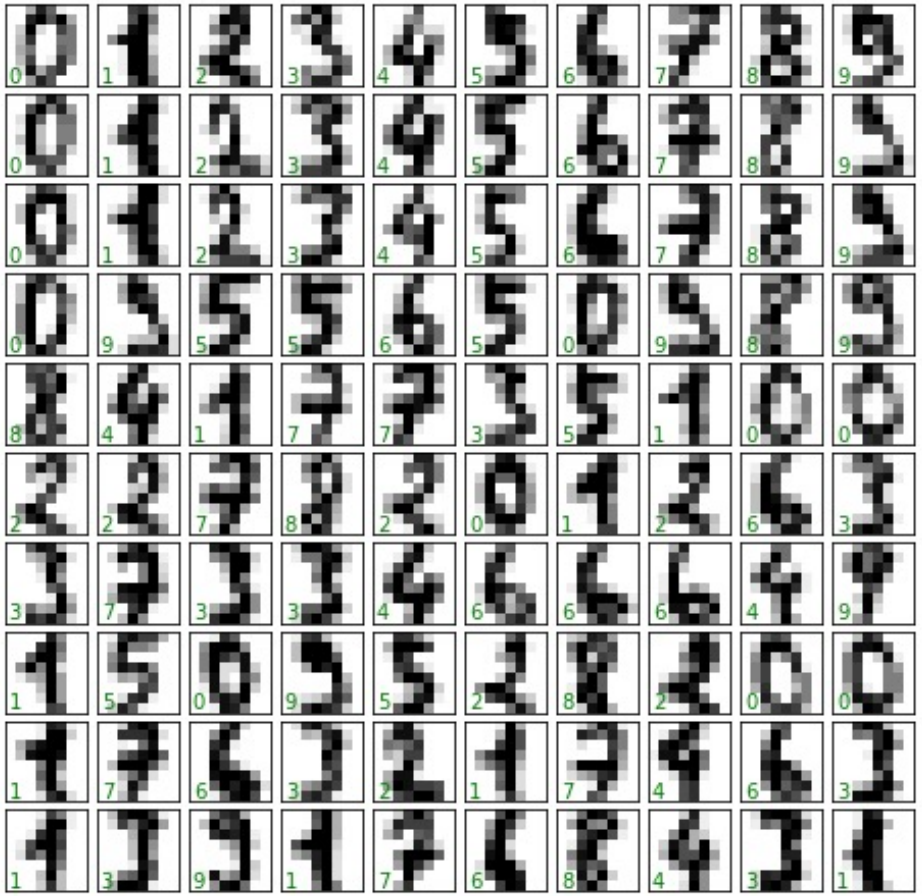
- The images data is a three-dimensional array: 1,797 samples, each consisting of an 8×8 grid of pixels.

```
In [1]: from sklearn.datasets import load_digits
        digits = load_digits()
        digits.images.shape

Out[1]: (1797, 8, 8)
```

ST. MARY'S UNIVERSITY

Visualizing the Digits Data

```
import matplotlib.pyplot as plt
fig, axes = plt.subplots(10,10, figsize=(8,8),
                         subplot_kw={'xticks':[], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.1,wspace=0.1))
for i, ax in enumerate(axes.flat):
    ax.imshow(digits.images[i], cmap='binary', interpolation='nearest')
    ax.text(0.05,0.05, str(digits.target[i]),
            transform=ax.transAxes, color='green')
```



Feature Matrix and Target Array

- In order to work with this data within Scikit-Learn, we need a two-dimensional, [n_samples, n_features] representation.
- We can accomplish this by treating each pixel in the image as a feature — that is, by flattening out the pixel arrays so that we have a length-64 array of pixel values representing each digit.
- Additionally, we need the target array, which gives the previously determined label for each digit.
- These two quantities are built into the digits dataset under the data and target attributes.

Feature Matrix and Target Array (Cont.)

```
In [12]: X=digits.data
         X.shape

Out[12]: (1797, 64)

In [11]: y=digits.target
         y.shape

Out[11]: (1797,)

In [15]: print(X[:5,:])
         print(y[50:60])
```

Train and Test Set Splits

- As with the Iris data previously, we will split the data into a training and test set.

```
In [18]: from sklearn.model_selection import train_test_split
         Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=0)
```

Instantiating and Fitting the Model

```
In [20]: from sklearn.naive_bayes import GaussianNB
         model = GaussianNB()
         model.fit(Xtrain, ytrain)
         y_prediction = model.predict(Xtest)
```

ST. MARY'S UNIVERSITY

Performance Evaluation

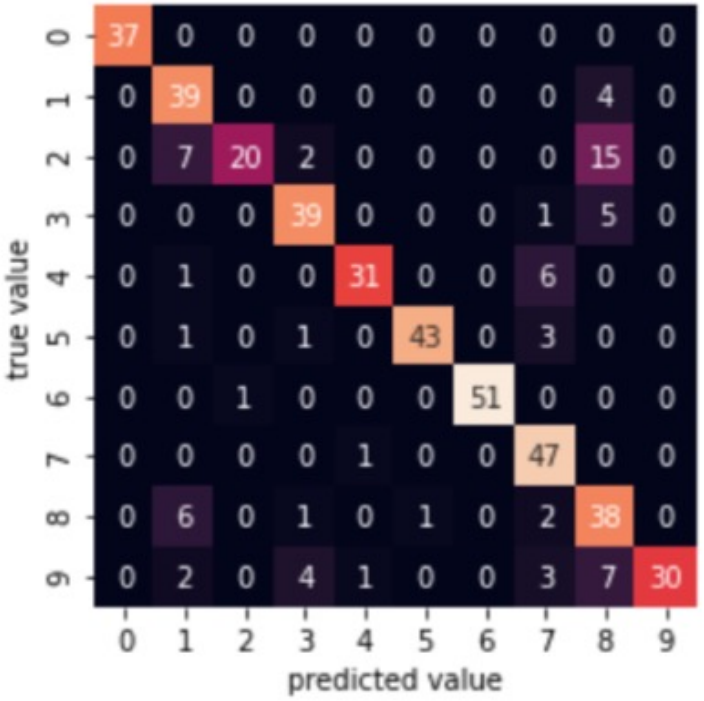
```
In [22]: from sklearn.metrics import classification_report
         from sklearn.metrics import accuracy_score
         print(accuracy_score(ytest, y_prediction))
         print(classification_report(ytest, y_prediction))
```

```
0.8333333333333334
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        37
           1       0.70      0.91      0.79        43
           2       0.95      0.45      0.62        44
           3       0.83      0.87      0.85        45
           4       0.94      0.82      0.87        38
           5       0.98      0.90      0.93        48
           6       1.00      0.98      0.99        52
           7       0.76      0.98      0.85        48
           8       0.55      0.79      0.65        48
           9       1.00      0.64      0.78        47

    accuracy                           0.83       450
   macro avg       0.87      0.83      0.83       450
weighted avg       0.87      0.83      0.83       450
```

Confusion Matrix

- This shows us where the mislabeled points tend to be: for example, a large number of twos here are misclassified as either ones or eights.

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
mat = confusion_matrix(ytest, y_prediction)

sns.heatmap(mat, square=True, annot=True, cbar=False)
plt.xlabel('predicted value')
plt.ylabel('true value');
```



Classification Algorithms and Comparison

- Naive Bayes
- Logistic regression
- K-nearest neighbors
- (Kernel) SVM
- Decision tree

Naive Bayes

- Naive Bayes applies the Bayes' theorem to calculate the probability of a data point belonging to a particular class/label.
- Given the probability of certain related values, the formula to calculate the probability of an event *B*, given event *A* to occur is calculated: P(B|A) = (P(A|B) * P(B) / P(A))
- This theory is considered naive, because it assumes that there is no dependency between any of the input features.
- Naive Bayes algorithm has been proven to perform really well in certain use cases like spam filters.

Logistic Regression

- Logistic regression is an extension to the linear regression algorithm.
  - Linear regression algorithm: the variable to be predicted depends on only one other variable.
  - This is calculated by using the formula that is generally used in calculating the slope of a line.
  - $y = w_0 + w_1 * x_1$
  - *y* refers to the target variable,
  - x_1 refers to the independent variable
  - w_1 refers to the coefficient that expresses the relationship between *y* and x_1 and is also known as the slope.
  - w_0 is the constant coefficient, or the intercept. It refers to the constant offset that *y* will always be with respect to the independent variables.

Logistic Regression (Cont.)

- In a logistic regression algorithm, instead of predicting the actual continuous value, we predict the probability of an outcome.
- To achieve this, a *logistic function* is applied to the outcome of the linear regression.
- This outputs a value between 0 and 1.

K-nearest neighbors

- The general idea behind K-nearest neighbors (KNN) is that data points are considered to belong to the class with which it shares the most number of common points in terms of its distance.
- K number of nearest points around the data point to be predicted are taken into consideration.
- These K points at this time already belong to a class.
- The data point under consideration is said to belong to the class with which the most number of points from these K points belong.
- The most popular formula to calculate this is the Euclidean distance.

Support Vector Machines

- Support Vector Machines (SVM) output an optimal line of separation between the classes, based on the training data entered as input.
- This line of separation is called a hyperplane in a multi-dimensional environment.
- After the model is constructed with this hyperplane, any new point to be predicted checks to see which side of the hyperplane this values lies in.

Decision Trees

- Decision tree-based models use training data to derive rules that are used to predict an output.
- For example, assume that the problem statement was to identify if a person can play tennis today.
- Depending on the values from the training data, the model forms a decision tree.
  - First check the outlook column. If it's overcast, you definitely never go.
  - But if it's sunny and humid, then you don't go.
  - If it's sunny and normal, you go.
  - If it's rainy and windy, you don't go.
  - And if it's rainy and not windy, you go.

Iris Data – Logistic Regression Model

```
from sklearn.linear_model import LogisticRegression
lrModel = LogisticRegression(solver='liblinear', multi_class='ovr')
lrModel.fit(Xtrain, ytrain)
lryPredicition = lrModel.predict(Xtest)
#Evaluate the preformance
print(accuracy_score(ytest, lryPredicition))
print(confusion_matrix(ytest, lryPredicition))
print(classification_report(ytest, lryPredicition))
```

```
0.8421052631578947
[[13  0  0]
 [ 0 10  6]
 [ 0  0  9]]
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        13
  versicolor       1.00      0.62      0.77        16
   virginica       0.60      1.00      0.75         9

    accuracy                           0.84        38
   macro avg       0.87      0.88      0.84        38
weighted avg       0.91      0.84      0.84        38
```

Iris Data – K-nearest neighbors

```
from sklearn.neighbors import KNeighborsClassifier
knnModel = KNeighborsClassifier()
knnModel.fit(Xtrain, ytrain)
knnPrediction = knnModel.predict(Xtest)
#Evaluate the preformance
print(accuracy_score(ytest, knnPrediction))
print(confusion_matrix(ytest, knnPrediction))
print(classification_report(ytest, knnPrediction))
```

```
1.0
[[13  0  0]
 [ 0 16  0]
 [ 0  0  9]]
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        13
  versicolor       1.00      1.00      1.00        16
   virginica       1.00      1.00      1.00         9

    accuracy                           1.00        38
   macro avg       1.00      1.00      1.00        38
weighted avg       1.00      1.00      1.00        38
```

Iris Data – Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dtModel = DecisionTreeClassifier()
dtModel.fit(Xtrain, ytrain)
dtPrediction = dtModel.predict(Xtest)
#Evaluate the preformance
print(accuracy_score(ytest, dtPrediction))
print(confusion_matrix(ytest, dtPrediction))
print(classification_report(ytest, dtPrediction))
```

```
0.9736842105263158
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        13
  versicolor       1.00      0.94      0.97        16
   virginica       0.90      1.00      0.95         9

    accuracy                           0.97        38
   macro avg       0.97      0.98      0.97        38
weighted avg       0.98      0.97      0.97        38
```

Iris Data – Support Vector Machine (SVM)

```
from sklearn.svm import SVC
svmModel = SVC(gamma='auto')
svmModel.fit(Xtrain, ytrain)
svmPrediction = svmModel.predict(Xtest)
#Evaluate the preformance
print(accuracy_score(ytest, svmPrediction))
print(confusion_matrix(ytest, svmPrediction))
print(classification_report(ytest, svmPrediction))
```

```
0.9736842105263158
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        13
  versicolor       1.00      0.94      0.97        16
   virginica       0.90      1.00      0.95         9

    accuracy                           0.97        38
   macro avg       0.97      0.98      0.97        38
weighted avg       0.98      0.97      0.97        38
```

ST. MARY'S UNIVERSITY

Classification Model: Distinguish between Different Types of Fruits

Fruits Data

- The fruits dataset was created by Dr. Iain Murray from University of Edinburgh. He bought a few dozen oranges, lemons and apples of different varieties, and recorded their measurements in a table. And then the professors at University of Michigan formatted the fruits data slightly and it can be downloaded from here.

Fruits Classification Model

```
import pandas as pd
import matplotlib.pyplot as plt
fruits = pd.read_table('fruit_data_with_colors.txt')
fruits.head()
fruits.pop('fruit_label')
fruits.head()
fruits.pop('fruit_subtype')
fruits.head()
print(fruits.shape)
print(fruits['fruit_name'].unique())
print(fruits.groupby('fruit_name').size())
```

```
import seaborn as sns
sns.countplot(fruits['fruit_name'],label="Count")
plt.show()
from matplotlib import pyplot
# histograms
#Create histogram of each input variable to get an idea of the distribution.
fruits.hist()
pyplot.show()
#It looks like perhaps color score has a near Gaussian distribution.
sns.set()
sns.pairplot(fruits,hue='fruit_name', size=1.5);
# descriptions
fruits.describe()
#We can see that the numerical values do not have the same scale.
#We will need to apply scaling to the test set that we computed for the training set.
```

```
X= fruits.drop('fruit_name', axis=1)
X.Shape
y=fruits['fruit_name']
y.Shape
#Create Training and Test Sets and Apply Scaling
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import GaussianNB
```

```
#Choose a model and fit it on training data (learning)
model= GaussianNB()
model.fit(X_train, y_train)
#predict on the test data
y_predicition = model.predict(X_test)
#Evaluate the preformance
print(accuracy_score(y_test, y_predicition))
print(confusion_matrix(y_test, y_predicition))
print(classification_report(y_test, y_predicition))
```

Unsupervised Learning: Clustering Algorithms

- Clustering algorithms seek to learn, from the properties of the data, an optimal division or discrete labeling of groups of points.
- Many clustering algorithms are available in Scikit-Learn and elsewhere, but perhaps the simplest to understand is an algorithm known as k-means clustering , which is implemented in sklearn.cluster.Kmeans
- Example: Topic Modeling

Introducing k-Means

- The k -means algorithm searches for a predetermined number of clusters within an unlabeled multidimensional dataset.
- It accomplishes this using a simple conception of what the optimal clustering looks like:
  - The "cluster center" is the arithmetic mean of all the points belonging to the cluster.
  - Each point is closer to its own cluster center than to other cluster centers.

$$A = \frac{1}{n}\sum_{i=1}^{n} a_i$$

A = arithmetic mean

n = number of values

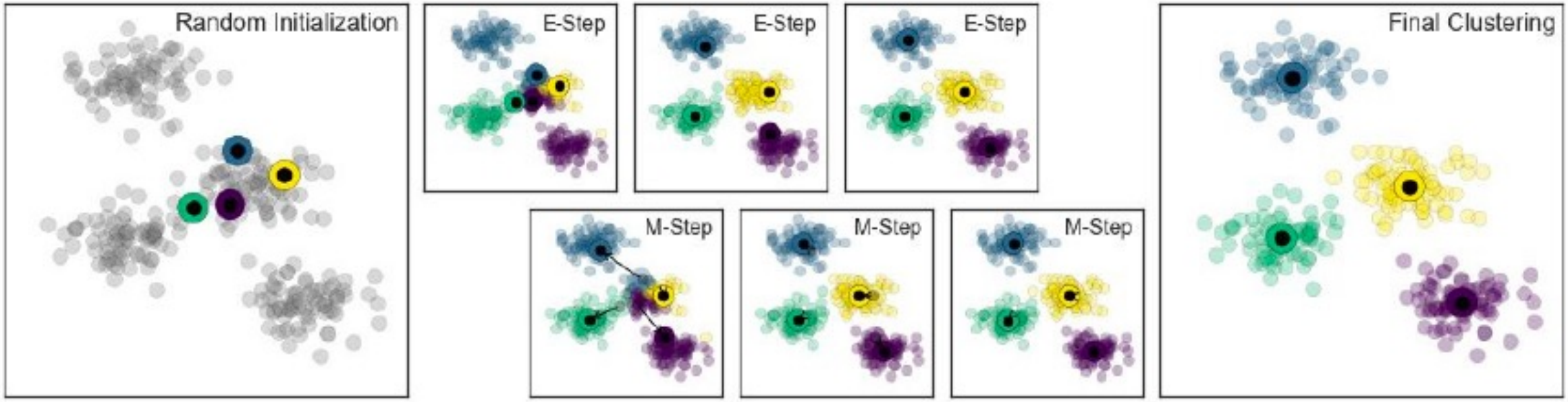
a_i = data set values

In Depth: k-Means Clustering

- k-Means Algorithm: Expectation–Maximization algorithm
- In short, the expectation–maximization approach consists of the following procedure:

    1. Guess some cluster centers
    2. Repeat until converged
        a. E-Step : assign points to the nearest cluster center
        b. M-Step : set the cluster centers to the mean

Visualization of the E–M algorithm for k-means



k-Means on digits

- Here we will attempt to use k -means to try to identify similar digits without using the original label information
- This might be similar to a first step in extracting meaning from a new dataset about which you don't have any a priori label information
- We will start by loading the digits and then finding the KMeans clusters. Recall that the digits consist of 1,797 samples with 64 features, where each of the 64 features is the brightness of one pixel in an 8×8 image.

ST. MARY'S UNIVERSITY

In [1]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set() #for plot styling
import numpy as np
from sklearn.cluster import KMeans
```

In [2]:
```
from sklearn.datasets import load_digits
digits = load_digits()
digits.data.shape
```

Out[2]:
```
(1797, 64)
```

In [3]:
```
kmeans = KMeans(n_clusters=10, random_state=0)
clusters = kmeans.fit_predict(digits.data)
kmeans.cluster_centers_.shape
```

Out[3]:
```
(10, 64)
```

In [4]:
```
fig, ax = plt.subplots(2, 5, figsize=(8, 3))
centers = kmeans.cluster_centers_.reshape(10,8,8)
for axi, center in zip(ax.flat, centers):
    axi.set(xticks=[], yticks=[])
    axi.imshow(center, interpolation='nearest', cmap=plt.cm.binary)
```