강의자료 링크: http://bit.ly/hgu_mlcamp_2019summer

머신러닝을 활용한 리얼월드 데이터 분석

Machine Learning with Real World Data

홍참길

charmgil@handong.edu

한동대학교 HANDONG GLOBAL UNIVERSITY

July 17, 2019 @ 전산전자공학부 여름 학기 전공 캠프

Camp Outline

	Day 1	Day 2	Day 3
13:00	Introduction Getting Started with Python	Decision Trees	Logistic Regression
14:00	Hands-on Session	Hands-on Session	Hands-on Session
15:00	*k*-Nearest Neighbors	Visualization using Matplotlib	Data Analytics Practicum
16:00	Hands-on Session	Hands-on Session	

Camp Outline

- Camp objectives
  - To learn how to handle real world data and apply ML techniques on them
  - To understand some machine learning ideas and methodologies
    - *k*-Nearest Neighbors
    - Decision Trees
    - Logistic Regression
  - To obtain hands-on experiences with ML models and algorithms

- Target audiences
  - 2nd year CS/CE students who have just taken Data Structures
    - Not assuming a thick concept of math/stat knowledge
  - Requirements
    - General knowledge on data structures (with C or C++)
    - Strong motivation

IFU

Review: Introduction

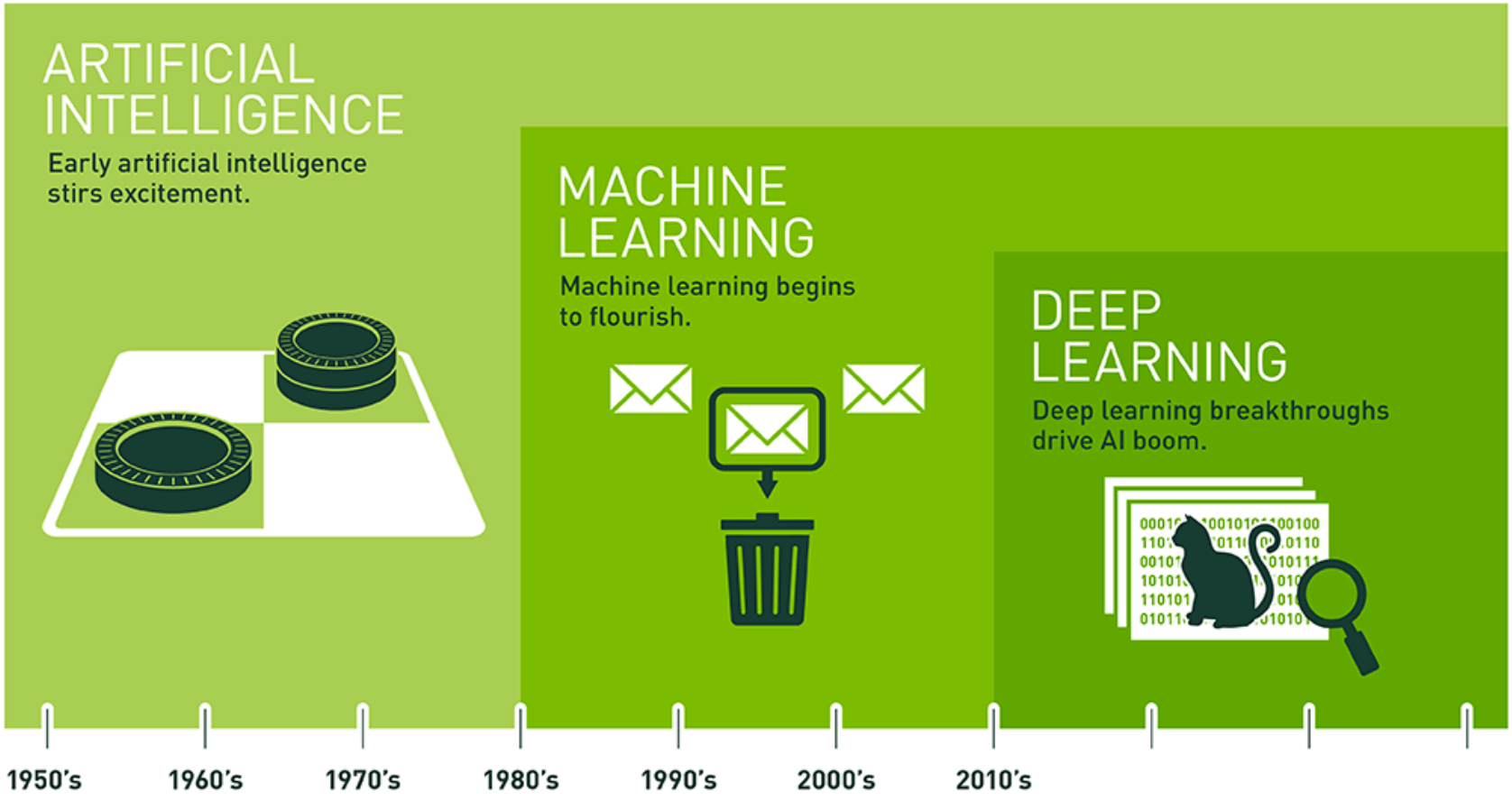
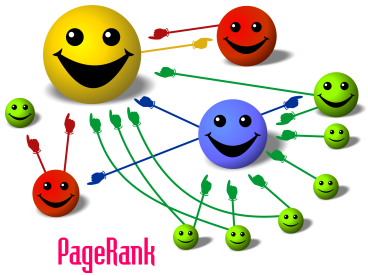
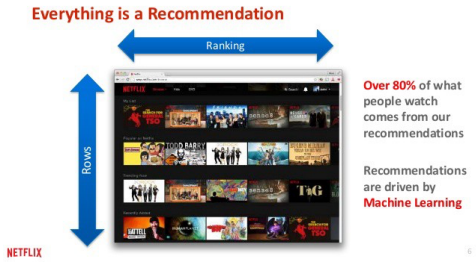


Image source: https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/

Review: Introduction

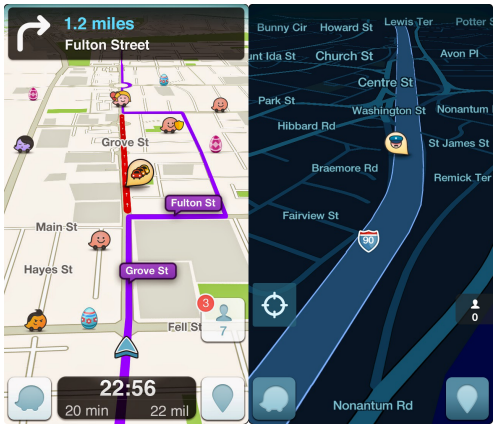
- ML in our life





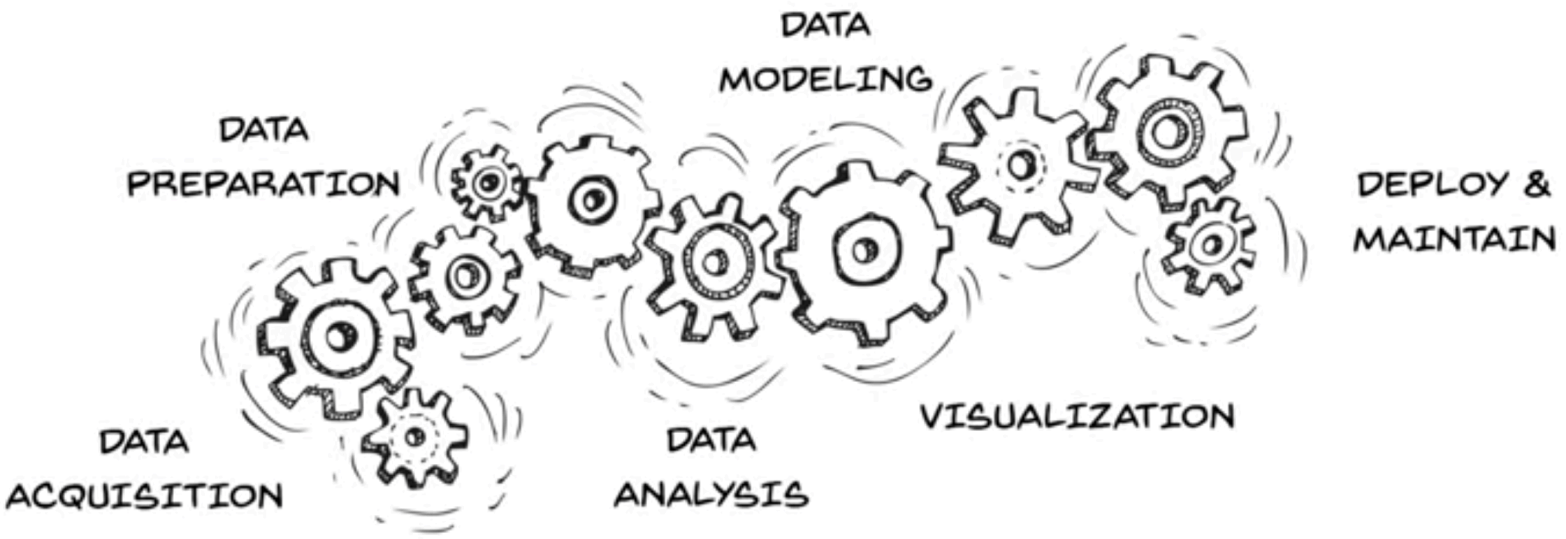
Over 80% of what people watch comes from our recommendations

Recommendations are driven by Machine Learning



IFU

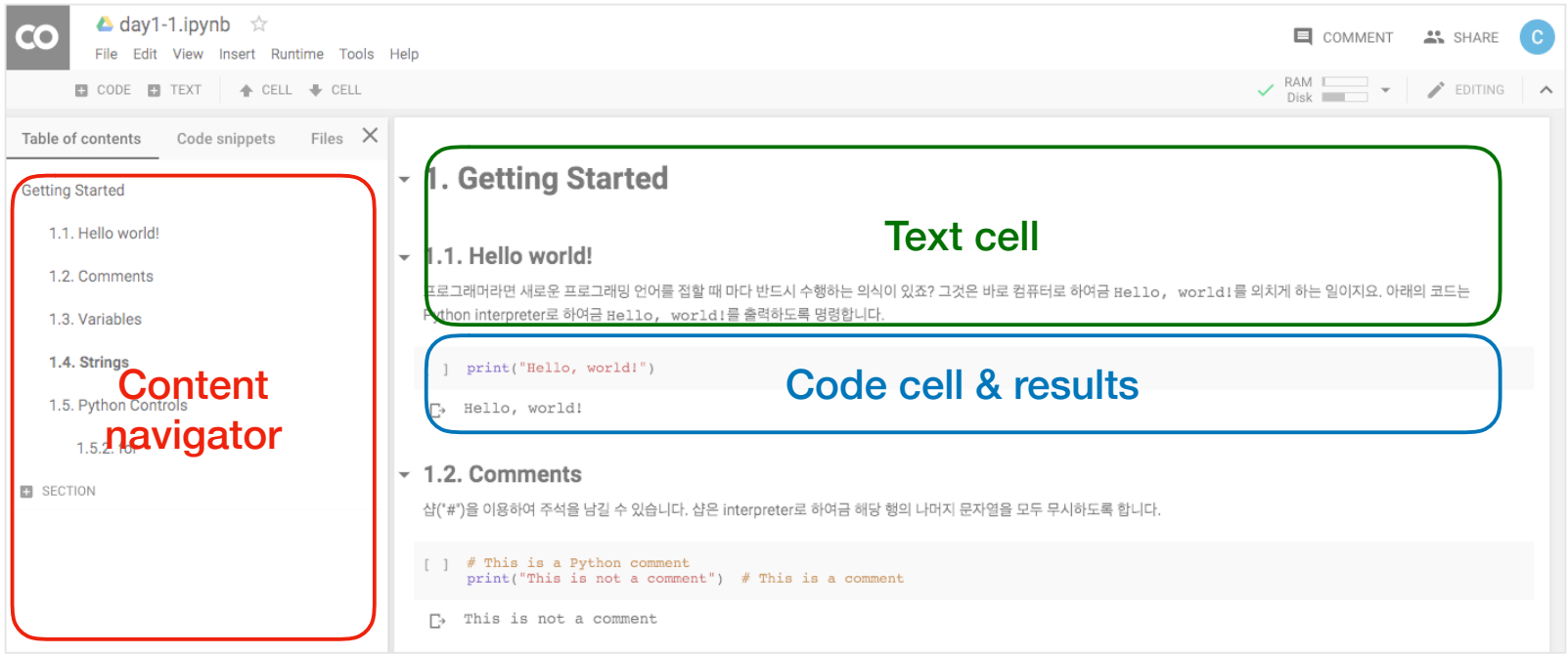
Review: Common ML Workflow



- Phase 1: Acquisition and Preparation
- Phase 2: Analysis, Modeling, and Visualization
- Phase 3: Deployment and Maintenance

Review: Python ABC

- How to use Python?
  - Google Colaboratory (Jupyter Notebook)



Review: Machine Learning Taxonomy

- Unsupervised learning
- Supervised learning
- Semi-supervised learning (reinforcement learning)

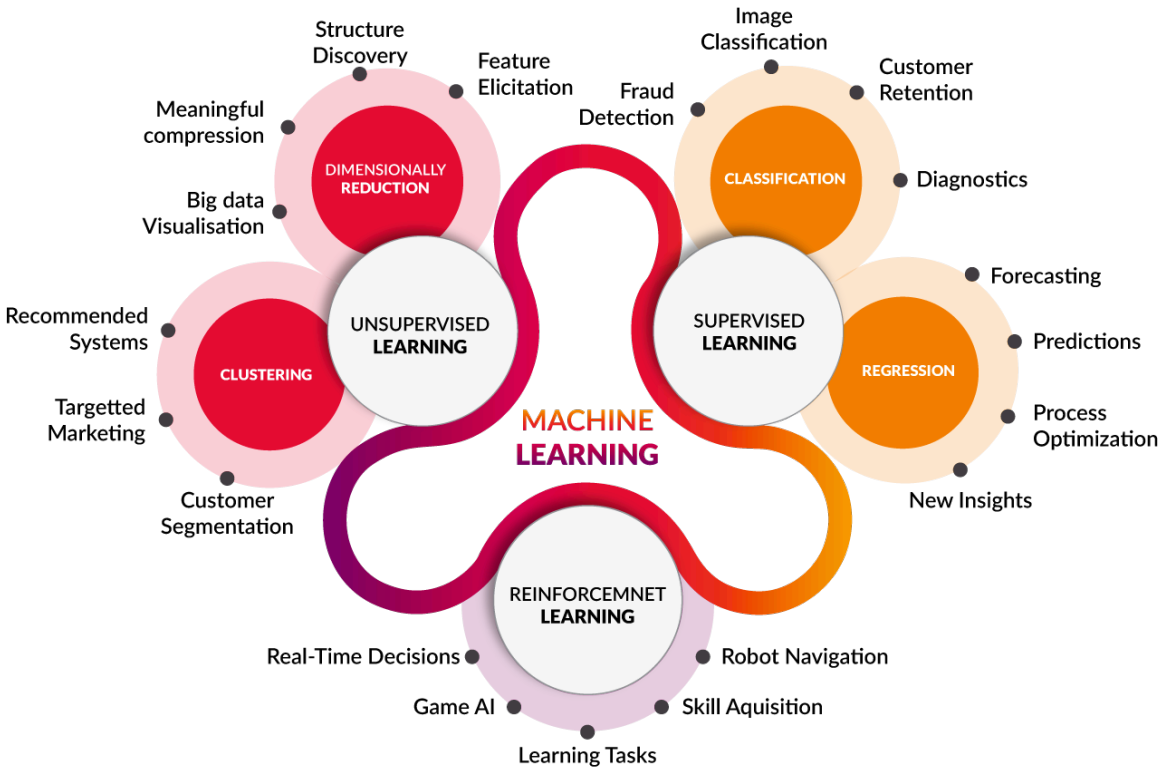


Image source: http://www.cognub.com/index.php/cognitive-platform/

Machine Learning Taxonomy

- Unsupervised learning
  - Draw inferences from data without labels (*there are no correct answers*)
  - To discover unknown data patterns and internal structure
  - *E.g.*, Clustering, outlier detection, autoencoders, self-organizing map, …

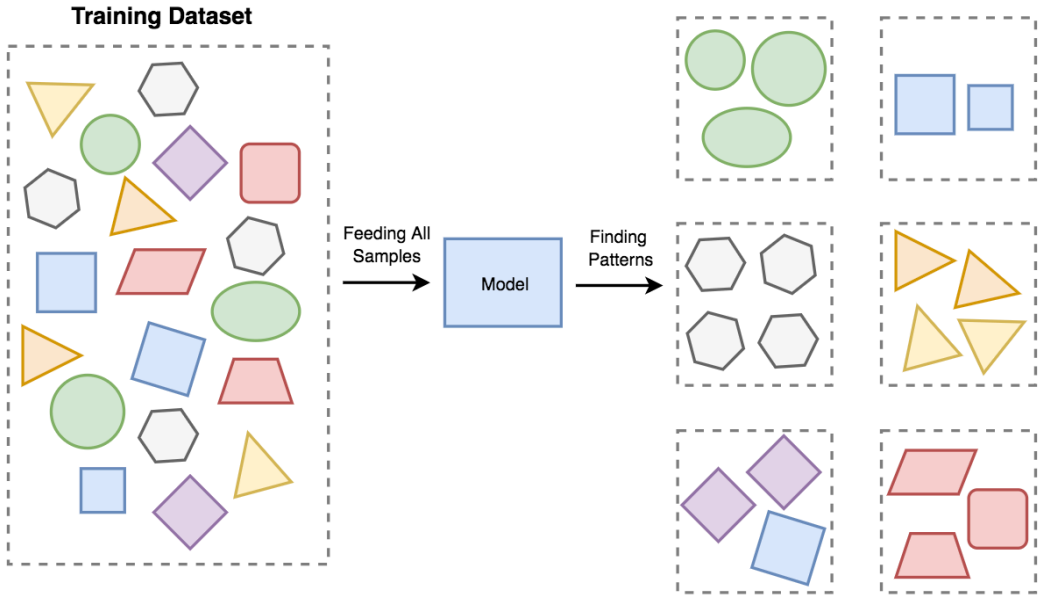


Image source: https://towardsdatascience.com/coding-deep-learning-for-beginners-types-of-machine-learning-b9e651e1ed9d

Machine Learning Taxonomy

- Supervised learning
  - Build predictive models based on data with labels (input-output pairs)
  - *E.g.*, classification, regression
    - Classification - the outcome is discrete (binary or categorical)
      - *E.g.*, Is this person healthy or with a disease?
    - Regression - the outcome is real or continuous number
      - *E.g.*, Predict one's life expectancy, based on his/her genetic info

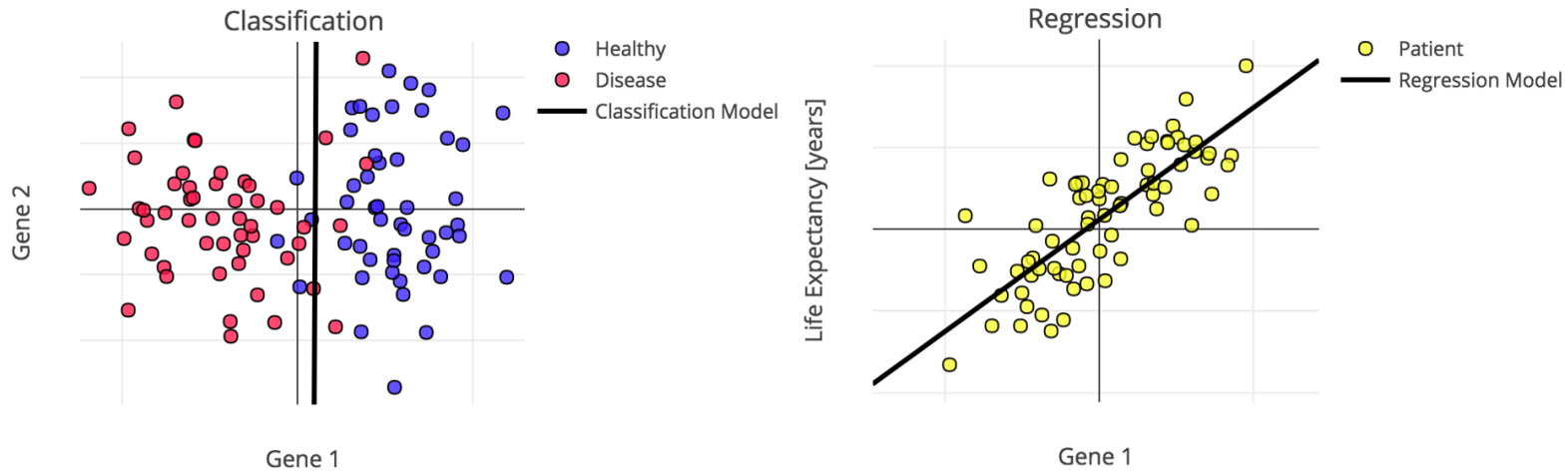
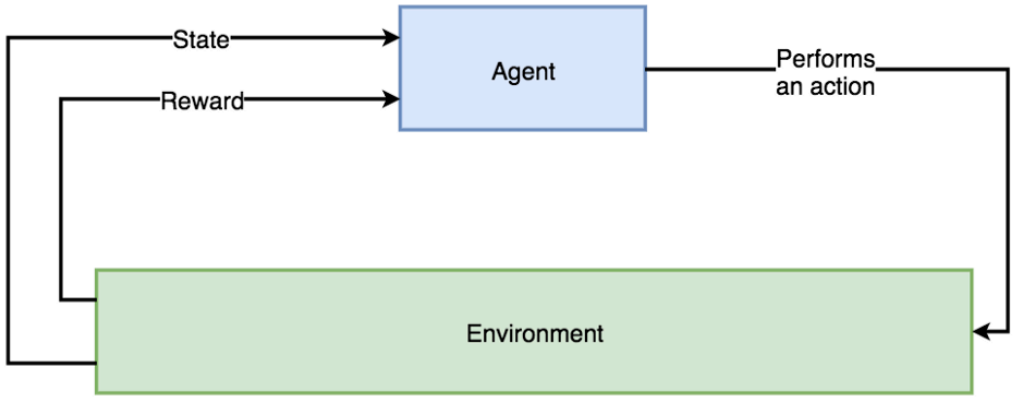


Image source: https://towardsdatascience.com/coding-deep-learning-for-beginners-types-of-machine-learning-b9e651e1ed9d

Machine Learning Taxonomy

- Semisupervised (reinforcement) learning
  - Develop models using partially labeled data (mix of labeled and unlabeled data)



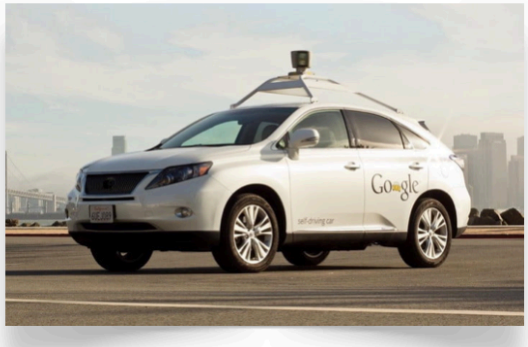
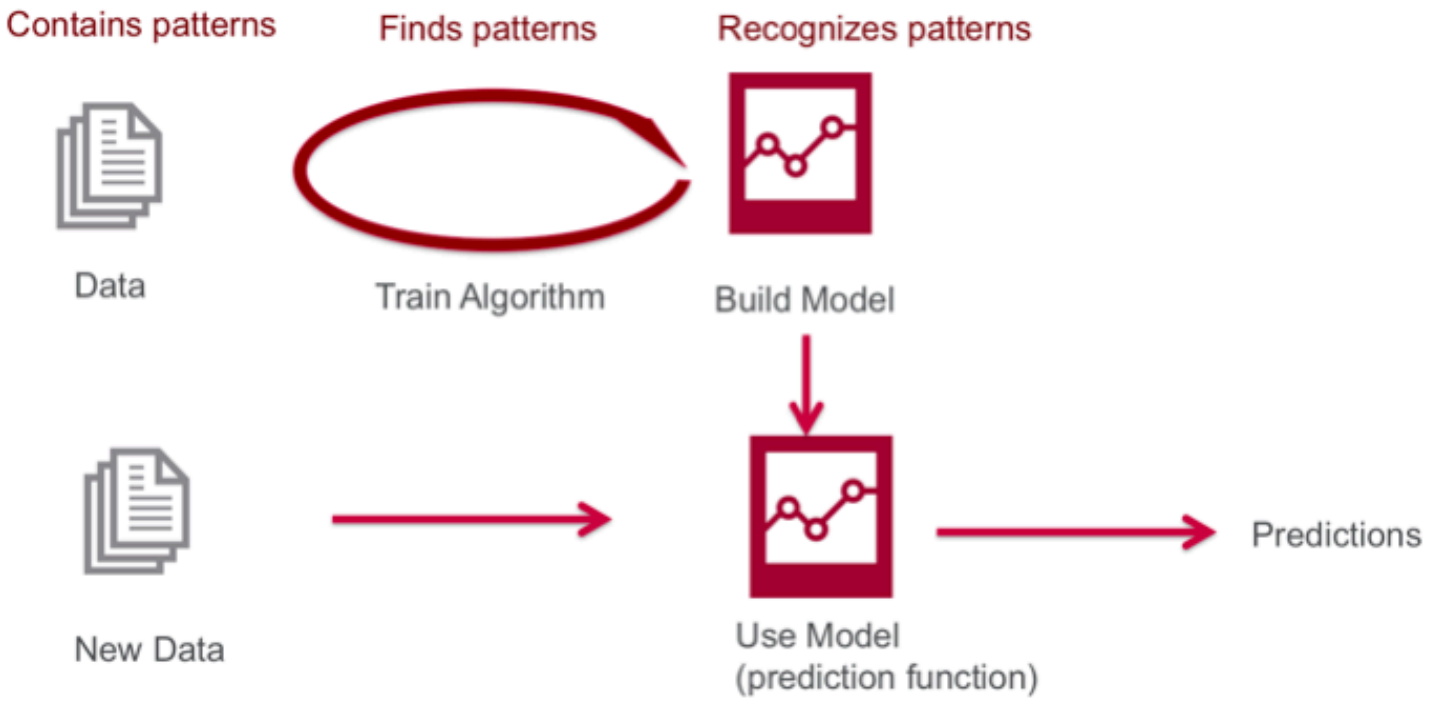


Image source: https://towardsdatascience.com/coding-deep-learning-for-beginners-types-of-machine-learning-b9e651e1ed9d

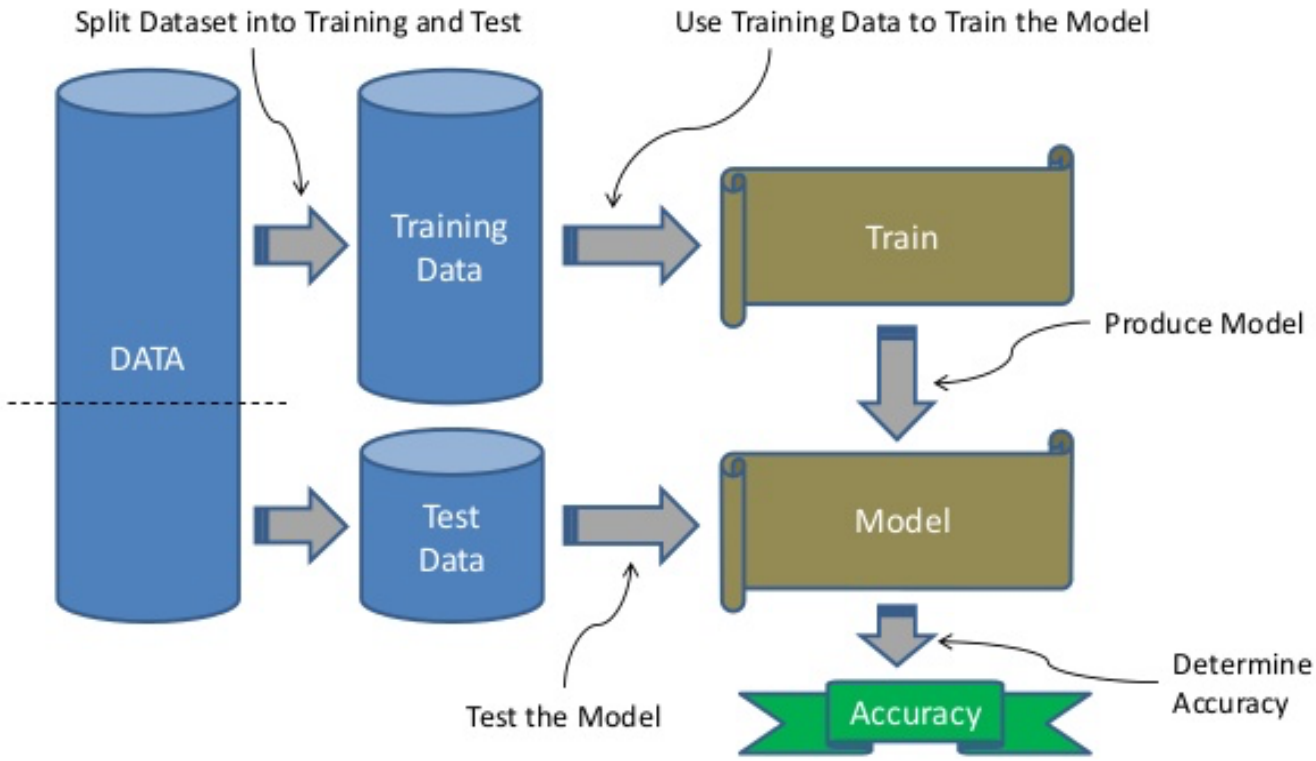
Common ML Workflow

- In a narrow scope, ML refers to…



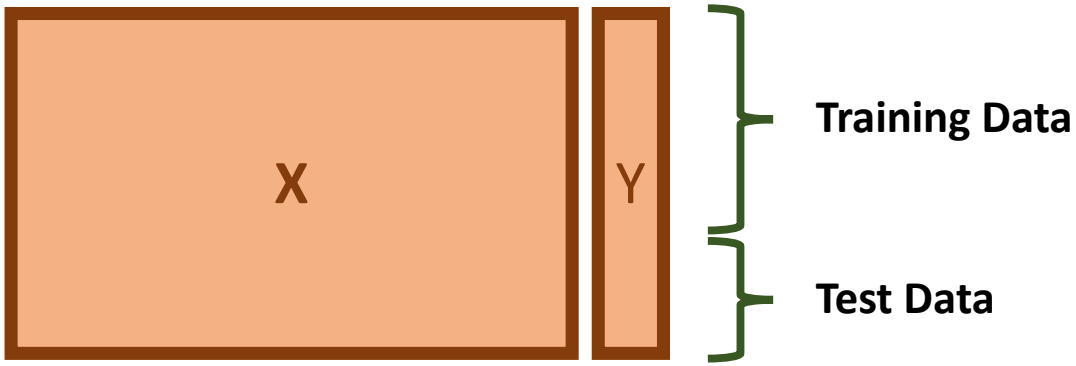
Training and Test Data

- Training data: a set of observations used to generate data models
- Test data: a set of observations used to evaluate trained data models



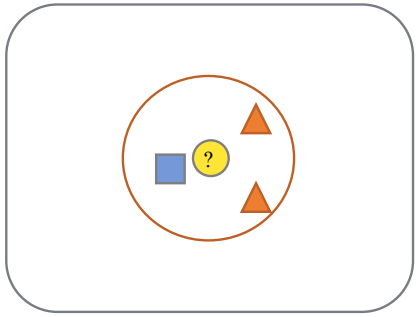
Data for Supervised Learning

- Data comes as a matrix
  - A row corresponds to an observation or a data instance
  - $\mathbf{X} = (X_1, \ldots, X_d)$: input attributes or features
  - Y: output, class, or label



k-Nearest Neighbors (kNN)

- A Simplest form of making predictions
  - *Here we are focusing on classification*; but the same idea could be applied to regression problems
- Instance-based approach
  - To make a decision, retrieve *k* closest training examples in the feature space
  - *A.k.a*, lazy learning approach; all computations are deferred until the prediction time
- Illustration (k = 3)



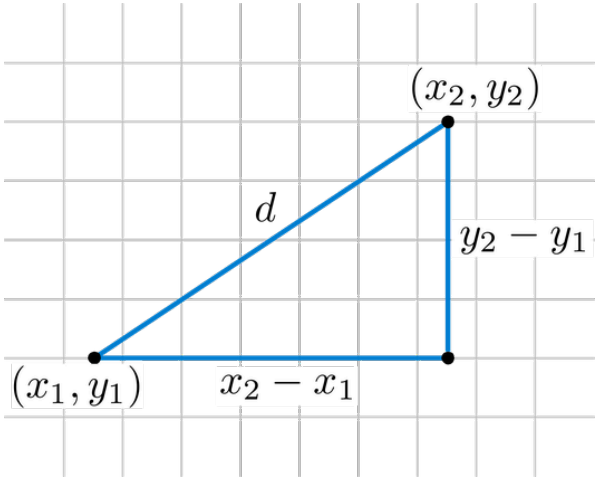
k-Nearest Neighbors (kNN)

- Requires 4 things:
  - Distance metric is determined according to the data type; *E.g.*, Euclidean, Manhattan, Hamming, …
  - *k* is determined empirically
  - To make a prediction, use a majority vote
  - Instance weighting could be used when making predictions

- To classify a test (unseen) instance:
  - Compute the distances to other training data instances
  - Identify *k* nearest neighbors (*k* closest data points to the test instance)
  - Determine the class label of the testing instance using the class labels of nearest neighbors (majority voting)

Data Normalization

- Consider Euclidean distance



$$E(x, y) = \sqrt{\sum_{i=0}^{n} (x_i - y_i)^2}$$

  - Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes
  - Example
    - *Height of a person may vary from 140 cm to 200 cm*
    - *Weight of a person may vary from 40 kg to 140 kg*
    - *Income of a person may vary from KRW 20M to 500M*

Agenda

- Python interesting facts
- Decision Trees
- Random Forest - Ensemble of Decision Trees
- (Visualizing Decision Trees)

Python Facts (1)

- Python does not support pointers
  - In Python, everything is done by reference (like Java)

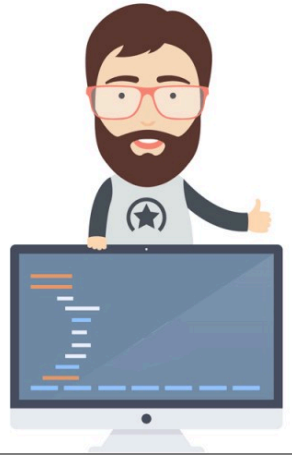


Image source: https://yourstory.com/mystory/interesting-facts-about-python-language

Python Facts (2)

- One function can return multiple values

```
1  # Multiple Return Values in Python!
2  def func():
3     return 1, 2, 3, 4, 5
4
5  one, two, three, four, five = func()
6  print(one, two, three, four, five)
7
8  a, _, _, b, c = func()
9  print(a, b, c)
10
```

```
1 2 3 4 5
1 4 5
```

Python Facts (3)

- One can use an "else" clause with a "for" loop
  - A special type of syntax
  - Executes only if the "for" loop exits naturally, without any break statements

```
1  def func(array):
2    for num in array:
3      if num%2==0:
4        print(num)
5        break # Case1: Break is called, so 'else' wouldn't be executed.
6    else: # Case 2: 'else' executed since break is not called
7      print("No call for Break. Else is executed")
8
9  print("1st Case:")
10 a = [2]
11 func(a)
12 print("2nd Case:")
13 a = [1]
14 func(a)
15
```

```
1st Case:
2
2nd Case:
No call for Break. Else is executed
```

Python Facts (4)

- *Function Argument Unpacking* is another awesome feature
  - One can unpack a list or a dictionary as function arguments using * and ** respectively
  - The arguments are passed in the same order in which they are specified in the list or dictionary
  - This is commonly known as the *Splat* operator

```
1 def point(x, y):
2   print(x,y)
3
4 foo_list = (3, 4)
5 bar_dict = {'y': 3, 'x': 2}
6
7 point(*foo_list) # Unpacking Lists
8 point(**bar_dict) # Unpacking Dictionaries
9
```

```
3 4
2 3
```

IFU

Python Facts (5)

- One can easily find the index (iteration number) inside a "for" loop
  - Wrap an iterable with ‘enumerate’ and it will yield the item along with its index

```
1 # Know the index faster
2 vowels=['a','e','i','o','u']
3 for i, letter in enumerate(vowels):
4   print (i, letter)
5
```

```
0 a
1 e
2 i
3 o
4 u
```

Python Facts (6)

- One can chain comparison operators
  - A condition 1<x<10 is executable

```
1  # Chaining Comparison Operators
2  i = 5;
3
4  ans = 1 < i < 10
5  print(ans)
6
7  ans = 10 > i <= 9
8  print(ans)
9
10 ans = 5 == i
11 print(ans)
12
```

```
True
True
True
```

Python Facts (7)

- Python can define Infinities

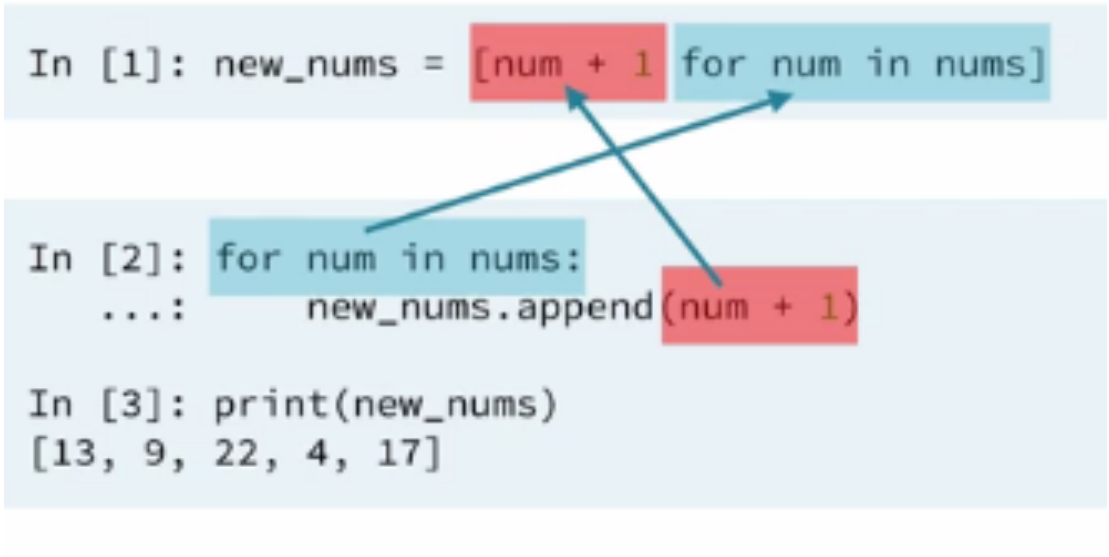
```
1  # Positive Infinity
2  p_infinity = float('Inf')
3
4  if 99999999999999 > p_infinity:
5    print("The number is greater than Infinity!")
6  else:
7    print("Infinity is greatest")
8
9  # Negetive Infinity
10 n_infinity = float('-Inf')
11 if -99999999999999 < n_infinity:
12   print("The number is lesser than Negative Infinity!")
13 else:
14   print("Negative Infinity is least")
15
```

```
Infinity is greatest
Negative Infinity is least
```

IGU

Python Facts (8)

- Instead of building a list with a loop, one can build it more concisely with a *list comprehension*



Python Facts (8)

- Instead of building a list with a loop, one can build it more concisely with a *list comprehension*

```
1 # Simple List Append
2 a = []
3 for x in range(0,10):
4   a.append(x)
5 print(a)
6
7 # List Comprehension
8 print([x for x in a])
9
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

IFU

Python Facts (8)

- Instead of building a list with a loop, one can build it more concisely with a *list comprehension*

```
mylist = []
for i in range(0,5):
    mylist.append(i)
mylist
[0, 1, 2, 3, 4]

mylist = [i for i in range(0,5)]
mylist
[0, 1, 2, 3, 4]
```

```
squarelist = []
for i in range(0,5):
    if i%2==0:
        squarelist.append(i*i)
squarelist
[0, 4, 16]

squarelist = [i*i for i in range(0,5)if i%2==0]
squarelist
[0, 4, 16]
```

Python Facts (9)

- Slice Operator is a way to get items from lists, as well as change them

```
1  # Slice Operator
2  a = [1,2,3,4,5]
3
4  print(a[0:2]) # Choose elements [0-2), upper-bound noninclusive
5
6  print(a[0:-1]) # Choose all but the last
7
8  print(a[::-1]) # Reverse the list
9
10 print(a[::2]) # Skip by 2
11
12 print(a[::-2]) # Skip by -2 from the back
13
```

```
[1, 2]
[1, 2, 3, 4]
[5, 4, 3, 2, 1]
[1, 3, 5]
[5, 3, 1]
```

IFU

Python Facts (10)

- There is a poem written by Tim Peters named as THE ZEN OF PYTHON which can be read by just writing `import this`

```
1 # Try to guess the result before you actually run it
2 import this
```

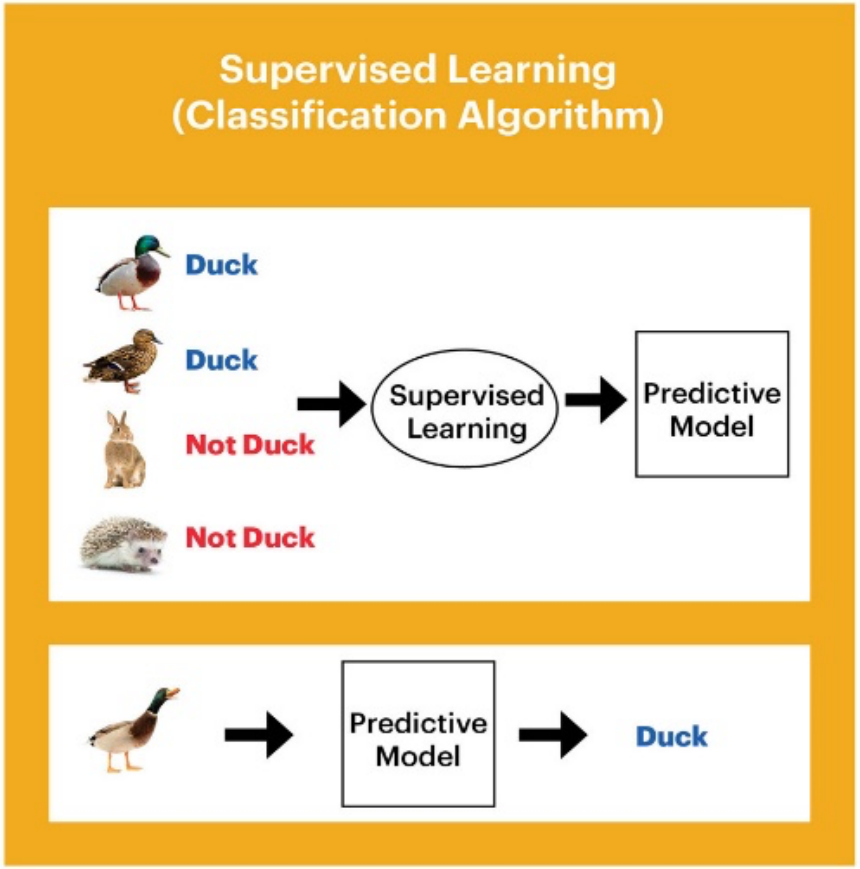
```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
```

Agenda

- Python interesting facts
- **Decision Trees**
- Random Forest - Ensemble of Decision Trees
- (Visualizing Decision Trees)

Background

- Supervised learning: "*I already have data and answers*"



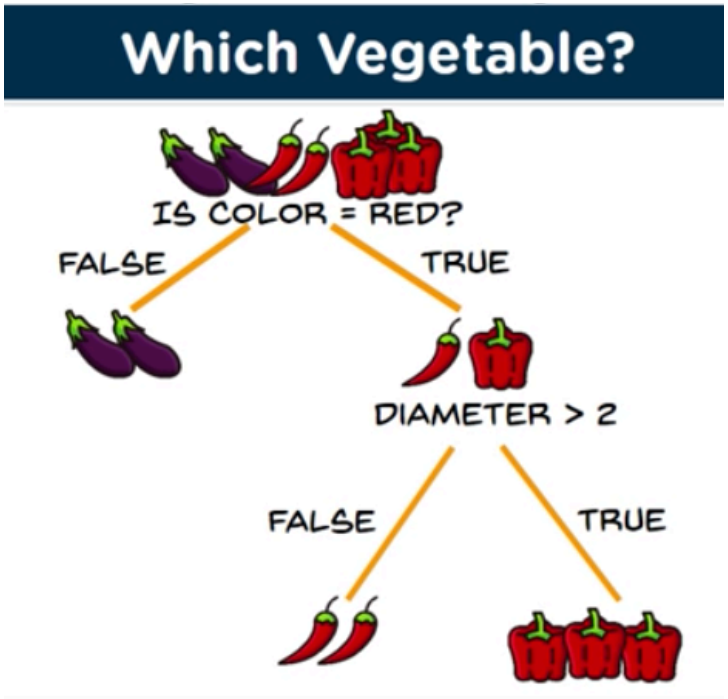
Background

- Supervised learning: "*I already have data and answers*"
  - Classification examples

Data	***Answers (labels)***
Measures from test stations	Good or scrap part
Loan application	Paid the debt in time or not
Credit card transaction records	Normal or fraud transaction

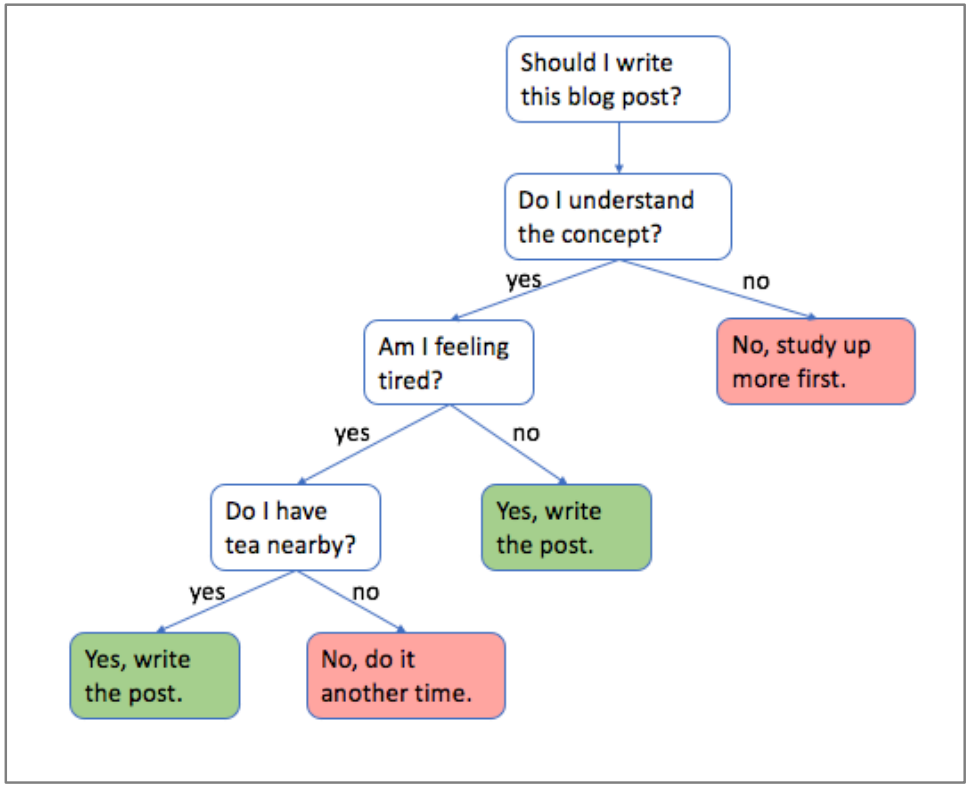
Decision Trees (DT)

- A tree-shaped model that represents courses of actions
  - Each branch of the tree represents a possible decision or action
    - *E.g.*, How to determine the type of vegetables in my shopping bag?



Decision Trees (DT)

- An effective tools to represent decision rules
  - *E.g., Do I want to write a blog post about an issue?*
    Am I eligible for military service?



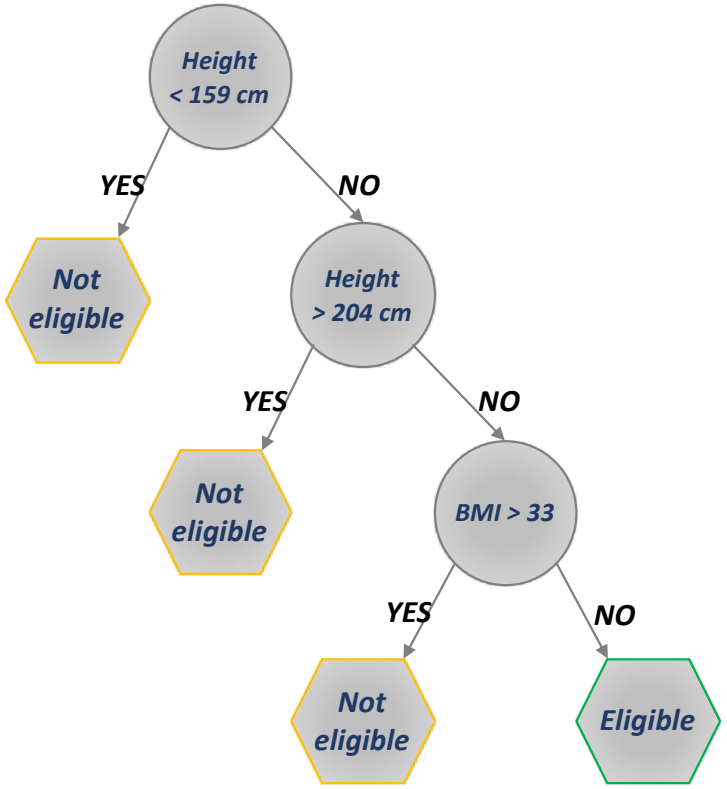
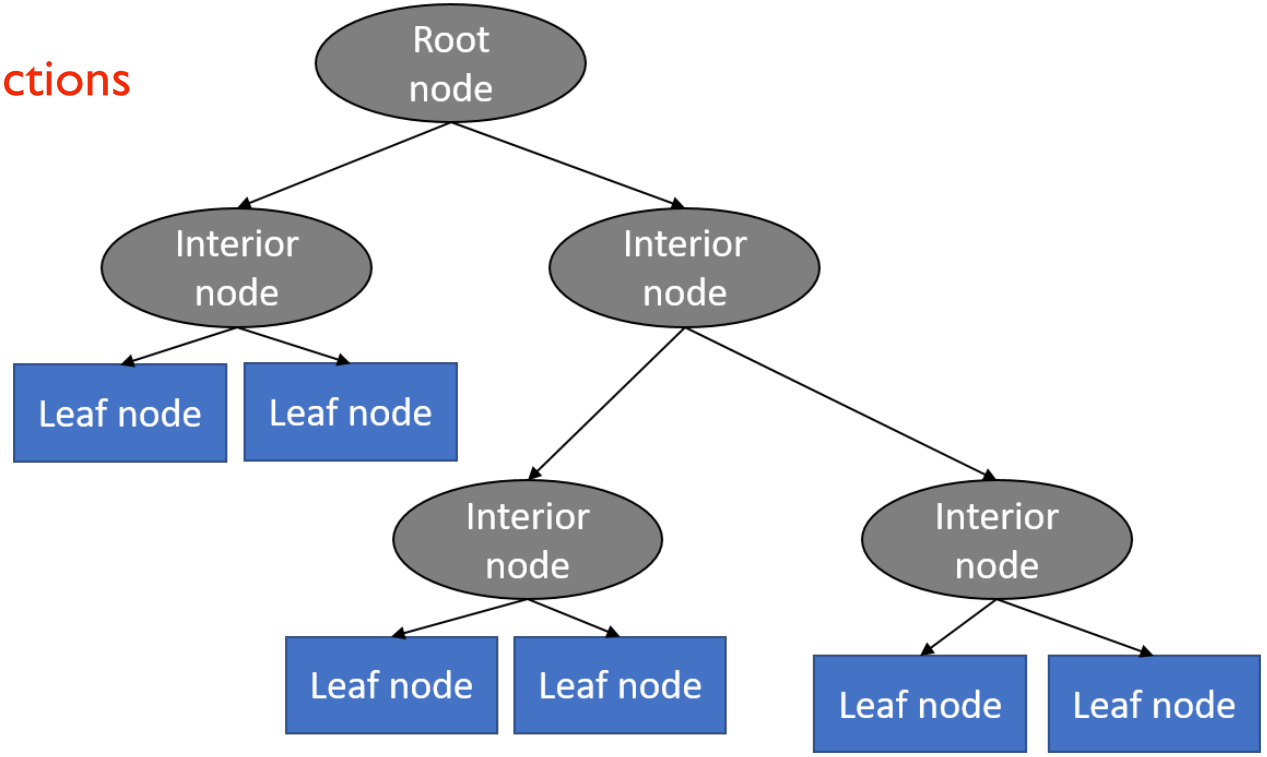


Image source: https://www.jeremyjordan.me/decision-trees/

Terminologies

- Nodes
  - Root node: initial query
  - Interior (internal) nodes: follow-up queries
  - Leaf nodes: Node without children
    - Decision nodes
    - Carry out predictions



Decision Trees (DT)

- Key Dev Questions
  - [Prediction] *Given a decision tree, how to make predictions?*
  - [Training] *How to train (learn) a decision tree from data?*

Prediction

- A decision tree $f: \mathbf{X} \rightarrow Y$
  - Assuming that a trained decision tree is given:
    - For each data instance, answer to the query on each node and take the branch with answer (move on to one of its child nodes)
    - When the instance arrives at a leaf node, make a prediction according to the node's decision
  - This could be efficiently implemented as a recursive program

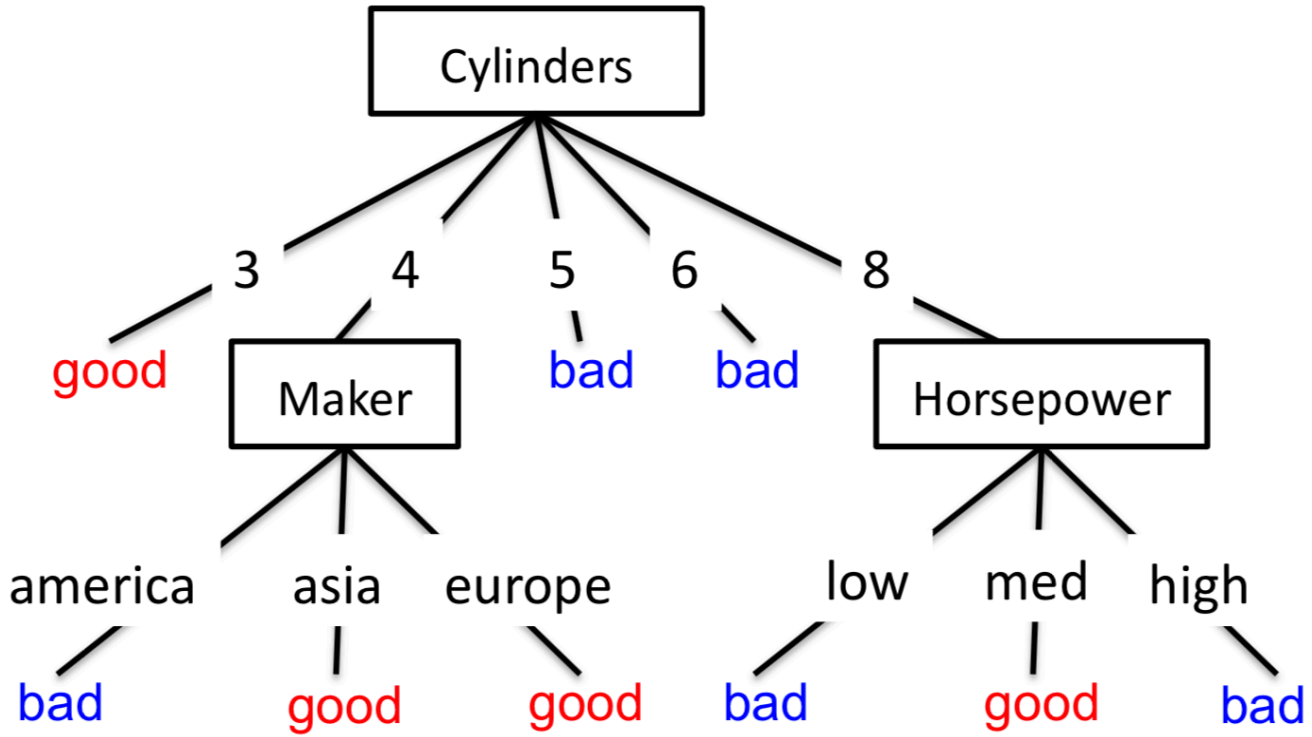
Prediction

- Example: Car evaluation dataset
  - https://archive.ics.uci.edu/ml/datasets/car+evaluation
  - Task: evaluate if the mpg (miles per gallon) of a car is good/bad

mpg	cylinders	displacement	horsepower	weight	acceleration	modelyear	maker
good	4	low	low	low	high	75to78	asia
bad	6	medium	medium	medium	medium	70to74	america
bad	4	medium	medium	medium	low	75to78	europe
bad	8	high	high	high	low	70to74	america
bad	6	medium	medium	medium	medium	70to74	america
bad	4	low	medium	low	medium	70to74	asia
bad	4	low	medium	low	low	70to74	asia
bad	8	high	high	high	low	75to78	america
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
bad	8	high	high	high	low	70to74	america
good	8	high	medium	high	high	79to83	america

Prediction

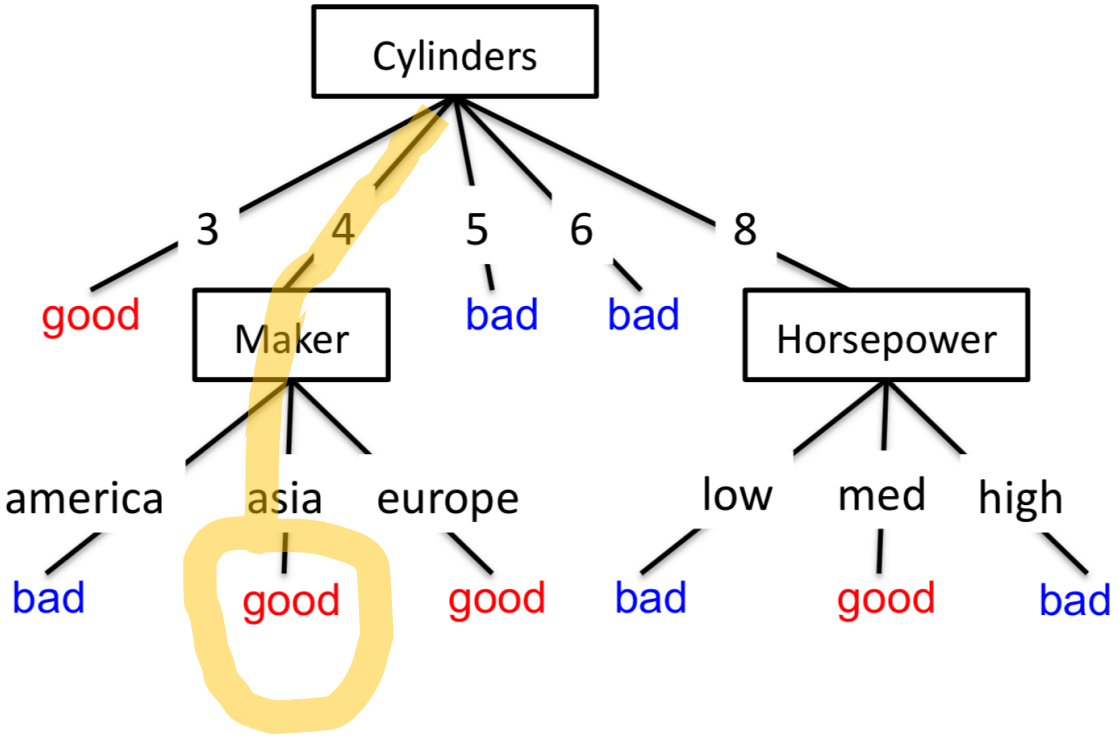
- Example: Car evaluation dataset
  - A trained decision tree



Prediction

- Example: Car evaluation dataset
  - A trained decision tree

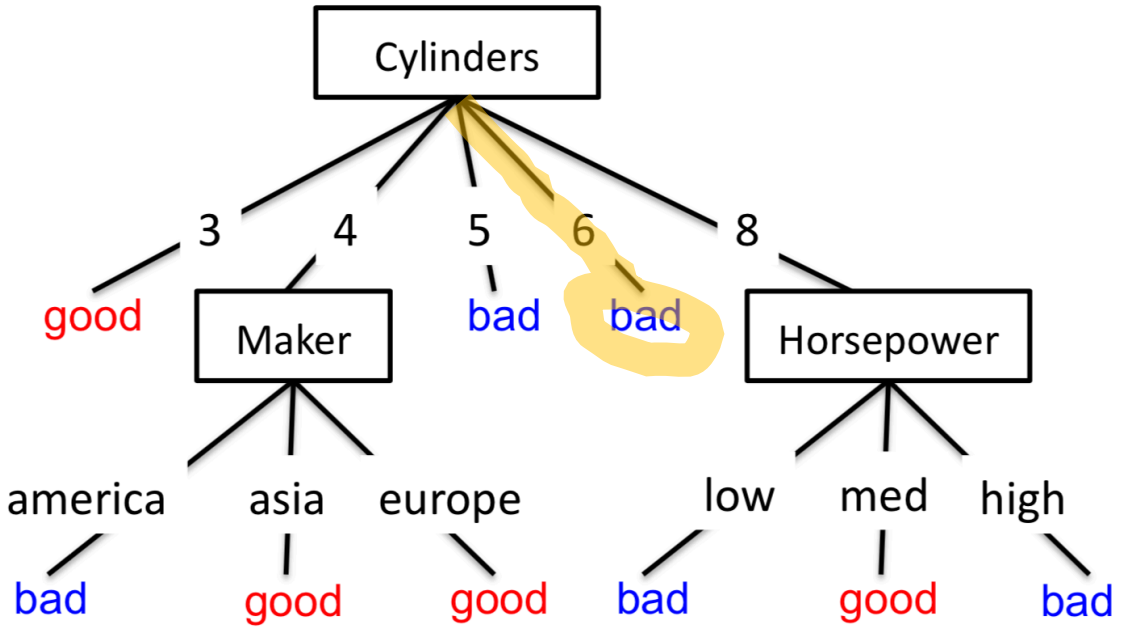
cylinders	displacement	horsepower	weight	acceleration	modelyear	maker
4	low	low	low	high	75to78	asia



Prediction

- Example: Car evaluation dataset
  - A trained decision tree

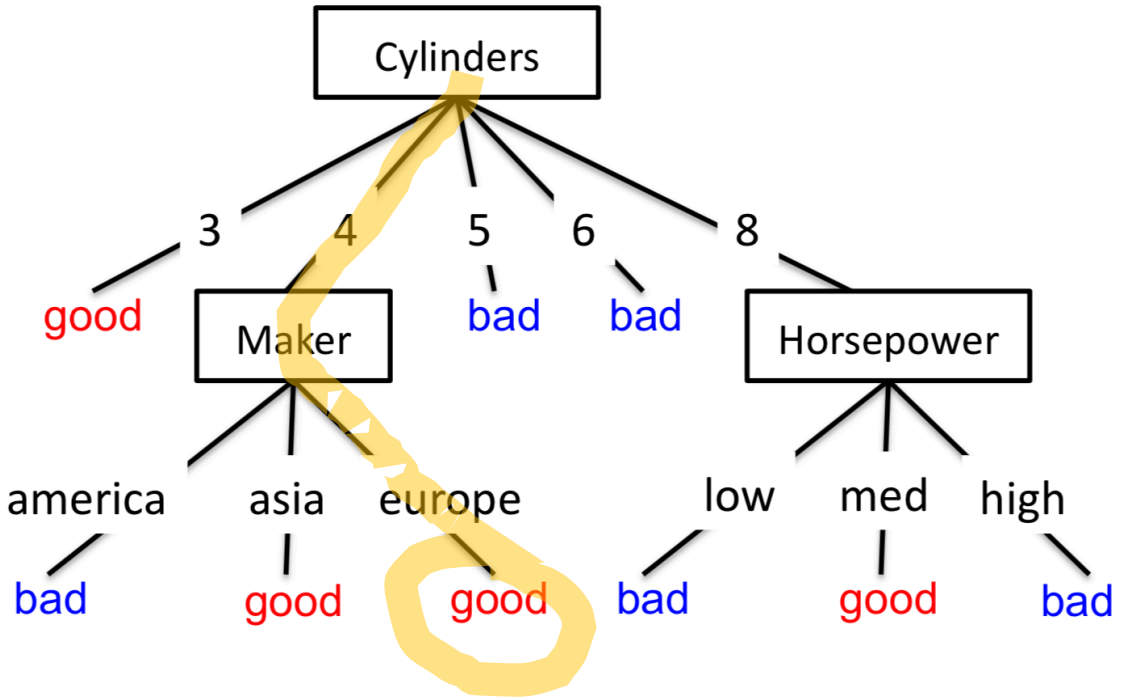
cylinders	displacement	horsepower	weight	acceleration	modelyear	maker
6	medium	medium	medium	medium	70to74	america



Prediction

- Example: Car evaluation dataset
  - A trained decision tree

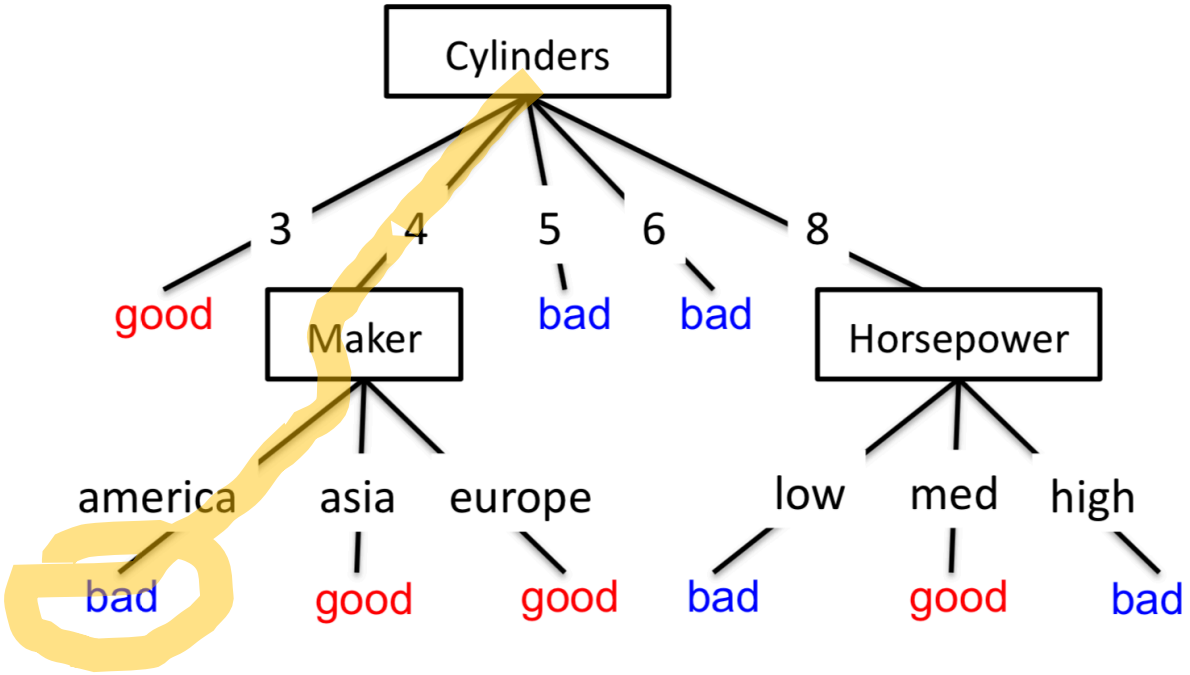
cylinders	displacement	horsepower	weight	acceleration	modelyear	maker
4	medium	medium	medium	low	75to78	europe



Prediction

- Example: Car evaluation dataset
  - A trained decision tree

cylinders	displacement	horsepower	weight	acceleration	modelyear	maker
4	low	low	low	low	79to83	america



Training

- Idea
  - Start from an empty decision tree, with all train data
  - Split on next best attribute (feature)
    - Use, for example, information gain to decide the splitting condition

- This could be efficiently implemented as a recursive program

- Question: How to make splits?
  How to split the data into multiple nodes?

Training

- Background: Entropy (degree of complexity in data)
  - High entropy
    - Y is from a uniform-like distribution
    - Near-flat histogram
    - Less predictable
  - Low entropy
    - Y is from a varied (peaks and valleys) distribution
    - Histogram has many lows and highs
    - More predictable

Training

- Splitting criterion: Information gain
  - Decrease in entropy (uncertainty) after splitting a node
    - IG(X) = H(Y) - H(Y|X)

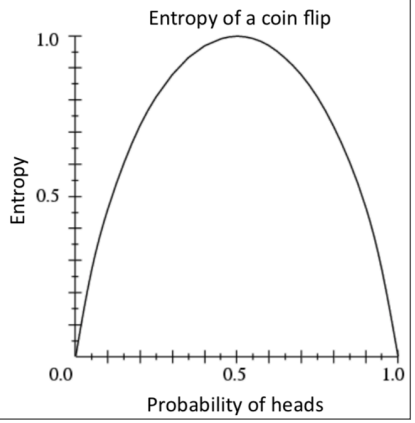
      = (Entropy before split) - (Entropy after split)

      - H(Y): Entropy of a variable Y
      - H(Y|X): Conditional entropy of Y given X

$$H(Y) = -\sum_{i=1}^{k} P(Y = y_i) \log_2 P(Y = y_i)$$

More uncertainty, more entropy!

Information Theory interpretation: *H(Y)* is the expected number of bits needed to encode a randomly drawn value of Y (under most efficient code)



Training

- Glossary
  - H(Y): Entropy of a variable Y

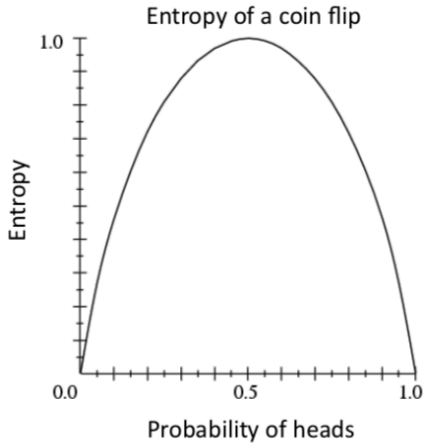
$$H(Y) = -\sum_{i=1}^{k} P(Y = y_i) \log_2 P(Y = y_i)$$

  - H(Y|X): Conditional entropy of Y given X

$$H(Y \mid X) = -\sum_{j=1}^{v} P(X = x_j) \sum_{i=1}^{k} P(Y = y_i \mid X = x_j) \log_2 P(Y = y_i \mid X = x_j)$$

Training

Entropy Example



$$H(Y) = -\sum_{i=1}^{k} P(Y = y_i) \log_2 P(Y = y_i)$$

P(Y=t) = 5/6

P(Y=f) = 1/6

H(Y) = - 5/6 $\log_2$ 5/6 - 1/6 $\log_2$ 1/6

= 0.65

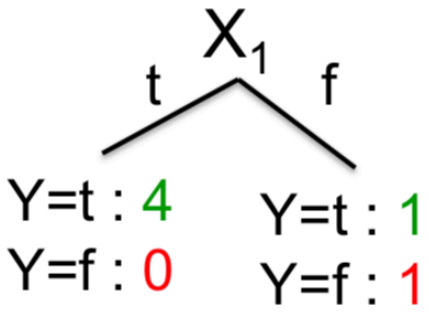
X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F

Conditional Entropy Example

Example:

$P(X_1=t) = 4/6$

$P(X_1=f) = 2/6$



X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F

$$H(Y|X_1) = - 4/6\ (1 \log_2 1 + 0 \log_2 0)$$
$$- 2/6\ (1/2 \log_2 1/2 + 1/2 \log_2 1/2)$$
$$= 2/6$$

Information Gain Example

In our running example:

$$IG(X_1) = H(Y) - H(Y|X_1)$$
$$= 0.65 - 0.33$$

$IG(X_1) > 0$ → we prefer the split!

X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F

Decision Trees (DT)

- Pros
  - Simple to understand, interpret, and visualize
  - Little effort is required for data preparation
    - Normalization is not required
  - Non-linear decision boundary could be learned
  - Can handle both numerical and categorical data

- Cons
  - Overfitting
  - High variance (model tends to be less stable)

Agenda

- Python interesting facts
- Decision Trees
- **Random Forest - Ensemble of Decision Trees**
- (Visualizing Decision Trees)

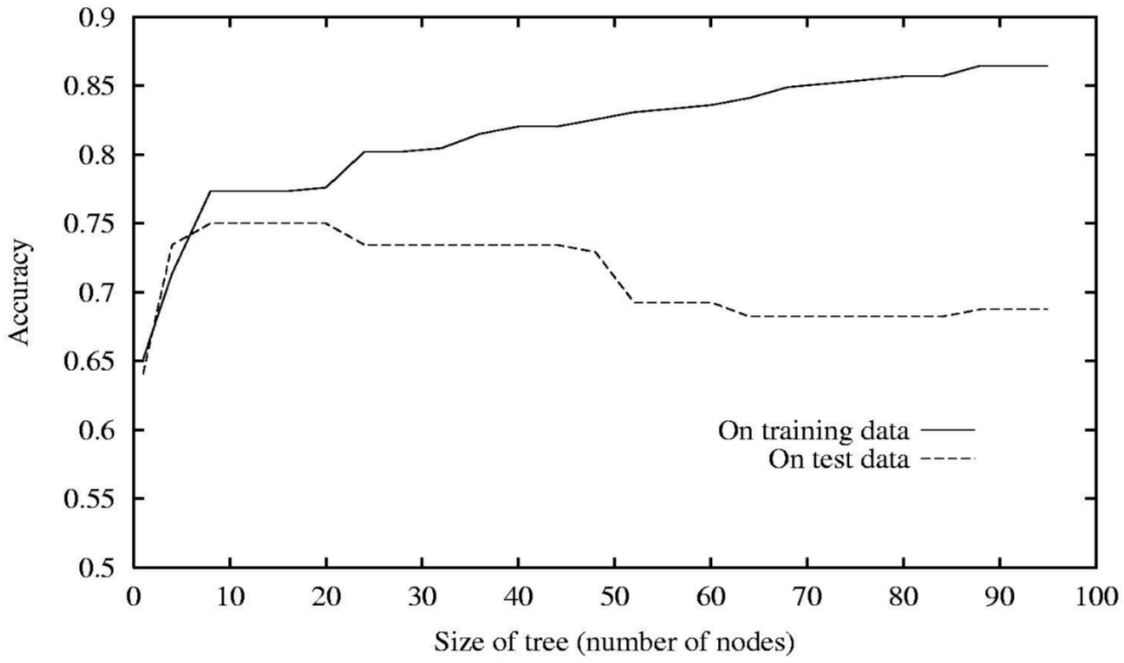
Background

- Issues with Decision Trees
  - Overfitting: occurs when the algorithm captures noise in the data
  - Unstable: the model can get unstable with merely small variations in data (the model could get too much sensitive)

- Solutions
  - Must use tricks to find "simple trees"
    - Early stopping of training (learning)
      - Fixed depth: do not grow trees further than a specified depth
      - Minimum population per leaf node: do not split a node if the number of data instances fall in the node is smaller than a specified number
    - Readjusting/simplifying trained trees
      - Pruning: simplify trees by merging leaf nodes

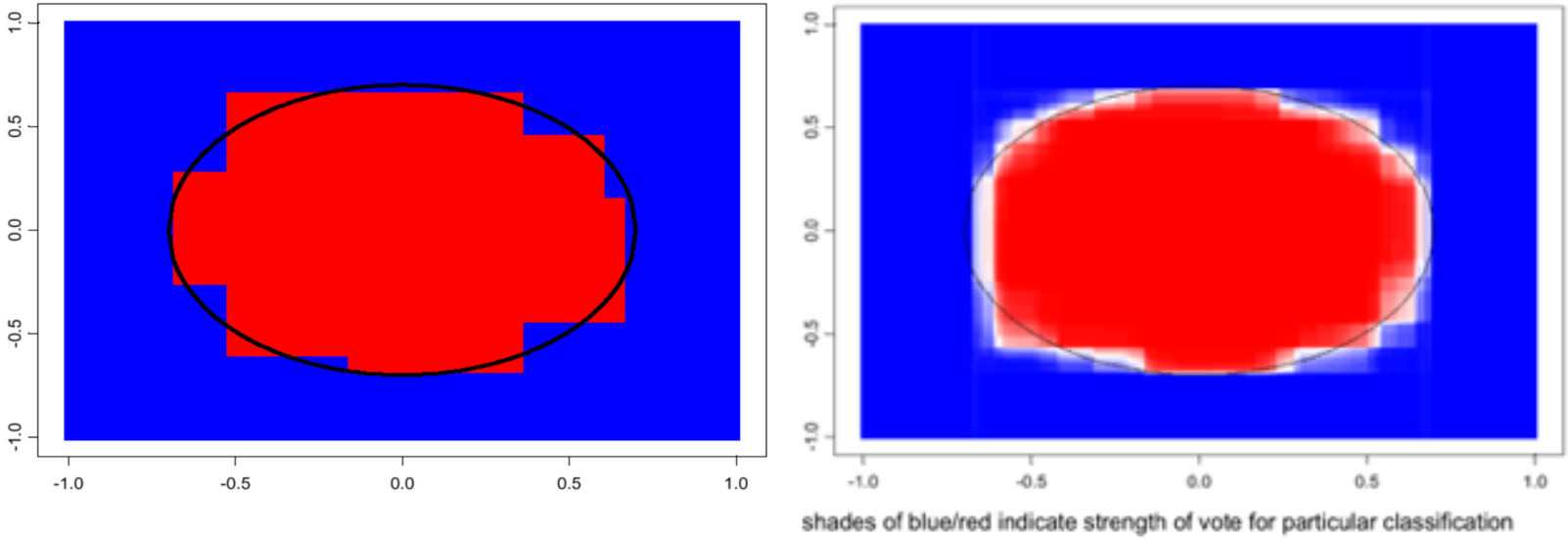
Background

- Issues with Decision Trees
  - Overfitting: occurs when the algorithm captures noise in the data
  - Unstable: the model can get unstable with merely small variations in data (the model could get too much sensitive)



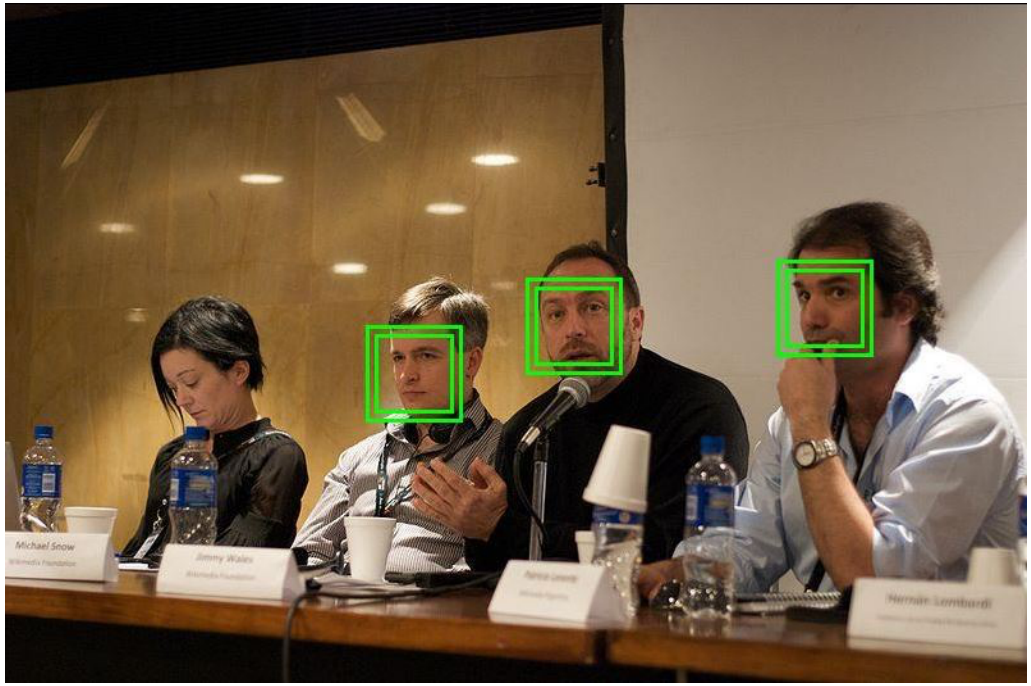
Random Forests (RF)

- Ensemble (mixture) of decision tress
  - Use multiple decision trees together and improve the predictive performance
  - Illustration: using 1 tree vs. 100 trees



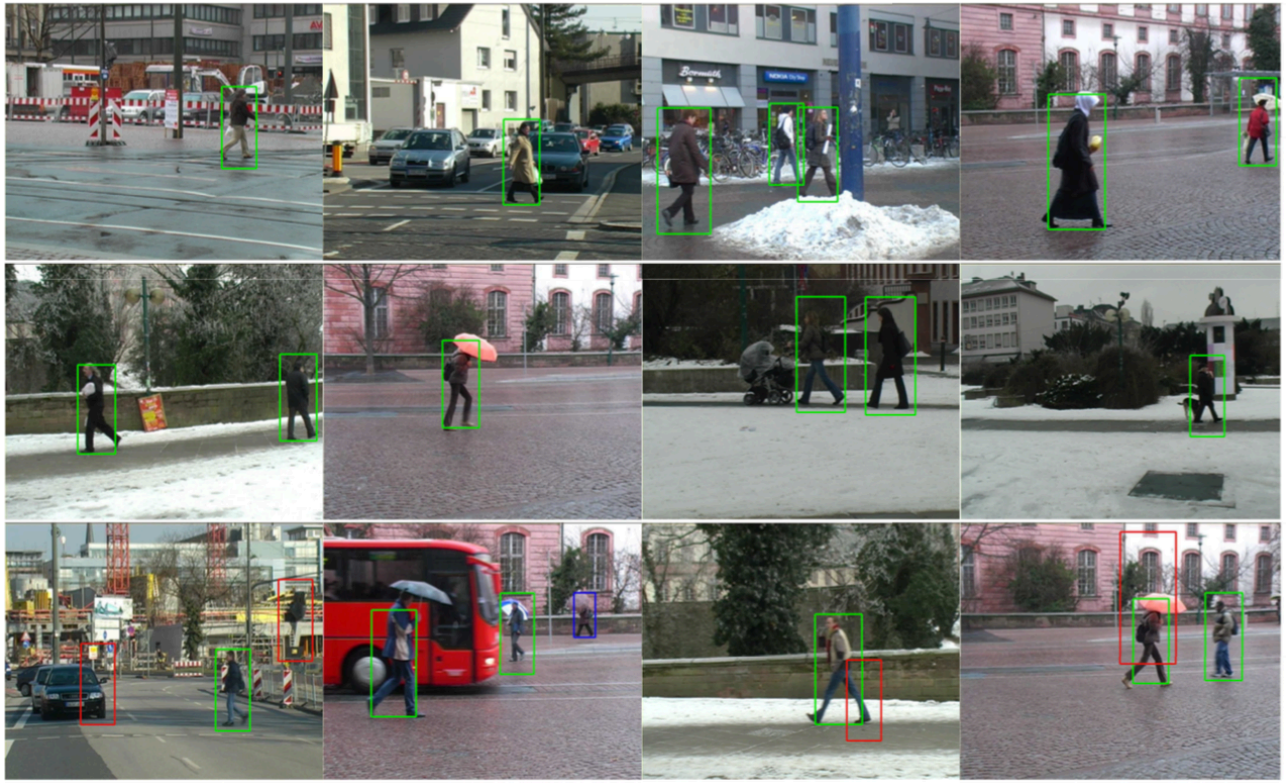
RF in Applications

- Face recognition



RF in Applications

- Pedestrian detection



- Microsoft Kinect

Agenda

- Python interesting facts
- Decision Trees
- Random Forest - Ensemble of Decision Trees
- **(Visualizing Decision Trees)**

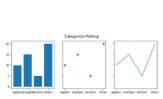
Data Visualization with Python

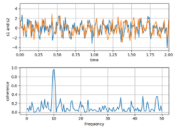
- Matplotlib
  - 2D plotting library which produces publication quality figures
  - One can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, *etc*.

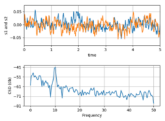
Image source: https://matplotlib.org/

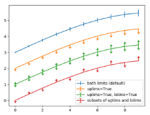
Data Visualization with Python

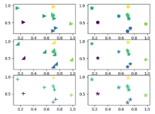
- Gallery (with code): https://matplotlib.org/3.1.0/gallery/index.html

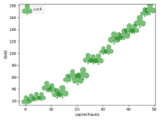

Plotting categorical variables

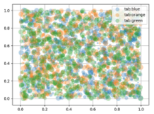

Plotting the coherence of two signals

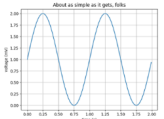

CSD Demo

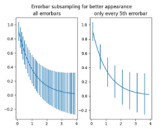

Errorbar limit selection

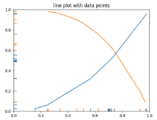

Scatter Star Poly

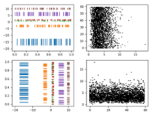

Scatter Symbol

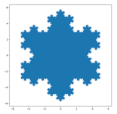

Scatter plots with a legend

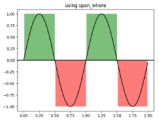

Simple Plot

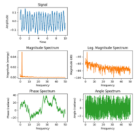

Errorbar Subsample

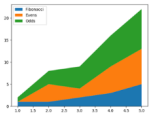

EventCollection Demo

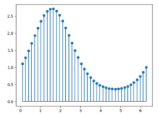

Eventplot Demo

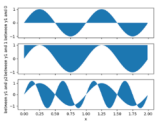

Filled polygon

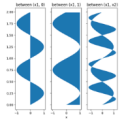

Using span_where

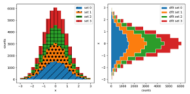

Spectrum Representations

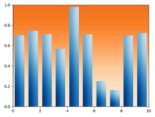

Stackplot Demo

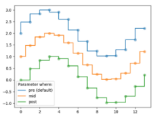

Stem Plot

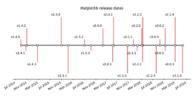

Filling the area between lines

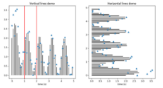

Fill Betweenx Demo

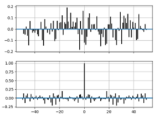

Hatch-filled histograms


Bar chart with gradients


Step Demo


Creating a timeline with lines, dates, and text


hlines and vlines


Cross- and Auto-Correlation Demo

IfU

Data Visualization with Python

- Gallery (with code): https://matplotlib.org/3.1.0/gallery/index.html

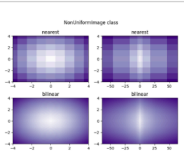
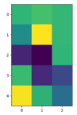
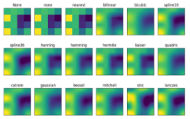
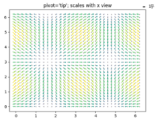


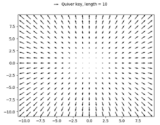
Image Nonuniform

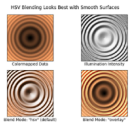

Blend transparency with color in 2-D images

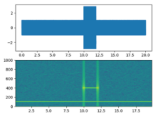

Modifying the coordinate formatter

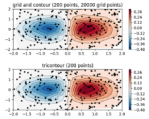

Interpolations for imshow/matshow

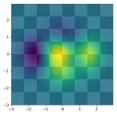

Advanced quiver and quiverkey functions

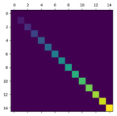

Quiver Simple Demo

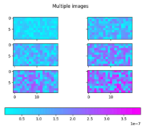

Shading example

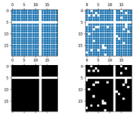

Spectrogram Demo

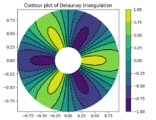

Contour plot of irregularly spaced data

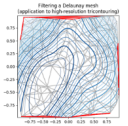

Layer Images

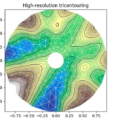

Matshow

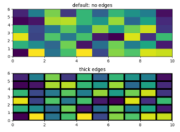

Multi Image

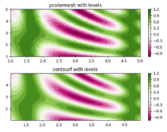

Spy Demos

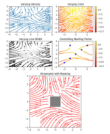

Tricontour Demo

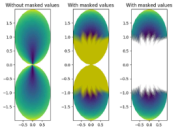

Tricontour Smooth Delaunay

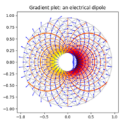

Tricontour Smooth User

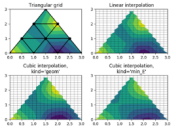

Pcolor Demo

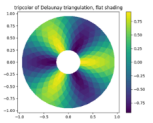

pcolormesh

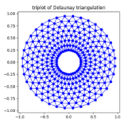

Streamplot


QuadMesh Demo


Trigradient Demo


Triinterp Demo


Tripcolor Demo


Triplot Demo

(Visualizing Decision Trees)

- Optional task for today
  - Visualize the trees that we trained from data

References

- Peter Harrington, Machine Learning in Action. Manning Publications, 2012.
- John Guttag (김영섭 역), Introduction to Computation and Programming Using Python with Application to Understanding Data. 2nd Ed. MIT Press, 2016.
- Henrik Brink et al., Real-world Machine Learning. Manning Publications, 2016.
- Tom Mitchell, Machine Learning. McGraw-Hill, 1997.

Thank you!

Machine Learning with Real World Data (Day 2)

홍참길, 한동대학교 전산전자공학부

July 17, 2019 @ 전산전자공학부 여름 학기 전공 캠프

charmgil@handong.edu

한동대학교 HANDONG GLOBAL UNIVERSITY