Practical Data Science: Classification

Dr. Yongli Ren

(yongli.ren@rmit.edu.au)

Computer Science & IT

School of Science

Outline

- Part 1: Overview
- Part 2: Classification I
- Part 3: Classification II

Practical Data Science – COSC2670

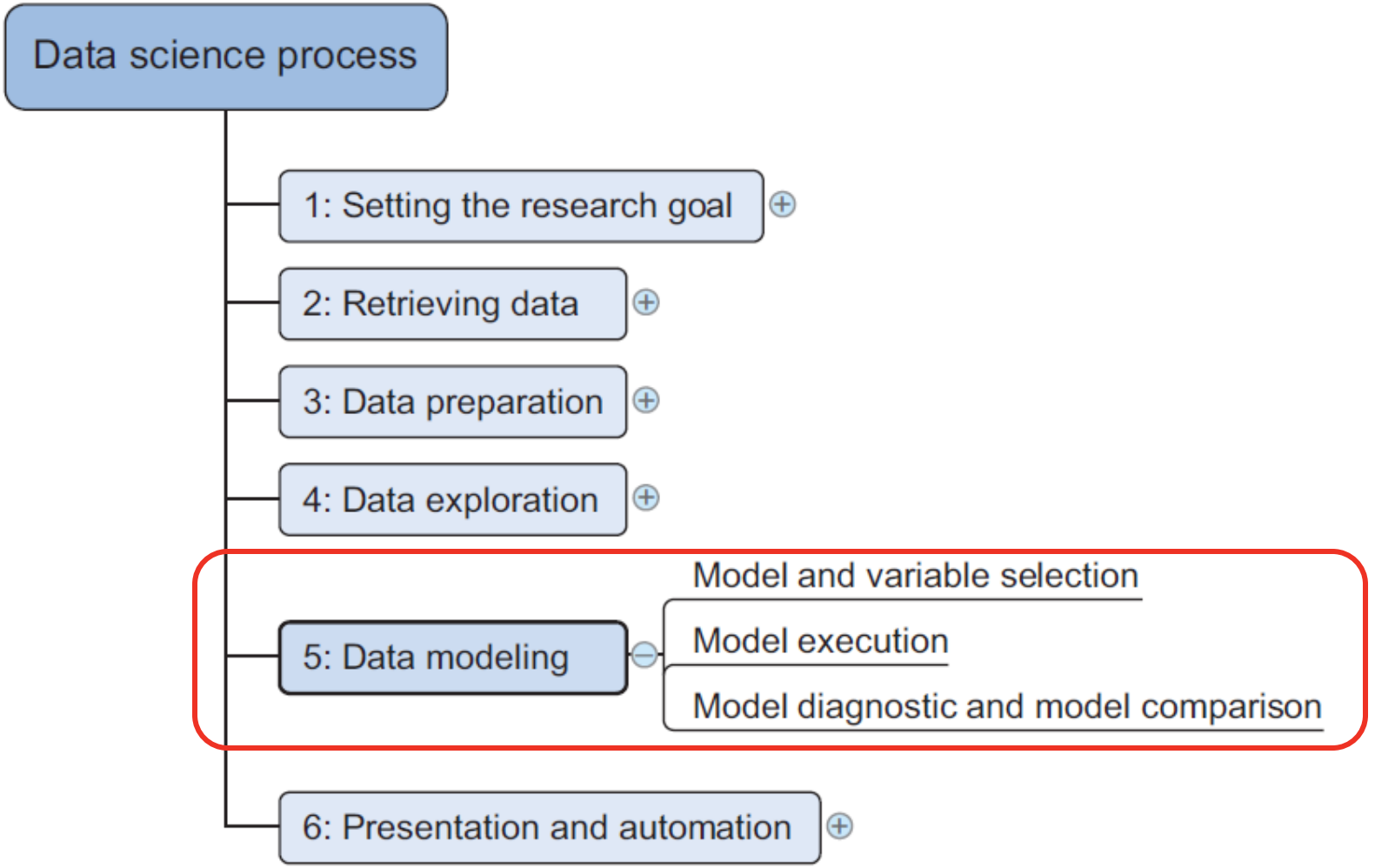
PART 1: OVERVIEW

Overview

- Do you know how computers learn to protect you from malicious persons?
  - Computers filter out more than 60% of your emails and can learn to do an even better job at protecting you over time.

- Can you explicitly teach a computer to recognize persons in a picture?
  - It's possible but impractical to encode all the possible ways to recognize a person,
  - but you'll soon see that the possibilities are nearly endless.
- To succeed, you'll need to add a new skill to your toolkit, *machine learning*,
  - which is the core of data modelling in data science projects.

Where Machine Learning is Used in the Data Science Process?



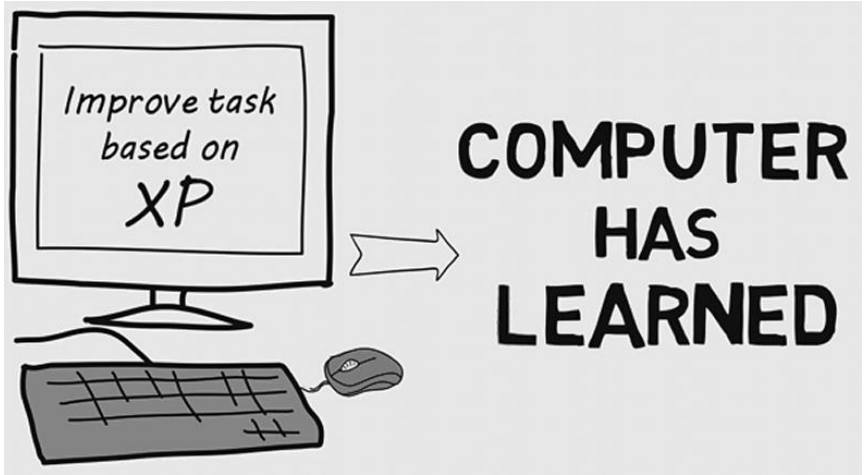
D. Cielen et. Al., Chapter 3

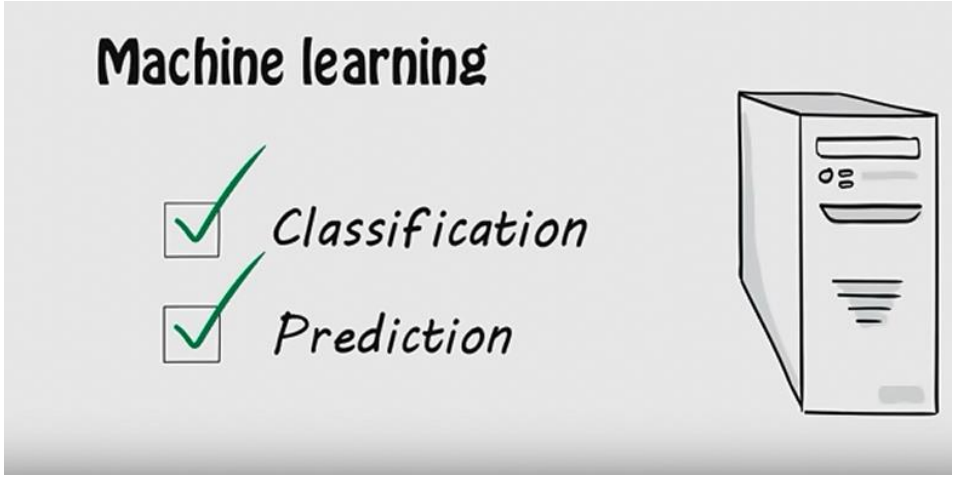
What is Machine Learning?

"Machine learning is a field of study that gives computers the ability to learn without being explicitly programmed."

—Arthur Samuel, 1959

- 60s introduction:
  - https://www.youtube.com/watch?v=NOjUaY0Qn3g
- The more data or "experience" the computer gets, the better it becomes at its designated job, like a human does.





What is Machine Learning?

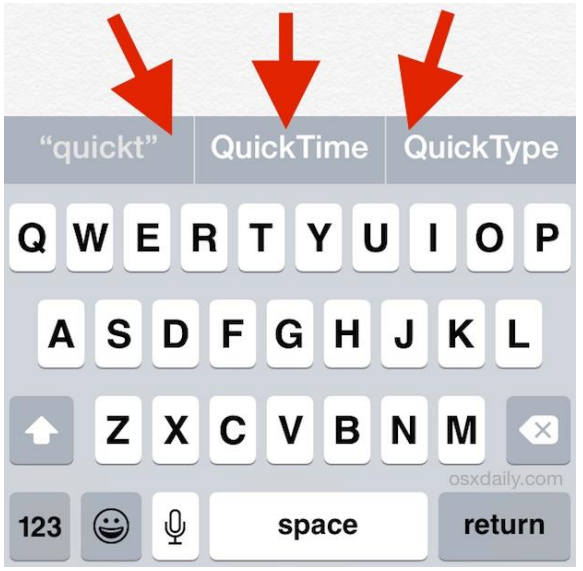
- When machine learning is seen as a process, the following definition is insightful:

"Machine learning is the process by which a computer can work more accurately as it collects and learns from the data it is given."

—Mike Roberts

- For example, as a user writes more text messages on a phone, the phone learns more about the user's common vocabulary and can predict (*autocomplete*) their words faster and more accurately.

QuickType Suggestion (iOS)



http://osxdaily.com/2014/10/05/hide-quicktype-bar-keyboard-ios/

Applications for machine learning in data science

- *Classification, clustering, and regression (prediction)* are of primary importance to a data scientist.
- To achieve these goals, one of the main *tools* a data scientist uses is *machine learning*.
- The uses for automatic classification are wide ranging, e.g.:
  - Finding oil fields, gold mines, or archaeological sites based on existing sites
  - Finding place names or persons in text
  - Identifying people based on pictures or voice recordings
  - Recognizing birds based on their whistle
  - Identifying tumours and diseases
  - …..

Applications for machine learning in data science

- Sometimes, data scientists build a *model*
  - that provides insight to the underlying processes of a phenomenon.
- When the goal of a model *isn't prediction but interpretation*, it's called *root cause analysis*.
- Here are a few examples:
  - Understanding and optimizing a business process, such as
    - Determining which products add value to a product line
    - Discovering what causes diabetes
    - Determining the causes of traffic jams

The Process for Model Building

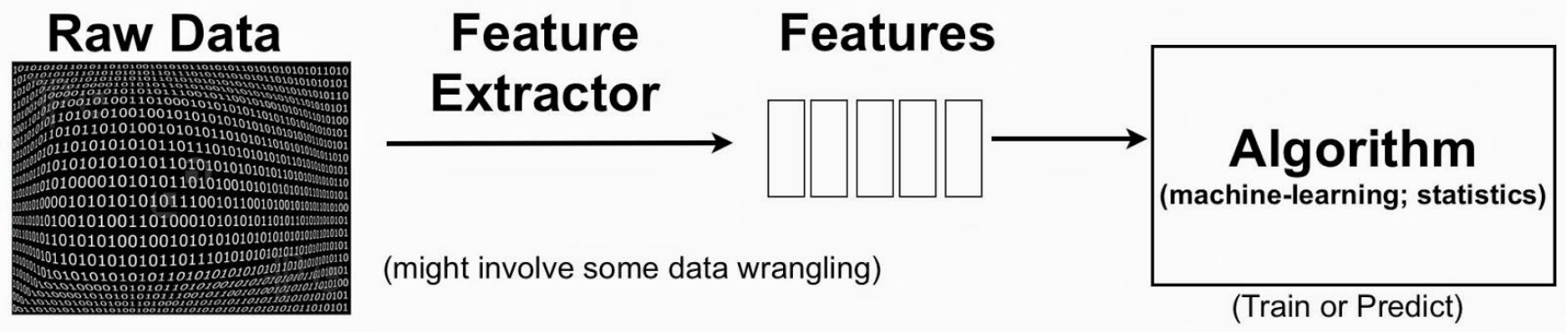
- The modelling phase consists of four steps:

  1. Feature engineering and model selection
  2. Training the model
  3. Model validation and selection
  4. Applying the trained model to unseen data

- Before you find a good model, you'll probably iterate among the first three steps.
- The last step isn't always present because sometimes the goal isn't prediction but explanation (*root cause analysis*).
  - For instance, you might want to find out the causes of species' extinctions but not necessarily predict which one is next in line to leave our planet.

Engineering Features and Selecting a Model

- With engineering features, you must
  - come up with and create possible predictors for the model.
- This is one of the most important steps in the process
  - because a model recombines these features to achieve its predictions.
- Often you may need to consult an expert or the appropriate literature to come up with meaningful features.

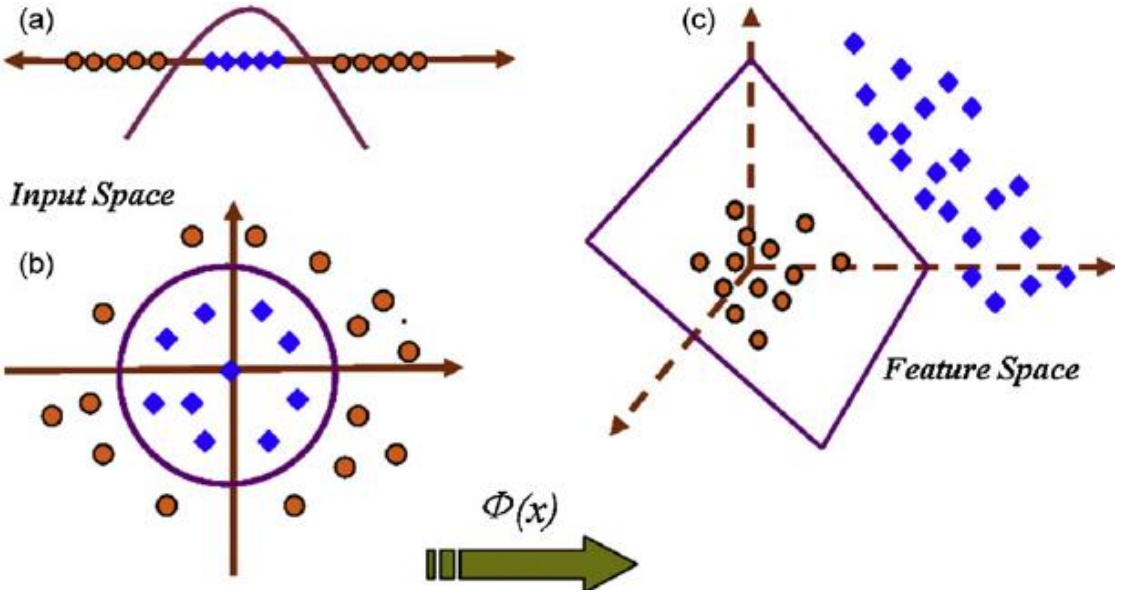


http://radar.oreilly.com/2014/06/streamlining-feature-engineering.html

Engineering Features and Selecting a Model

- Certain features are the variables you get from a data set,
  - as is the case with the provided data sets in most school exercises.
- In practice, you'll need to find the features yourself,
  - which may be scattered among different data sets.
    - For complex projects it may be necessary to bring together 20 or more different data sources before you have the raw data that's required.
- Often you'll need to apply a transformation to an input before it becomes a good predictor, or to combine multiple inputs.
- An example of combining multiple inputs would be *interaction variables*:
  - the impact of either single variable is low, but if both are present their impact becomes immense.
- This is especially true in chemical and medical environments.
  - For example, although vinegar and bleach are fairly harmless common household products by themselves, mixing them results in poisonous chlorine gas, which killed thousands during World War I.

Engineering Features and Selecting a Model

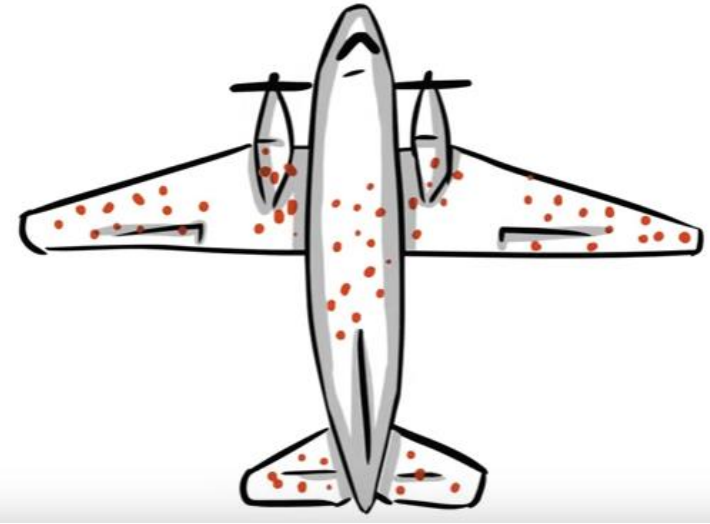


Engineering Features and Selecting a Model

- One of the biggest mistakes in model construction is the *availability bias*:
  - Your features are only the ones that you could easily get your hands on, so your model consequently represents a one-sided "truth".
  - Models suffering from availability bias often fail when they're validated
    - because it becomes clear that they're not a valid representation of the truth.

Engineering Features and Selecting a Model - Abraham Wald and Armour Placement

- In World War II, after bombing runs, many of the English/US planes came back with bullet holes
  - in the wings and the body of the plane.
- Almost none of them had bullet holes
  - in the cockpit, tail rudder, or engine block.
- The question is: where should extra armour be placed to best protect the planes?
- https://www.youtube.com/watch?v=adnNclkrxfE



Training Your Model

- With the right predictors in place and a modelling technique in mind, you can progress to model training.
- In this phase, you present your model with data from which it can learn.
- The most common modelling techniques have industry-ready implementations in almost every programming language, including *Python*.
- These enable you to train your models by executing a few lines of code.
  - For more state-of-the art data science techniques, you'll probably end up
    - doing heavy mathematical calculations and
    - implementing them with modern computer science techniques.
- Once a model is trained, it's time to test whether it can be extrapolated to reality:
  - *model validation.*

Validating a Model

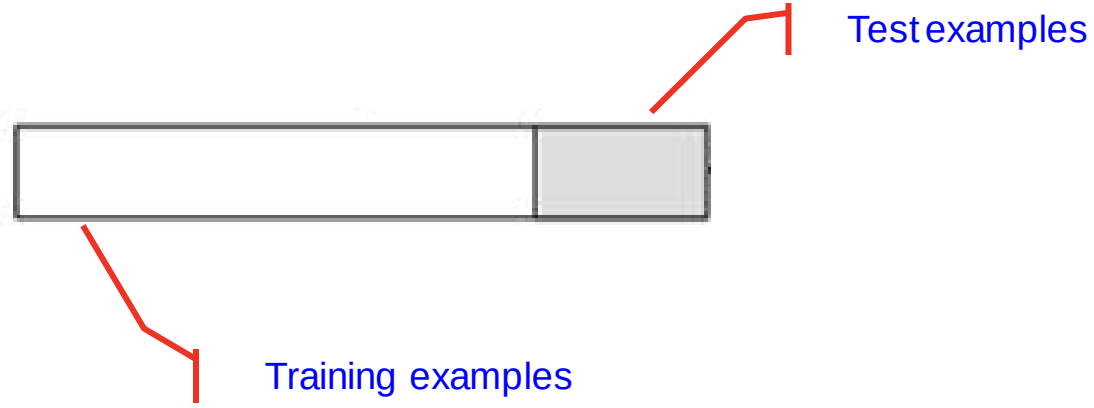
- Data science has many modelling techniques, and the question is
  - which one is the right one to use.
- A good model has two properties:
  - it has good predictive power and
  - it generalizes well to data it hasn't seen.
- To achieve this you define
  - an error measure (how wrong the model is) and
  - a validation strategy.

Validating a Model

- Two common *error measures* in machine learning are
  - the *classification error rate* for classification problems and
  - the *mean squared error* for regression (prediction) problems.
- The classification error rate is
  - the percentage of observations in the test data set that your model mislabelled;
  - lower is better.
- The *mean squared error* measures
  - how big the average error of your prediction is.
    - You can't cancel out a wrong prediction in one direction with a faulty prediction in the other direction.
      - E.g. overestimating future turnover for next month by 5,000 doesn't cancel out underestimating it by 5,000 for the following month.
    - Bigger errors get even more weight than they otherwise would.
      - Small errors remain small or can even shrink (if <1), whereas big errors are enlarged and will definitely draw your attention.

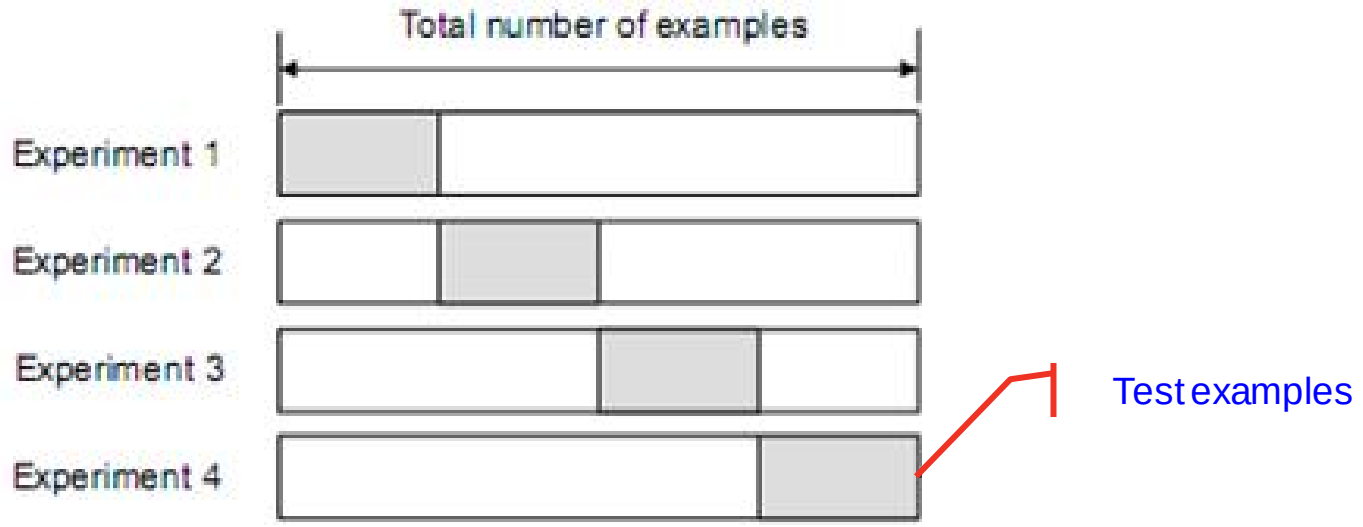
Validating a Model - Validation Strategy

- Many *validation strategies* exist, including the following common ones:
  - *Dividing your data into a training set with X% of the observations and keeping the rest as a holdout data set*
    - (a data set that's never used for model creation)
  - This is the most common technique.



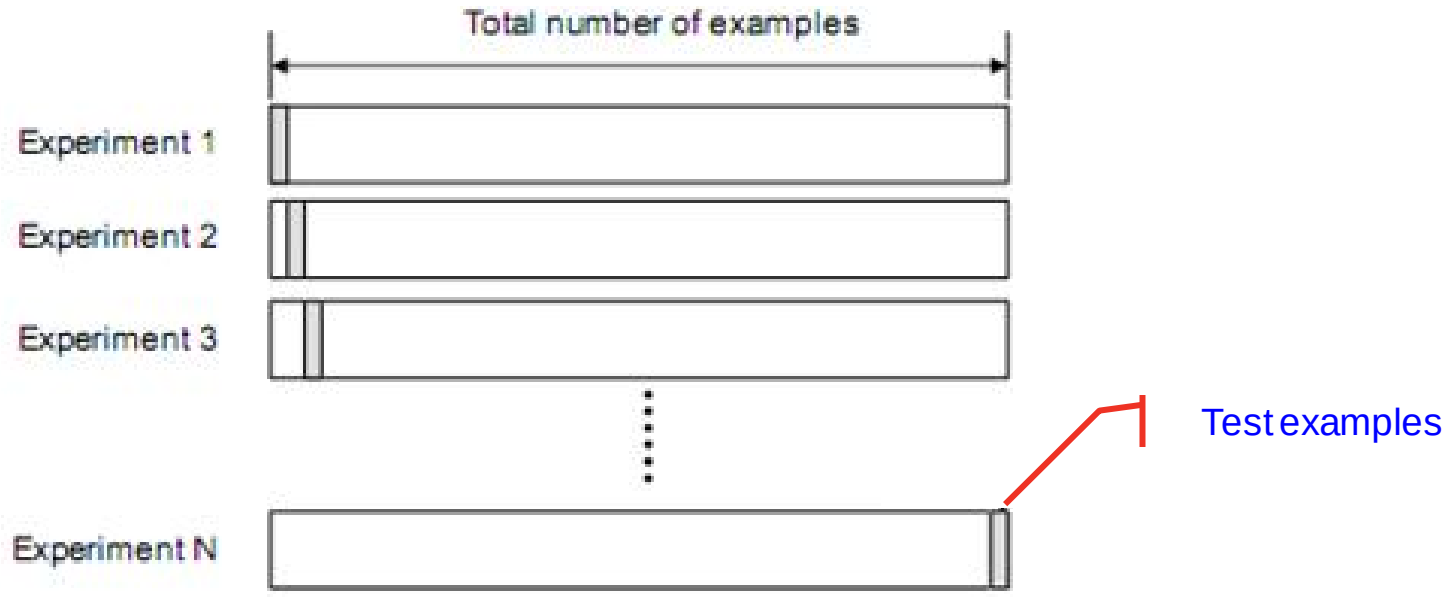
Validating a Model
- Validation Strategy

- *k-folds cross validation*
  - This strategy divides the data set into k parts and uses each part one time as a test data set while using the others as a training data set.
  - This has the advantage that you use all the data available in the data set.



Validating a Model
- Validation Strategy

- *Leave-1 out*
  - This approach is the same as k-folds but with $k=N$,
    - where N denotes the total number of examples.
  - You always leave one observation out and train on the rest of the data.
  - This is usually used only on small data sets, so it's more valuable to people evaluating laboratory experiments than to big data analysts.



D. Cielen et. Al., Chapter 3

Predicting New Observations

- If you've implemented the first three steps successfully, you now have a trained model that generalizes to unseen data.
- The process of applying your model to new data is called *model scoring*.
- In fact, model scoring is something you implicitly did during validation,
  - only now you don't know the correct outcome.
- By now you should trust your model enough to use it for real.
- Model scoring involves two steps.
  - You prepare a data set that has features exactly as defined by your model.
    - This boils down to repeating the data preparation you did in step one of the modelling process but for a new data set.
  - You apply the model on this new data set, and this results in a prediction.
- Now let's look at the different types of machine learning techniques:
  - a different problem requires a different approach.

Machine Learning Techniques

- We can divide the different approaches to machine learning by
  - the amount of human effort that's required to coordinate them and
  - how they use *labelled data*
    - data with a category or a real-value number assigned to it that represents the outcome of previous observations.
- *Supervised learning* techniques
  - attempt to discern results and learn by trying to find patterns in a labelled data set.
  - Human interaction is required to label the data.

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

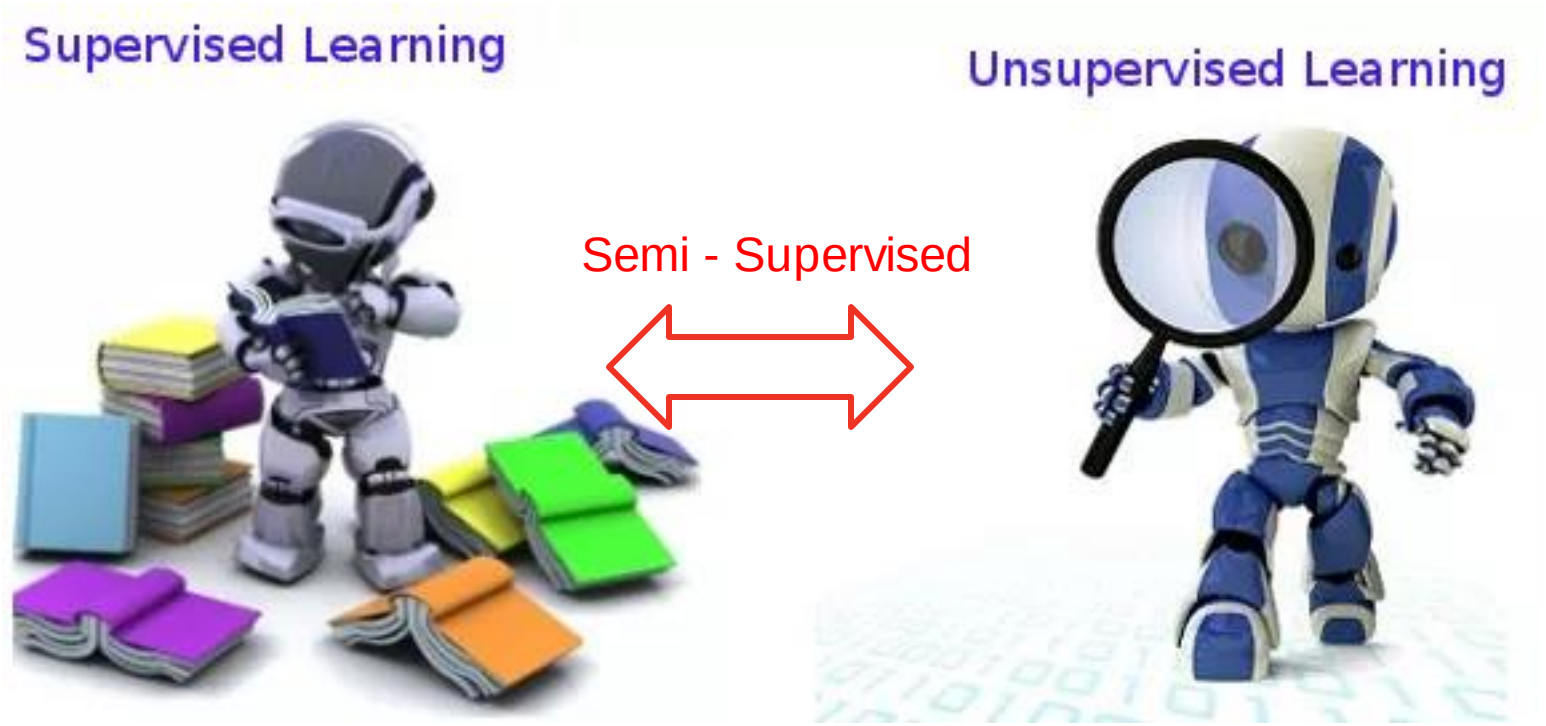
	Sex	Length	Diameter	Height	Shell weight	Rings
0	M	0.455	0.365	0.095	0.15	15
1	M	0.350	0.265	0.090	0.07	7

Machine Learning Techniques

- *Unsupervised learning* techniques
  - don't rely on labelled data, and attempt to find patterns in a data set without human interaction.

Machine Learning Techniques

- *Semi-supervised learning* techniques
  - need some labelled data, and therefore human interaction,
  - so as to find patterns in the data set, but they can still progress toward a result and learn even if passed unlabelled data as well.



Practical Data Science – COSC2670

PART 2: CLASSIFICATION I

Supervised Learning
- Classification

- In machine learning and statistics, *classification* is
  - the problem of identifying to which of a set of categories (sub-populations) a new observation belongs,
  - on the basis of a training set of data containing observations (or instances) whose category membership is known.
    - An example would be assigning a given email into "spam" or "non-spam" classes
    - or assigning a diagnosis to a given patient as described by
      - observed characteristics of the patient (gender, blood pressure, presence or absence of certain symptoms, etc.).
- *Classification is considered an instance of supervised learning,*
  - i.e. learning where a training set of correctly identified observations is available.
- Classifier
  - An algorithm that implements classification.

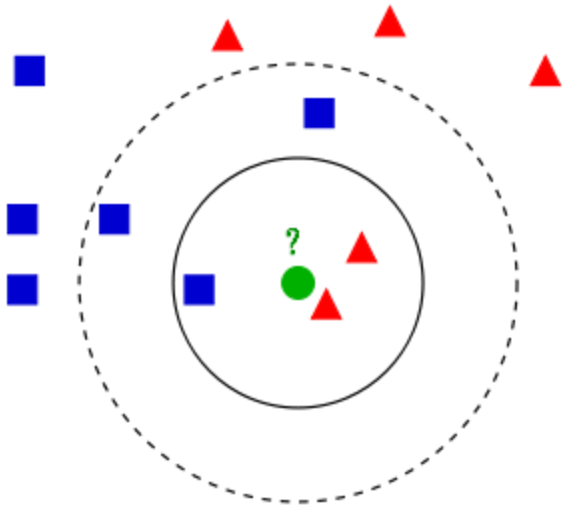
Supervised Learning - Classification

- An example implementation of this would be identifying digits from images.
- Let's dive into a case study on number classification.
- One of the many common approaches on the web to stopping computers from hacking into user accounts is the Captcha check:
  - a picture of text and numbers that the human user must decipher and enter into a form field before sending the form back to the web server.

A simple Captcha control can be used to prevent automated spam being sent through an online web form.

Supervised Learning
- Classification

- With the help of the *k-Nearest Neighbors*,
  - a simple yet powerful algorithm to categorize observations into classes
  - you can recognize digits from textual images.



- The test sample (green circle) should be classified either to
  - the first class of blue squares or to
  - the second class of red triangles.
- If $k = 3$ (*solid line circle*) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle.
- If $k = 5$ (*dashed line circle*) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

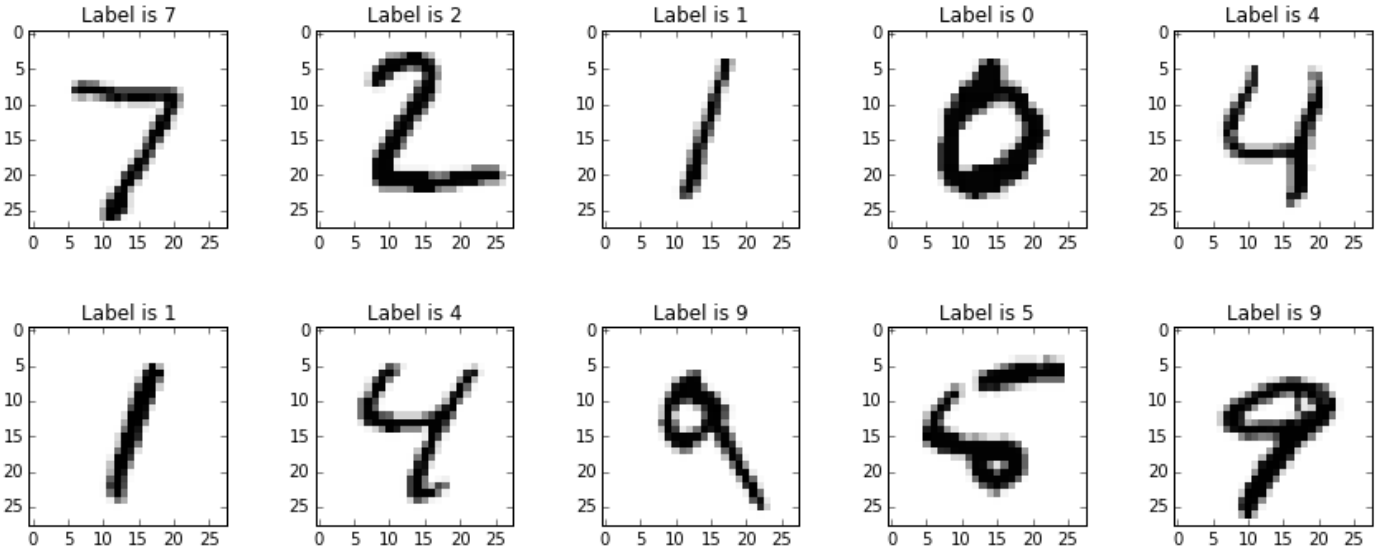
Supervised Learning
- Classification

- With the help of the *k-Nearest Neighbors* algorithm,
  - a simple yet powerful algorithm to categorize observations into classes
  - you can recognize digits from textual images.
- These images are unlike the Captcha checks many websites have in place to make sure you're not a computer trying to hack into the user accounts.
- Let's see how hard it is to let a computer recognize images of numbers.
- Our research goal is
  - to let a computer recognize images of numbers (step one of the data science process).
  - The data we'll be working on is the MNIST image data set.

Mixed National Institute of Standards and Technology database

Classification
- Retrieving Data

- The MNIST images can be found in the data sets package of Scikit-learn and are already normalized for you, so *we won't need much data preparation*.



- But let's first fetch our data as step two of the data science process, with the following listing.

```
from sklearn.datasets import load_digits
import pylab as pl
digits = load_digits()
```

Imports digits database.

Loads digits.

Classification
- Exploring Data

- Working with images isn't much different from working with other data sets.
- In the case of a gray image, you put a value in every matrix entry that depicts the gray value to be shown.
- The following code demonstrates this process and is step four of the data science process: data exploration.

```
pl.gray()
pl.matshow(digits.images[0])
pl.show()
digits.images[0]
```

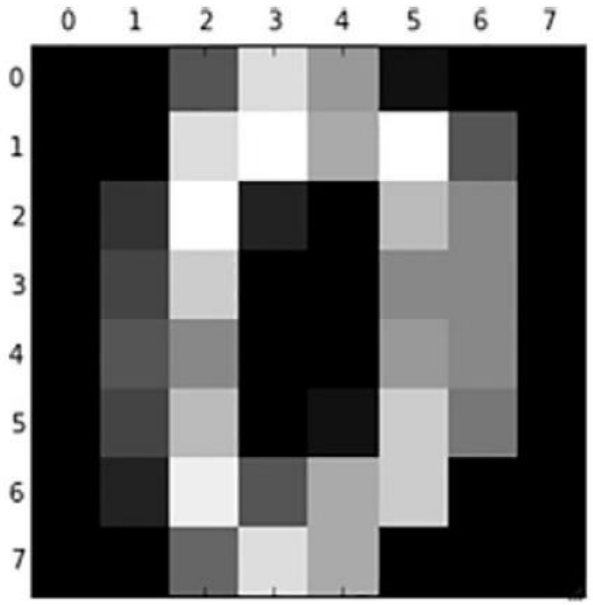
Turns image into gray-scale values.

Shows first images.

Shows the corresponding matrix.

Classification
- Exploring Data

- How does a blurry “0” image translate into a data matrix?

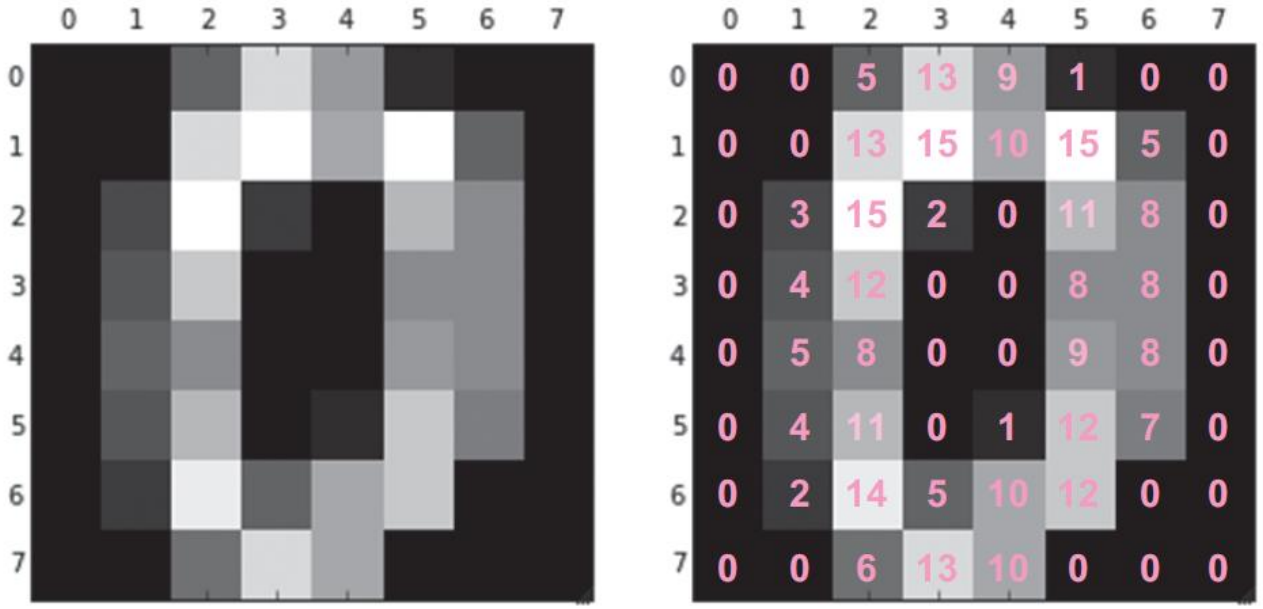



```
([[  0.,   0.,   5.,  13.,   9.,   1.,   0.,   0.],
  [  0.,   0.,  13.,  15.,  10.,  15.,   5.,   0.],
  [  0.,   3.,  15.,   2.,   0.,  11.,   8.,   0.],
  [  0.,   4.,  12.,   0.,   0.,   8.,   8.,   0.],
  [  0.,   5.,   8.,   0.,   0.,   9.,   8.,   0.],
  [  0.,   4.,  11.,   0.,   1.,  12.,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

Blurry grayscale representation of the number 0 with its corresponding matrix. The higher the number, the closer it is to white; the lower the number, the closer it is to black.

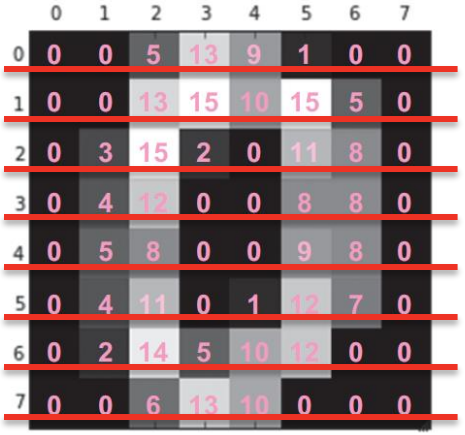
Classification
- Exploring Data

- The figure on the previous slide shows the actual code output, but perhaps the figure below can clarify this slightly,
  - because it shows how each element in the vector is a piece of the image.



We'll turn an image into something usable by the classifier by getting the grayscale value for each of its pixels (shown on the right) and putting those values in a list

Classification
- Exploring Data



- Easy so far, isn't it?
- There is, naturally, a little more work to do.
- The *k-nearest neighbors* classifier is expecting a list of values, but
  - pl.matshow() returns a two-dimensional array (a matrix) reflecting the shape of the image.
- To flatten it into a list, we need to call reshape() on digits.images.
- The net result will be a one-dimensional array that looks something like this:

```
array([[ 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.]])
```

Classification
- Modelling Data

- Now, the number of dimensions was reduced from two to one.
- From this point on, it's a standard classification problem,
  - which brings us to step five of the data science process: *model building*.
- When passing the contents of an image into the classifier,
  - we need to pass it a training data set,
  - So that it can start learning how to predict the numbers in the images.
- Each image is also labeled with the number it actually shows.
- This will build a model in memory of the most likely digit shown in an image given its grayscale values.

Classification
- Modelling Data

- Once the program has gone through the training set and built the model,
  - we can then pass it the test set of data to see how well it has learned to interpret the images using the model.
- The following listing shows how to implement these steps in code.

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
import pylab as plt

y = digits.target
n_samples = len(digits.images)
X = digits.images.reshape((n_samples, -1))
print(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
clf = KNeighborsClassifier(3)
fit = clf.fit(X_train, y_train)
predicted = fit.predict(X_test)
confusion_matrix(y_test, predicted)
```

Load packages

Step 1: Select target variable

Step 2: Prepare data. Reshape adapts the matrix form. This method could, for instance, turn a 10x10 matrix into 100 vectors.

Step 3: Split into test set and training set.

Step 4: Select a k-Nearest Neighbor classifier

Step 5: Fit the data

Step 6: Predict on unseen data

Step 7: Create confusion matrix

Classification
- Modelling Data

- The end result of this code is called a *confusion matrix*,
  - a two-dimensional array
  - shows how often the number predicted was the correct number on the main diagonal and also in the matrix entry (i,j), where j was predicted but the image showed i.
- As shown as follows, we can see that the model predicted the number 2 correctly 17 times (at coordinates 3,3), but also that the model predicted the number 8 15 times when it was actually the number 2 in the image (at 3,9).

```
array([[37,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 39,  0,  0,  0,  0,  1,  0,  3,  0],
       [ 0,  9, 17,  3,  0,  0,  0,  0, 15,  0],
       [ 0,  0,  0, 38,  0,  0,  0,  2,  5,  0],
       [ 0,  1,  0,  0, 27,  0,  2,  8,  0,  0],
       [ 0,  1,  0,  1,  0, 43,  0,  3,  0,  0],
       [ 0,  0,  0,  0,  0,  0, 52,  0,  0,  0],
       [ 0,  0,  0,  0,  1,  0,  0, 47,  0,  0],
       [ 0,  5,  0,  1,  0,  1,  0,  4, 37,  0],
       [ 0,  2,  0,  7,  1,  0,  0,  3,  7, 27]])
```

Confusion matrix produced by predicting what number is depicted by a blurry image

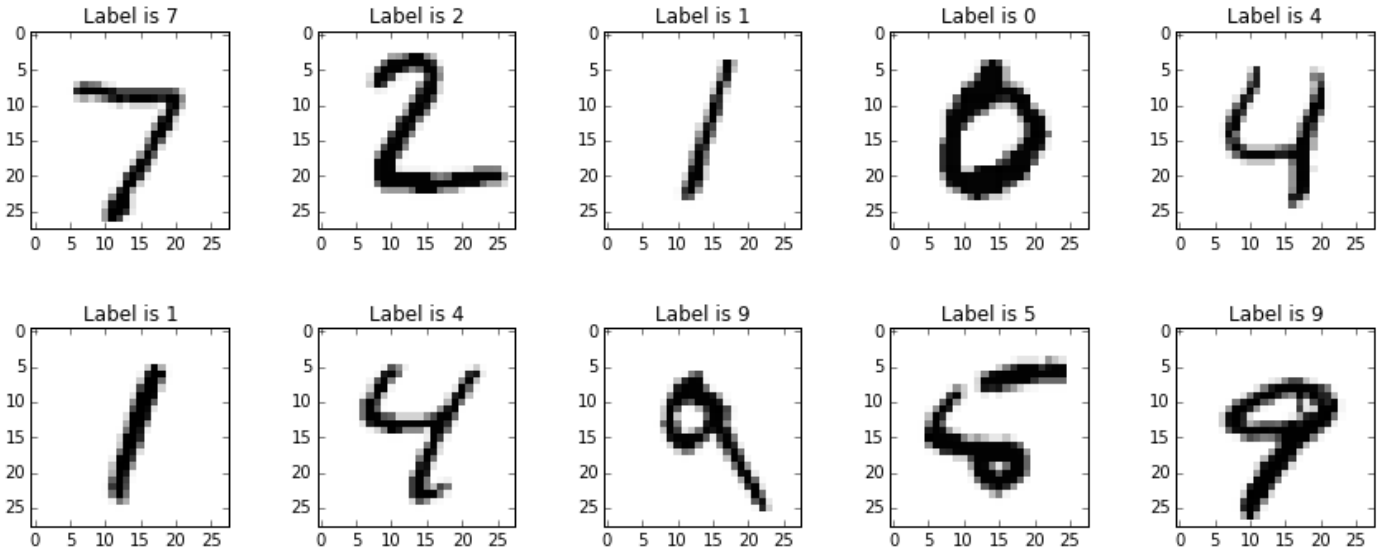
Classification
- Modelling Data

- From the confusion matrix,
  - we can deduce that for most images the predictions are quite accurate.
- In a good model, you'd expect
  - the sum of the numbers on the main diagonal of the matrix (also known as the matrix *trace*) to be very high compared to the sum of all matrix entries,
    - This indicates that the predictions were correct for the most part.

Classification

- Modelling Data: The Whole Story

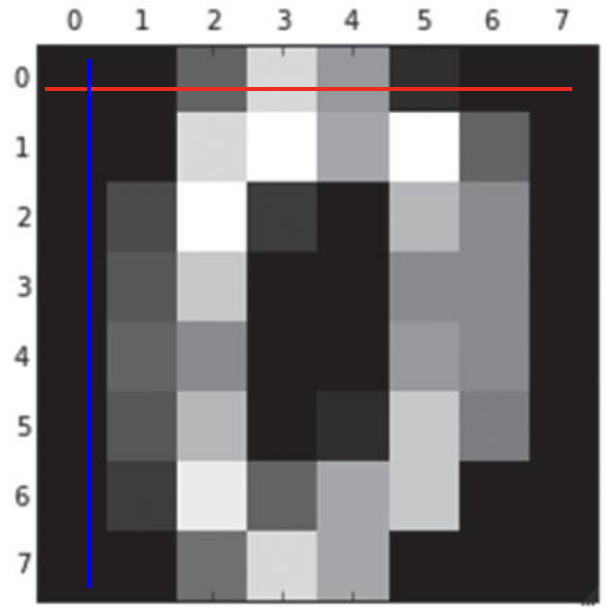
```
from sklearn.datasets import load_digits

digits = load_digits()
```

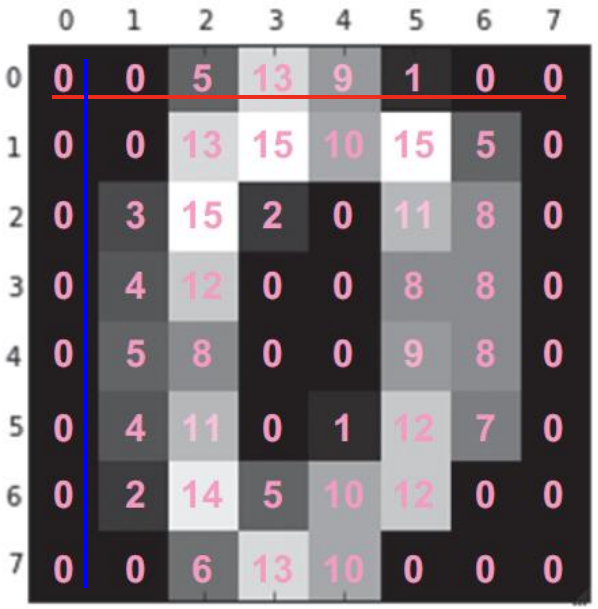


Classification

- Modelling Data: The Whole Story

```
[[  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.]])
```



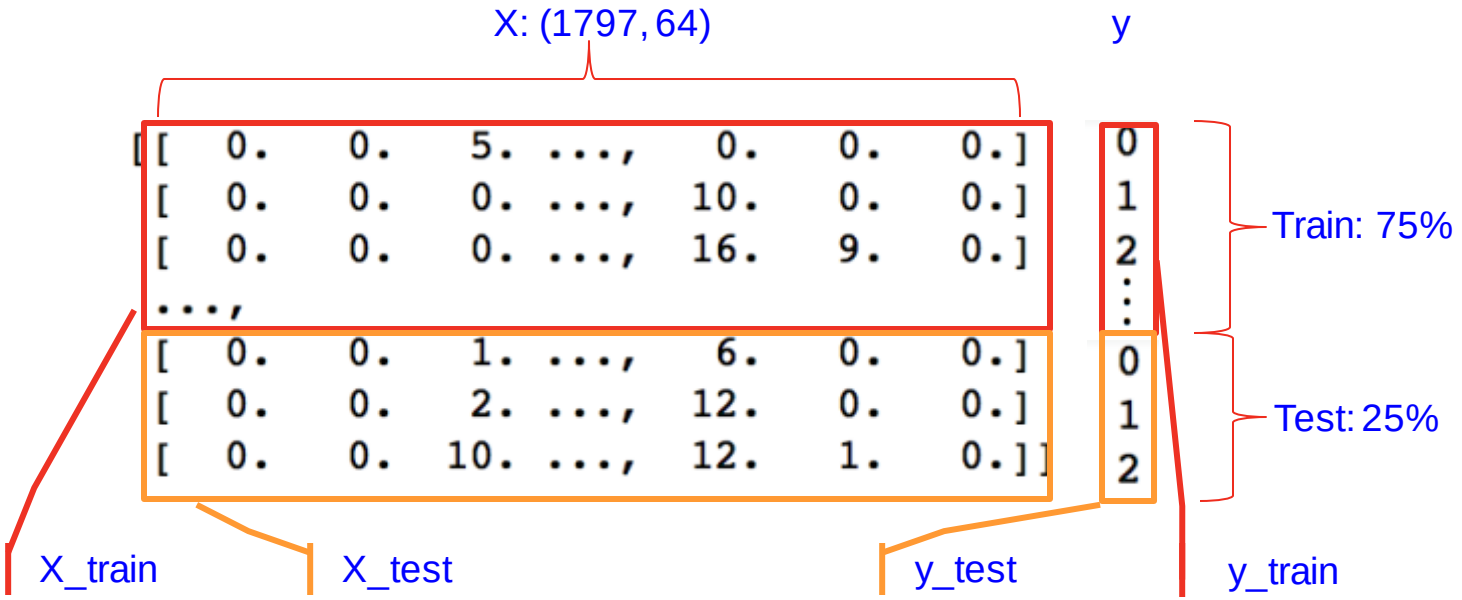


Classification

- Modelling Data: The Whole Story

```
X= digits.images.reshape((n_samples, -1))
y = digits.target
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

test_size = 0.25



Classification

- Modelling Data: The Whole Story

```
X= digits.images.reshape((n_samples, -1))
y = digits.target

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.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
clf = KNeighborsClassifier(3)
fit = clf.fit(X_train,y_train)
predicted = fit.predict(X_test)

confusion_matrix(y_test, predicted)
```

Select a k-Nearest Neighbor classifier

Fit the data

Predict on unseen data

Create confusion matrix

X_train

1st
2nd
3rd

```
[[  0.   0.   5. ...,   0.   0.   0.]
 [  0.   0.   0. ...,  10.   0.   0.]
 [  0.   0.   0. ...,  16.   9.   0.]
 ...,
 [  0.   0.   1. ...,   6.   0.   0.]
 [  0.   0.   2. ...,  12.   0.   0.]
 [  0.   0.  10. ...,  12.   1.   0.]]
```

X_test

y predicted

```
0
1
2
⋮
```

```
0
1
2
```

```
0
1
1
```

y_train

	0	1	2
0	1	0	0
1	0	1	0
2	0	✗	0

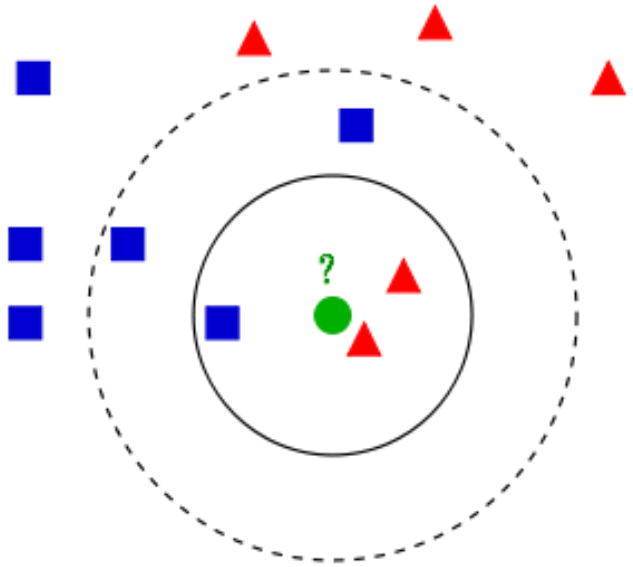
Classification error rate = 1/3

Classification

- *sklearn.neighbors.KNeighborsClassifier*

sklearn.neighbors.KNeighborsClassifier(*n_neighbors*, *weights*, *metric*, *p*)

- *n_neighbors:* int, optional (default = 5)
  - Number of neighbors to use.
  - The optimal choice of the value is highly data-dependent: in general
    - a larger value suppresses the effects of noise,
    - but makes the classification boundaries less distinct.



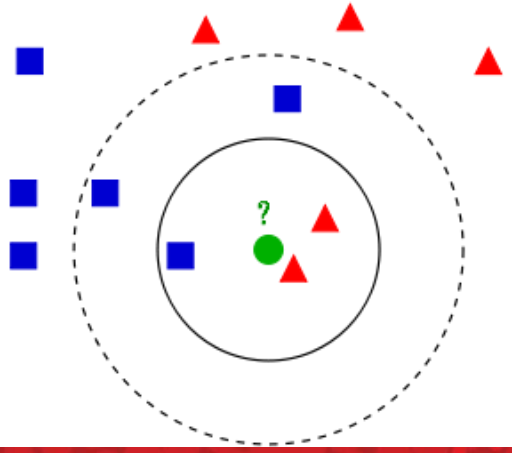
- The test sample (green circle) should be classified either to
  - the first class of blue squares or to
  - the second class of red triangles.
- $k = 3$: *solid line circle*
- $k = 5$: *dashed line circle*

Classification

- *sklearn.neighbors.KNeighborsClassifier*

sklearn.neighbors.KNeighborsClassifier(*n_neighbors*, *weights*, *metric*, *p*)

- *weights:* str or callable, optional (default = 'uniform')
  - weight function used in prediction. Possible values:
    - '*uniform*': uniform weights.
      - All points in each neighborhood are weighted equally.
    - '*distance*': weight points by the inverse of their distance.
      - Closer neighbors of a query point will have a greater influence than neighbors which are further away.



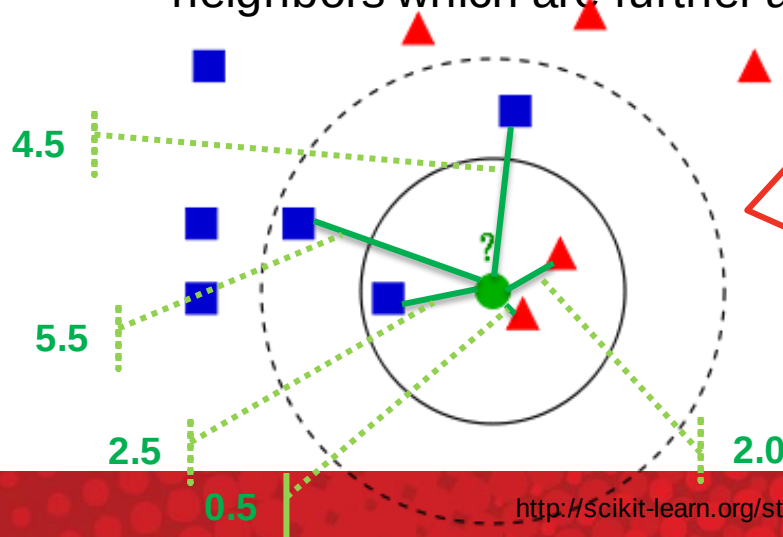
- If $k = 3$ (*solid line circle*) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle.
- If $k = 5$ (*dashed line circle*) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

Classification

- *sklearn.neighbors.KNeighborsClassifier*

sklearn.neighbors.KNeighborsClassifier(*n_neighbors*, *weights*, *metric*, *p*)

- *weights:* str or callable, optional (default = 'uniform')
  - weight function used in prediction. Possible values:
    - '*uniform*': uniform weights.
      - All points in each neighborhood are weighted equally.
    - '*distance*': weight points by the inverse of their distance.
      - Closer neighbors of a query point will have a greater influence than neighbors which are further away.



- If $k = 3$ (*solid line circle*)
  - Red triangle: 1/0.5 + 1/2 = 2.5
  - Blue square: 1/2.5 = 0.4
- If $k = 5$ (*dashed line circle*)
  - Red triangle: 1/0.5 + 1/2 = 2.5
  - Blue square:
    - 1/2.5 + 1/4.5 + 1/5.5 = 0.8040

Classification

- *sklearn.neighbors.KNeighborsClassifier*

sklearn.neighbors.KNeighborsClassifier(*n_neighbors*, *weights*, *metric*, *p*)

- *metric*: string or DistanceMetric object (default = '*minkowski*')
  - The distance metric.
  - The default metric is *minkowski*, and with *p=2* is equivalent to the standard *Euclidean* metric.

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

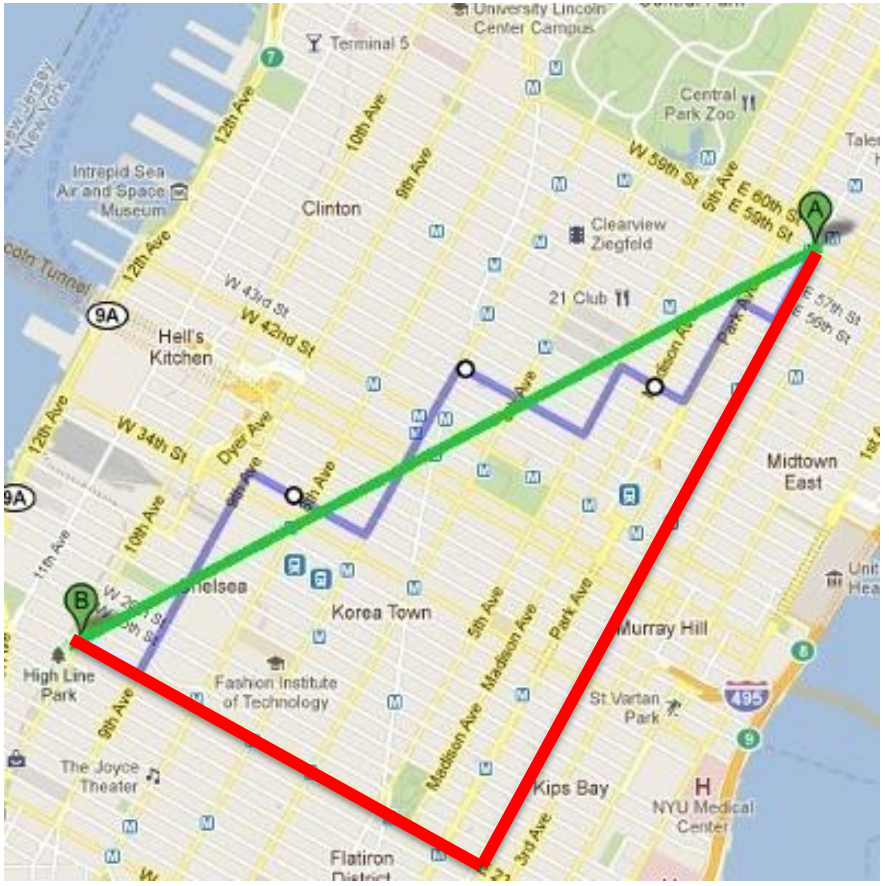
  - where $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n) \in \mathbb{R}^n$
  - See the documentation of the DistanceMetric class for a list of available metrics.
    - http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.DistanceMetric.html

Classification

- *sklearn.neighbors.KNeighborsClassifier*

sklearn.neighbors.KNeighborsClassifier(*n_neighbors*, *weights*, *metric*, *p*)

- *p*: integer, optional (default = 2)
  - Power parameter for the *Minkowski* metric.
  - When p = 1, this is equivalent to using *manhattan_distance* (*l1*), and
  - *euclidean_distance* (*l2*) for p = 2.
  - For arbitrary p, minkowski_distance (*l_p*) is used.



http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

https://en.wikipedia.org/wiki/Minkowski_distance

Classification
- How to select p?

- It depends on your data.
- For high dimensional vectors, a small p value might be better.
  - E.g. you might find that Manhattan ($p=1$) works better than the Euclidean ($p=2$) distance.
- The reason is:
  - Considering the case where the *minkowski* distance with $p = infinity$. Then, the distance is the highest difference between any two dimensions of your vectors.
  - We can see this doesn't make sense in many dimensions
    - Because we would be ignoring most of the dimensionality and measuring distance based on a single attribute.
- Thus reducing the exponent makes the other features play a bigger role in the distance calculation.
- The lower the p the less relevant a high difference in some given dimension will be.

https://www.quora.com/What-is-the-difference-between-Manhattan-and-Euclidean-distance-measures

Classification

- How to select *p*?

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

$$X = (x_1, x_2, \ldots, x_n) \quad Y = (y_1, y_2, \ldots, y_n)$$

		p=1			*p*=2			*p*=5		
$(x_1, x_2, \ldots$	X	1	2	3	1	2	3	1	2	3
$(y_1, y_2, \ldots$	Y	2	5	9	2	5	9	2	5	9
	$\lvert x_i - y_i \rvert$	1	3	6	1	3	6	1	3	6
	$\lvert x_i - y_i \rvert^p$	1	3	6	1	9	36	1	243	7776
	%	0.1	0.3	0.6	0.02	0.20	0.78	0.00	0.03	0.97
	$\sum_{i=1}^{n} \lvert x_i - y_i \rvert^p$	10			46			8020		
	$\left(\sum_{i=1}^{n} \lvert x_i - y_i \rvert^p\right)^{1/p}$	10			6.78			6.04		

Classification
- Modelling Data

- If we want to show our results in a more easily understandable way or
- we want to inspect several of the images and the predictions our program has made:
  - we can use the following code to display one next to the other.

```
images_and_predictions = list(zip(digits.images, fit.predict(X)))
for index, (image, prediction) in enumerate(images_and_predictions[:6]):
    plt.subplot(6, 3 ,index + 5)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Prediction: %i' % prediction)
plt.show()
```

Adds an extra subplot on a 6x3 plot grid. This code could be simplified as: plt.subplot (3, 2 ,index) but this looks visually more appealing.

Stores number image matrix and its prediction (as a number) together in array.

Loops through first 7 images.

Doesn't show an axis.

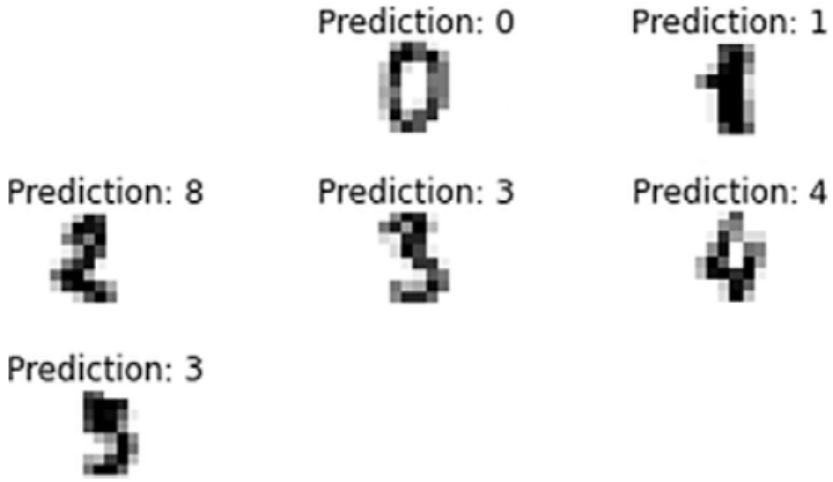
Shows image in grayscale.

Shows the predicted value as the title to the shown image.

Shows the full plot that is now populated with 6 subplots.

Classification
- Modelling Data

- The following figure shows
  - how all predictions seem to be correct except for the digit number 2, which it labels as 8.
- This is probably a reasonable mistake, since 2 shares some visual similarities with 8.
- The bottom left number is ambiguous, even to humans; is it a 5 or a 3? It's debatable, but the algorithm thinks it's a 3.



For each blurry image a number is predicted; only the number 2 is misinterpreted as 8. Then an ambiguous number is predicted to be 3 but it could as well be 5; even to human eyes this isn't clear.

Classification
- Modelling Data

- By discerning which images were misinterpreted, we can train the model further by
  - labeling them with the correct number they display and
  - feeding them back into the model as a new training set (step 5 of the data science process).
- This will make the model more accurate, so the cycle of learn, predict, correct continues and the predictions become more accurat
- This is a controlled data set we're using for the example.
- All the examples are the same size and they are all in 16 shades of gray.

Classification
- Modelling Data

- Expand that up to the variable size images of variable length strings of variable shades of alphanumeric characters shown in the Captcha control
  - you can appreciate why a model accurate enough to predict any Captcha image doesn't exist yet.

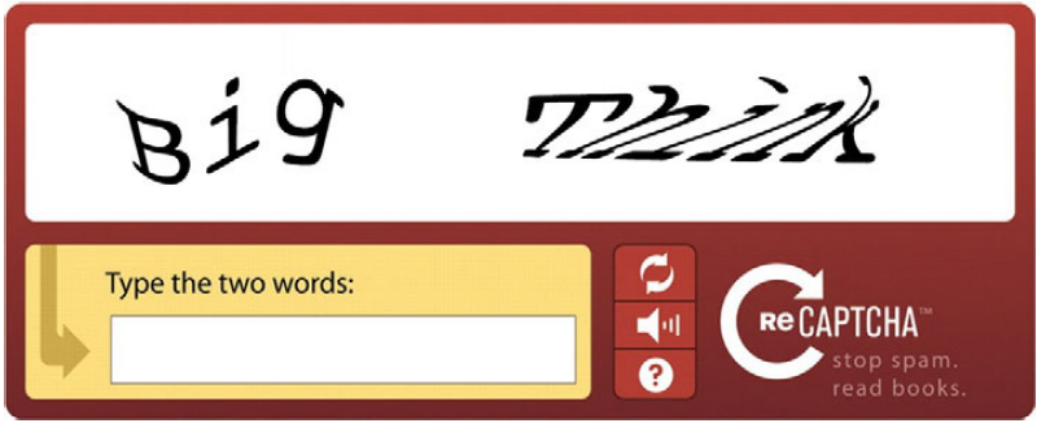


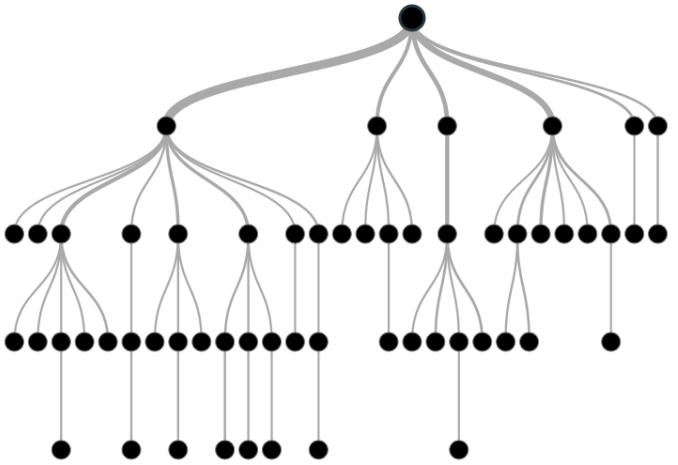
Figure 3.3 A simple Captcha control can be used to prevent automated spam being sent through an online web form.

Practical Data Science – COSC2670

PART 3: CLASSIFICATION II

Classification
- Decision Tree

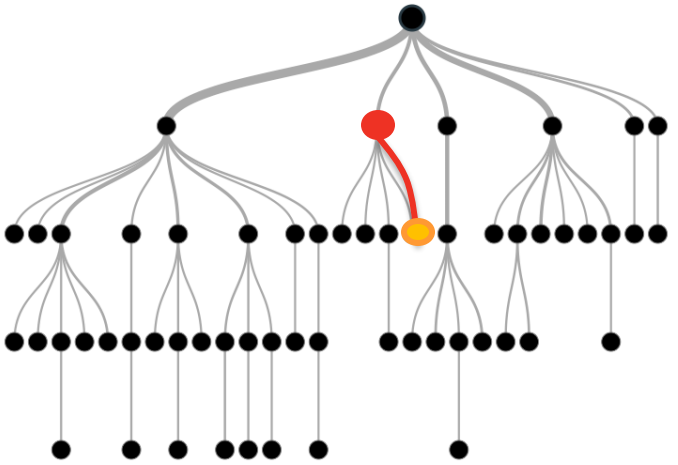
- What is a decision tree?
  - A type of supervised learning algorithm that is mostly used in classification problems.
  - It works for both *categorical and continuous* input and output variables.
  - It splits the population or sample into two or more homogeneous sets (or sub-populations)
    - based on most significant splitter / differentiator in input variables.



https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/

Classification
- Decision Tree

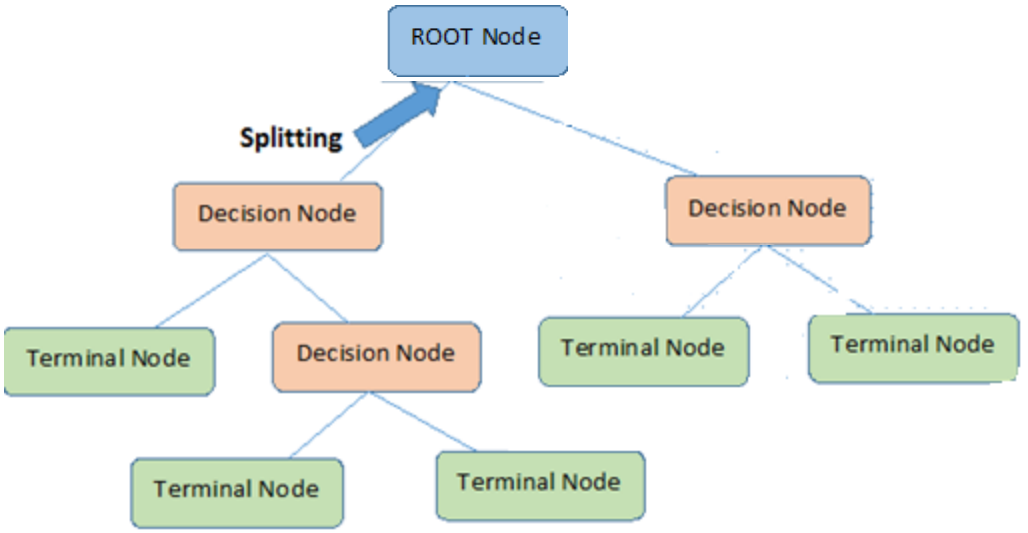
- What is a decision tree?
  - A decision tree is a flowchart-like structure, in which
    - each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails),
    - each branch represents the outcome of the test, and
    - each leaf node represents a class label (decision taken after computing all attributes).
  - *The paths from root to leaf represent classification rules.*



https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/

Terminology Related to Decision Tree

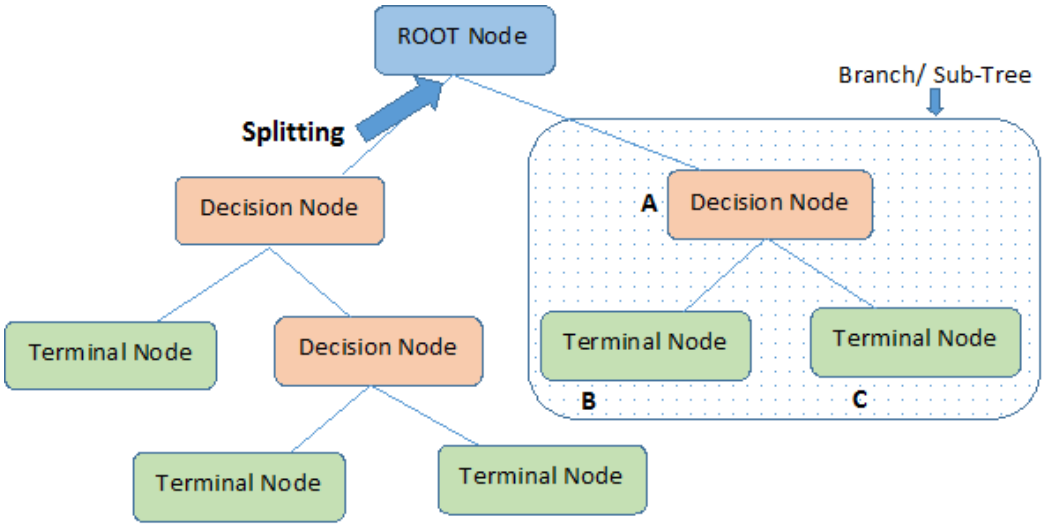
- *Root Node:* It represents entire population or sample and this further gets divided into two or more homogeneous sets.
- *Splitting:* A process of dividing a node into two or more sub-nodes.
- *Decision Node:* When a sub-node splits into further sub-nodes, then it is called a decision node.
- *Leaf/ Terminal Node:* Nodes that do not split are called Leaf or Terminal nodes.



Note:- A is parent node of B and C.

Terminology Related to Decision Tree

- *Branch / Sub-Tree:* A sub section of the entire tree is called branch or sub-tree.
- *Parent and Child Node:* A node which is divided into sub-nodes is called a parent node of sub-nodes, where the sub-nodes are the children of the parent node.



Decision Tree
- An Example

- Let's say we have a sample of 30 students with three variables
  - *Gender* (Boy/ Girl),
  - *Class* (IX/X),
  - *Height* (5 to 6 ft).

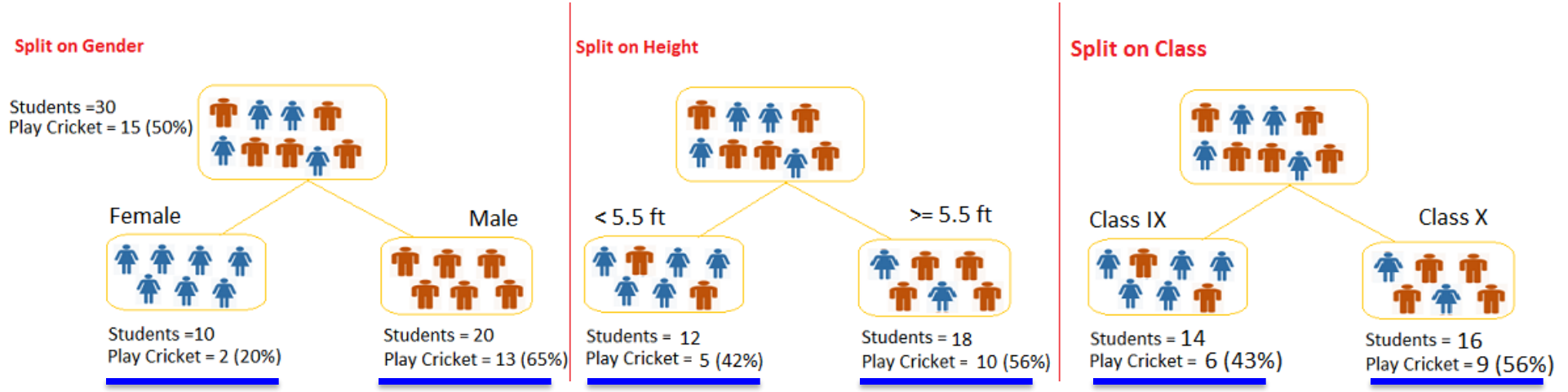
	Gender	Class	Height	Play Cricket
0	F	I	5.6	Yes
1	M	II	5.5	No
2	..	..	..	..
	M	I	5.7	?

- 15 out of these 30 play cricket in leisure time.
- We want to create a model to predict
  - who will play cricket during their leisure time?
    - Specifically, given a user's gender, class and height attributes, the model should predict whether the user will play or not.
- In this problem, we need to segregate students who play cricket in their leisure time, based on the most highly significant input variable among all three.

Decision Tree
- An Example

Homogeneous in terms of playing cricket or not.

- This is where a decision tree helps, it will
  - segregate the students based on all values of three variable and
  - identify the variable, which creates the best *homogeneous* sets of students (which are heterogeneous to each other).
- In the snapshot below, you can see that variable *Gender* is able to identify the best homogeneous sets compared to the other two variables.



Decision Tree

- As mentioned above, a decision tree identifies
  - the most significant variable and
  - the value of the variable that gives the best homogeneous sets of population.
- Now the question which arises is:
  - How should the variable and the split be identified?
  - To do this, decision tree uses various algorithms, which we will discuss in the following section.

Where to split?

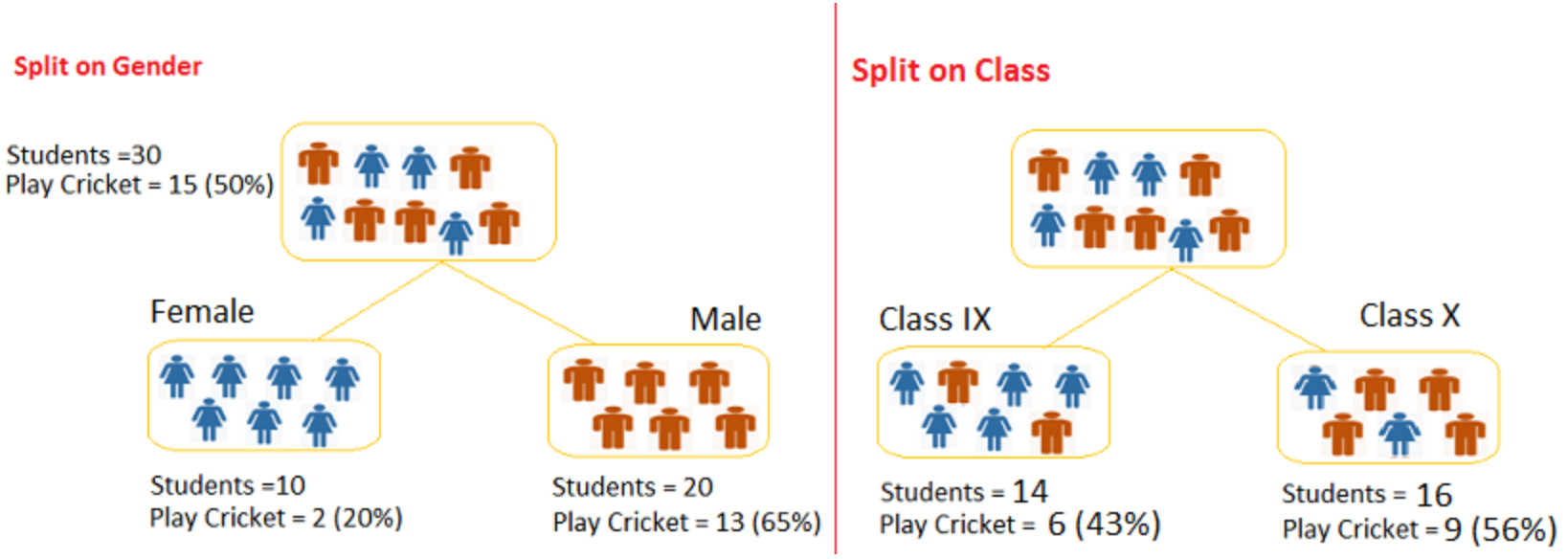
- The decision of making strategic splits heavily affects a tree's accuracy.
- Decision trees use multiple algorithms to decide whether to split a node into two or more sub-nodes.
- The creation of sub-nodes increases the homogeneity of resultant sub-nodes.
- The *purity* of the node increases with respect to the target variable.
- A decision tree
  - splits the nodes on all available variables and
  - then selects the split which results in most homogeneous sub-nodes.
- The algorithm selection is also based on the type of target variables.
- Let's look at one of the most commonly used algorithms in decision tree
  - *Gini index*

Where to split?
- Gini Index

- The Gini index says, if we select two items from a population at random then they must be of the same class, and the probability for this is 1 if the population is pure.
  - This works with categorical target variables
    - such as “Success (play)” or “Failure (not play)”.
  - It performs only Binary splits
  - The higher the value of Gini, the higher the homogeneity.
  - A decision tree uses the Gini method to create binary splits.
- Gini index calculation
  - Calculate Gini for sub-nodes, using the formula:
    - sum of squares of probability for success and failure (p^2+q^2).
  - Calculate Gini for split using the weighted Gini score of each node of that split

Where to split?
- Gini Index

- Referring to the previous example, where we want to segregate the students based on the target variable of *playing cricket or not*.
- In the snapshot below, we split the population using two input variables *Gender* and *Class*.
- Now, we want to identify *which split produces more homogeneous sub-nodes* using the Gini index.



Where to split?
- Gini Index

Gini for sub-node Female =
(0.2)*(0.2)+(0.8)*(0.8)=0.68

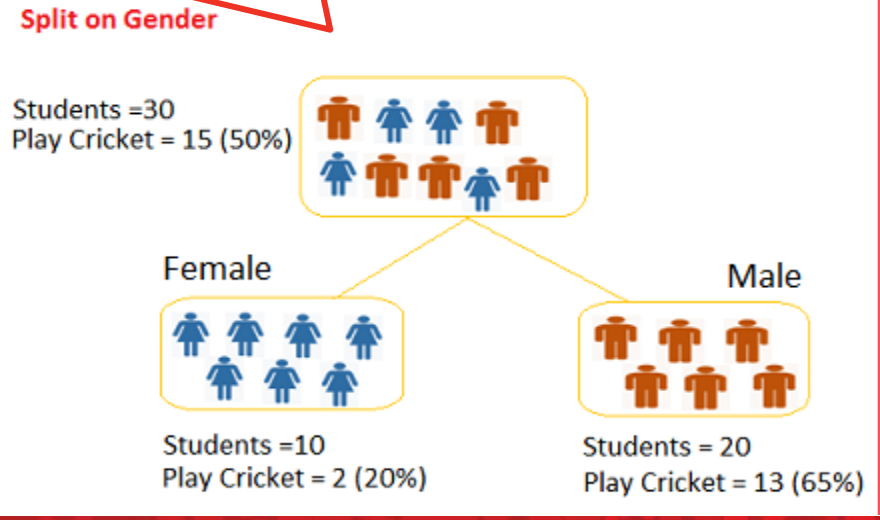
Gini for sub-node Male =
(0.65)*(0.65)+(0.35)*(0.35)=0.55

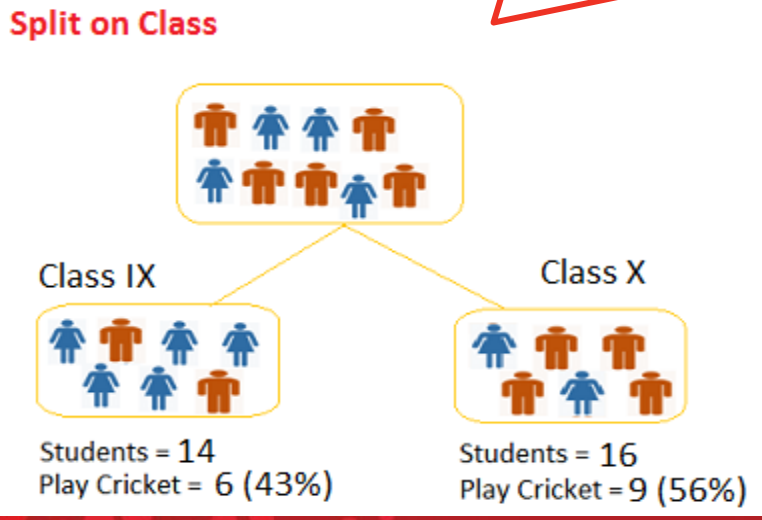
Calculate weighted Gini for Split Gender =
(10/30)*0.68+(20/30)*0.55 = **0.59**

Gini for sub-node Class IX =
(0.43)*(0.43)+(0.57)*(0.57)=0.51

Gini for sub-node Class X =
(0.56)*(0.56)+(0.44)*(0.44)=0.51

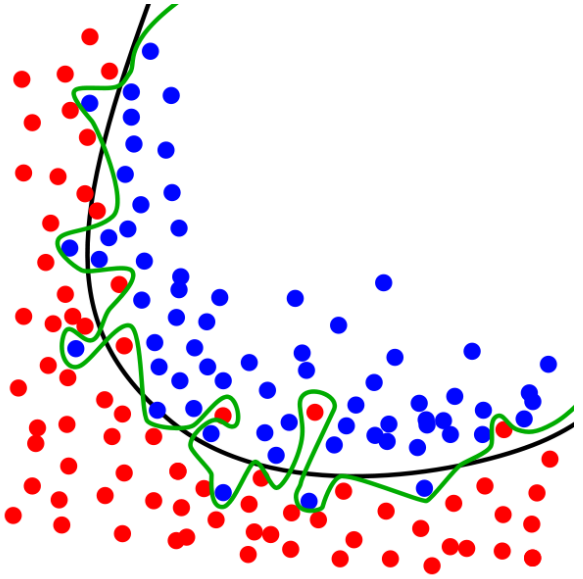
Calculate weighted Gini for Split Class =
(14/30)*0.51+(16/30)*0.51 = **0.51**





Key Parameters in Decision Tree

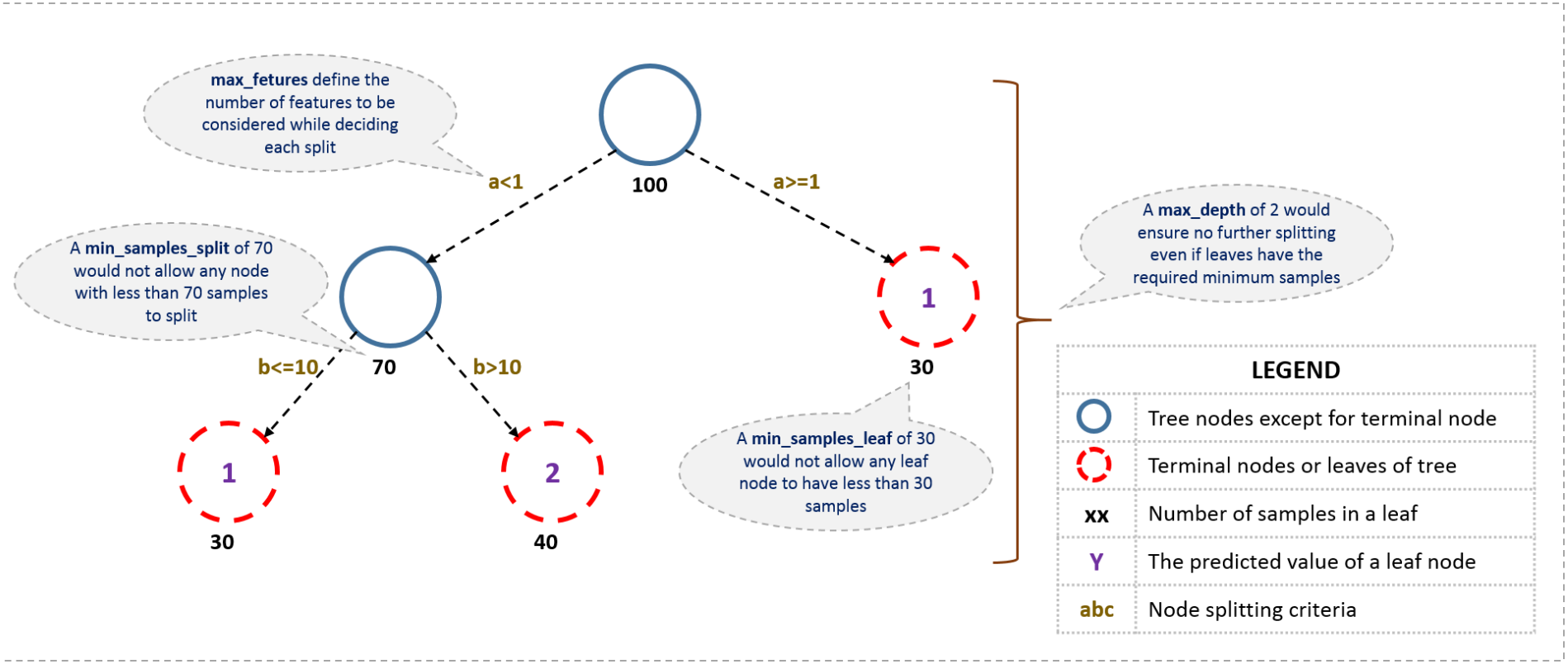
- *Overfitting* is one of the key challenges faced while modelling decision trees.
  - Overfitting occurs when a model is excessively complex, such as having too many parameters relative to the number of observations.



- If there is no limit set of a decision tree, it will give you 100% accuracy on training set
  - because in the worst case it will end up making 1 leaf for each observation.
- Thus, preventing overfitting is pivotal while modelling a decision tree and it can be done by
  - *Setting constraints on tree size*.

Setting Constraints on Tree Size

- This can be done by using various parameters which are used to define a tree.
  - First, lets look at the general structure of a decision tree:



https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/

Setting Constraints on Tree Size

- The parameters described below are irrespective of tool.
- It is important to understand the role of parameters used in tree modelling.
- These parameters are available in *sklearn.treeDecisionTreeClassifier*.
- Minimum samples for a node split
  - Defines the minimum number of samples (or observations) which are required in a node to be considered for splitting.
  - Used to control overfitting
    - Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.
    - Too high values can lead to *under-fitting,* therefore it should be tuned using *cross-validation*.

Setting Constraints on Tree Size

- *Minimum samples for a terminal node (leaf)*
  - Defines the minimum samples (or observations) required in a terminal node or leaf.
  - Used to *control overfitting* similar to *min_samples_split*.
  - Generally lower values should be chosen for imbalanced class problems, because the regions in which the minority class will be in majority will be very small.
- *Maximum depth of tree (vertical depth)*
  - The maximum depth of a tree.
  - Used to *control overfitting* as higher depth will allow model to learn relations very specific to a particular sample.
  - Should be tuned using cross-validation.

Setting Constraints on Tree Size

- *Maximum number of terminal nodes*
  - The maximum number of terminal nodes or leaves in a tree.
  - Can be defined in place of max_depth,
    - As when binary trees are created, a depth of 'n' would produce a maximum of 2^n leaves.
- *Maximum features to consider for split*
  - The number of features to consider while searching for a best split.
    - These will be randomly selected.
  - As a rule-of-thumb, the square root of the total number of features works well, but we should check up to 30-40% of the total number of features.
  - Higher values can lead to overfitting, but that can vary from case to case.

Decision Tree in *sklearn*

sklearn.tree.DecisionTreeClassifier(criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, max_features=None, max_leaf_nodes=None)

- *criterion*: string, optional (default="gini")
  - The function to measure the quality of a split. Supported criteria are "*gini*" for the Gini impurity and "*entropy*" for the information gain.
- *max_features*: int, float, string or None, optional (default=None)
  - The number of features to consider when looking for the best split:
    - If int, then consider max_features features at each split.
    - If float, then max_features is a percentage and int(max_features * n_features) features are considered at each split.
    - If "*auto*", then max_features=sqrt(n_features).
    - If "*sqrt*", then max_features=sqrt(n_features).
    - If "*log2*", then max_features=log2(n_features).
    - If None, then max_features=n_features.

Decision Tree in *sklearn*

sklearn.tree.DecisionTreeClassifier(criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, max_features=None, max_leaf_nodes=None)

- *max_depth*: int or None, optional (default=None)
  - The maximum depth of the tree.
  - If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- *min_samples_split* : int, float, optional (default=2)
  - The minimum number of samples required to split an internal node:
    - If int, then consider min_samples_split as the minimum number.
    - If float, then min_samples_split is a percentage and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.

Decision Tree in *sklearn*

sklearn.tree.DecisionTreeClassifier(criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, max_features=None, max_leaf_nodes=None)

- *min_samples_leaf* : int, float, optional (default=1)
  - The minimum number of samples required to be at a leaf node:
  - If int, then consider min_samples_leaf as the minimum number.
  - If float, then min_samples_leaf is a percentage and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.

Decision Tree
- Iris Dataset

```
from sklearn.datasets import load_iris

iris = load_iris()

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, random_state=0)
clf = DecisionTreeClassifier()
fit = clf.fit(X_train, y_train)
y_pre = fit.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pre)
print(cm)

from sklearn.metrics import classification_report
print(classification_report(y_test, y_pre))
```

Load datasets

Load packages

Split into test set and training set

Select decision tree classifier

Fit the data

Make prediction on test data

Measure confusion matrix

Classification precision/recall/f1-sore

```
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	13
1	1.00	0.94	0.97	16
2	0.90	1.00	0.95	9
avg / total	0.98	0.97	0.97	38

Decision Tree
- Iris Dataset

- *Precision*
  - The fraction of correctly predicted instances
- *Recall*
  - The fraction of relevant instances that are successfully predicted.
- *F1-score*
  - Combines precision and recall
  - Is the harmonic mean of precision and recall

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

- *Support*

predicted

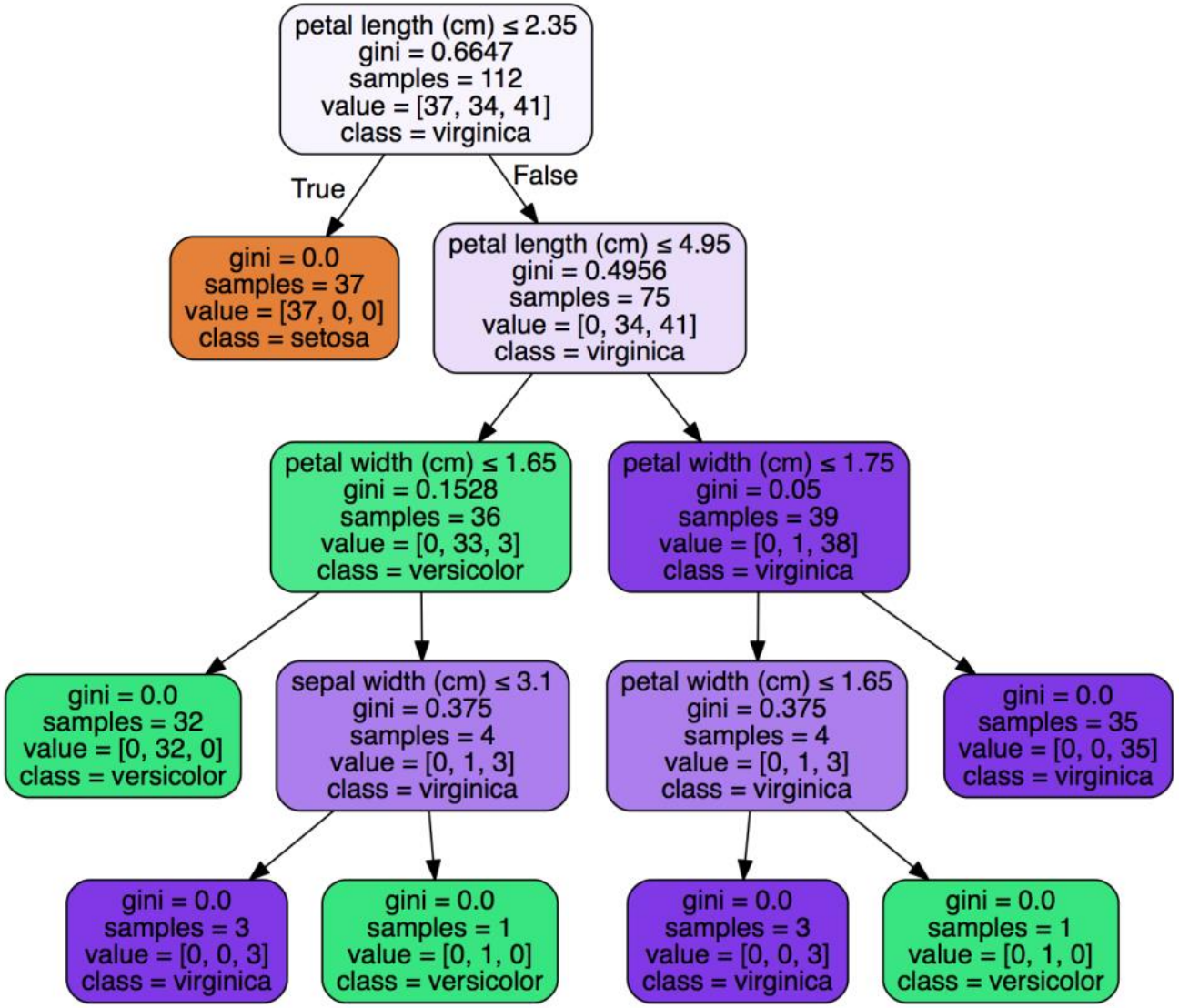
```
          0   1   2
0  [[ 13   0   0 ]
1    [  0  15   1 ]
2    [  0   0   9 ]]
```

13/(13+0+0) = 1.00

15/(0+15+1) = 0.94

	precision	recall	f1-score	support
0	1.00	1.00	1.00	13
1	1.00	0.94	0.97	16
2	0.90	1.00	0.95	9
avg / total	0.98	0.97	0.97	38

Tree Visualisation



Tree Visualisation

```
from sklearn import tree

with open("iris_new.dot", 'w') as f:
    f = tree.export_graphviz(clf, out_file=f, feature_names=iris.feature_names,
                             class_names=iris.target_names, filled=True, rounded=True, special_characters=True)
```

- *clf* : decision tree classifier
  - The decision tree to be
- *out_file* : file object or string, optional (default='tree.dot')exported to GraphViz.
- *feature_names* : list of strings, optional (default=None)
  - Names of each of the features.
- *class_names* : list of strings, bool or None, optional (default=None)
  - Names of each of the target classes in ascending numerical order.
- *filled :* bool, optional (default=False)
  - When set to True, paint nodes to indicate majority class for classification, extremity of values for regression, or purity of node for multi-output.

Tree Visualisation

```
from sklearn import tree

with open("iris_new.dot", 'w') as f:
    f = tree.export_graphviz(clf, out_file=f, feature_names=iris.feature_names,
                             class_names=iris.target_names, filled=True, rounded=True, special_characters=True)
```

- *rounded* : bool, optional (default=False)
  - When set to True, draw node boxes with rounded corners and use Helvetica fonts instead of Times-Roman.
- *special_characters* : bool, optional (default=False)
  - When set to False, ignore special characters for PostScript compatibility.

- Graphviz
  - http://www.graphviz.org/
- Online version
  - https://dreampuf.github.io/GraphvizOnline/

Advantages

- *Easy to Understand*:
  - Decision tree output is very easy to understand, even for people from a non-analytical background.
  - Does *not require any statistical knowledge* to read and interpret them.
  - *Graphical representation* is very intuitive and users can easily relate their hypothesis.
- *Less data cleaning required*:
  - Requires less data cleaning compared to some other modelling techniques.
  - *Not influenced by outliers and missing values* to a fair extent.
- *Data type is not a constraint*:
  - Can handle both numerical and categorical variables.
- *Non Parametric Method*: Decision tree is considered to be a non-parametric method.
  - This means that decision trees have no assumptions about the *space distribution and the classifier structure*.

Advantages

- *Useful in Data exploration*:
  - Decision tree is one of the *fastest* ways to identify the most significant variables, and the relation between two or more variables.
  - With the help of decision trees, we can create new variables / features that have better power to predict target variable.
  - It can also be used in *data exploration* stage.
    - For example, we are working on a problem where we have information available in hundreds of variables; a decision tree will help to identify the most significant variables.

Disadvantage

- *Overfitting*:
  - Overfitting is one of the most important practical difficulties for decision tree models.
  - This problem gets solved by setting constraints on *model parameters and other methods*.
- *Not fit for continuous variables*:
  - While working with continuous numerical variables, a decision tree looses information when it categorizes variables in different categories.

References and Further Reading

- A. Boschetti and L. Massaron, *Python Data Science Essentials*, Chapters 4 and 6
- D. Cielen and A. Meysman and M. Ali, *Introducing Data Science*, Chapter 3

Data
Science

Thanks!