

PYTHON FOR MACHINE LEARNING

Kumar Sambhav
HPC Specialist

Agenda

- Know about Python
- Python Programming basics
- Understanding Python modules and packages
- Machine Learning

Python, what is the hype about?

- Python is a general-purpose programming language
- Coherent, readable, reusable and maintainable code
- Low barrier to start
- Considerably smaller amount of code to achieve the same task as compared to C, C++, Java etc.
- Portability of code helps us to run the same code on any platform with no change.
- Great opensource package support

Java

```
1 import java.util.ArrayList;
2
3 public class Main {
4     public static void main(String[] args) {
5         ArrayList al = new ArrayList();
6         al.add("a");
7         al.add("b");
8         al.add("c");
9         System.out.println(al)
10    }
11 }
```

Python

```
1 aList = []
2
3 aList.append("a")
4 aList.append("b")
5 aList.append("c")
6
7 print(aList)
```

“Hello, World”

- **C**

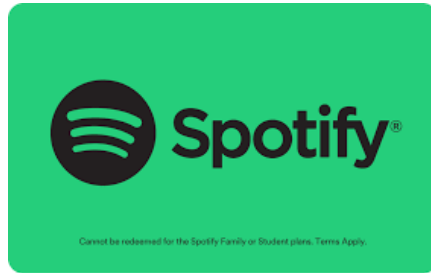
```
#include <stdio.h>

int main(int argc, char ** argv)
{
    printf("Hello, World!\n");
}
```
- **Java**

```
public class Hello
{
    public static void main(String argv[])
    {
        System.out.println("Hello, World!");
    }
}
```
- **now in Python**

```
print "Hello, World!"
```

Who uses Python?



And many many many many more

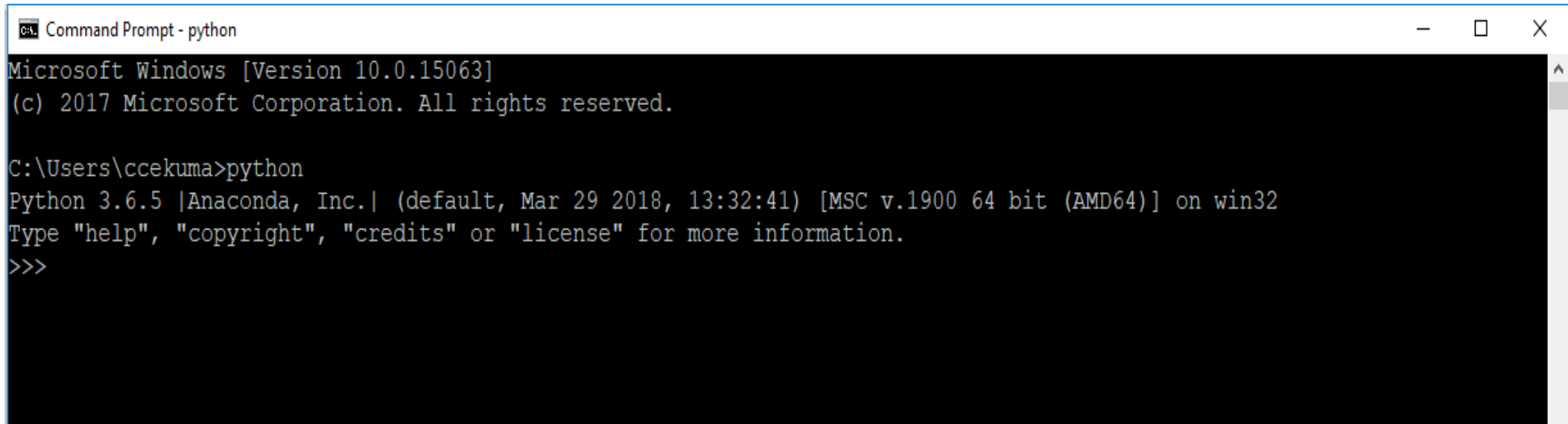
What can we do with Python ?

- Systems Programming
- GUI Development
- Component Integration with C/C++/Java based systems
- Database Programming
- Rapid Prototyping
- Numeric and Scientific Programming
- Machine Learning
- Data Visualization
- Data Mining

An much more !

Running Python

- Interactive Prompt or REPL (Read Eval Print Loop)
Useful for quickly running small pieces of code interactively

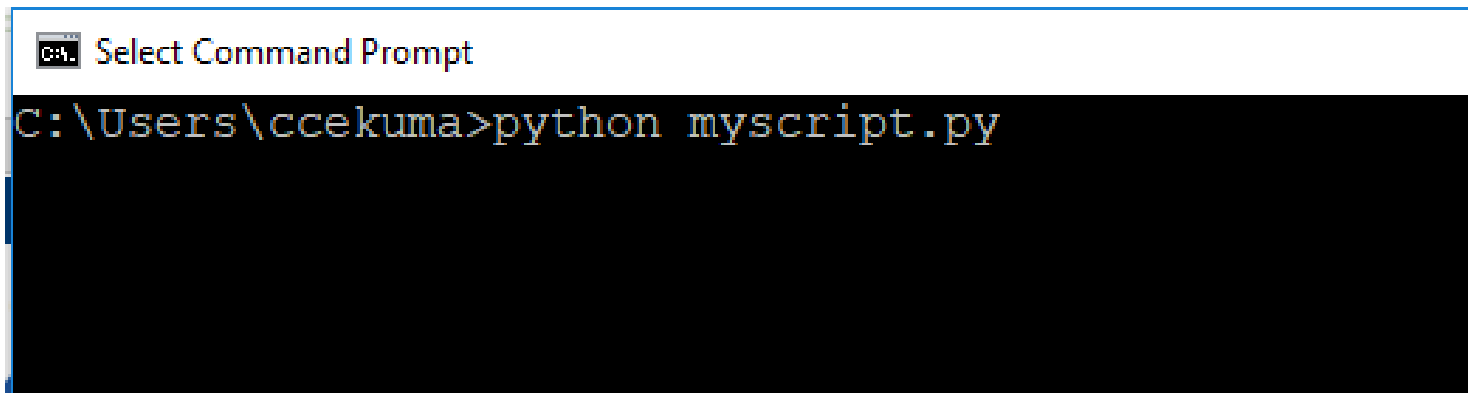


```
Command Prompt - python
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\ccekuma>python
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Running Python

- Python code can be written in files and saved with a .py extension
- The .py file can be then run as shown below :

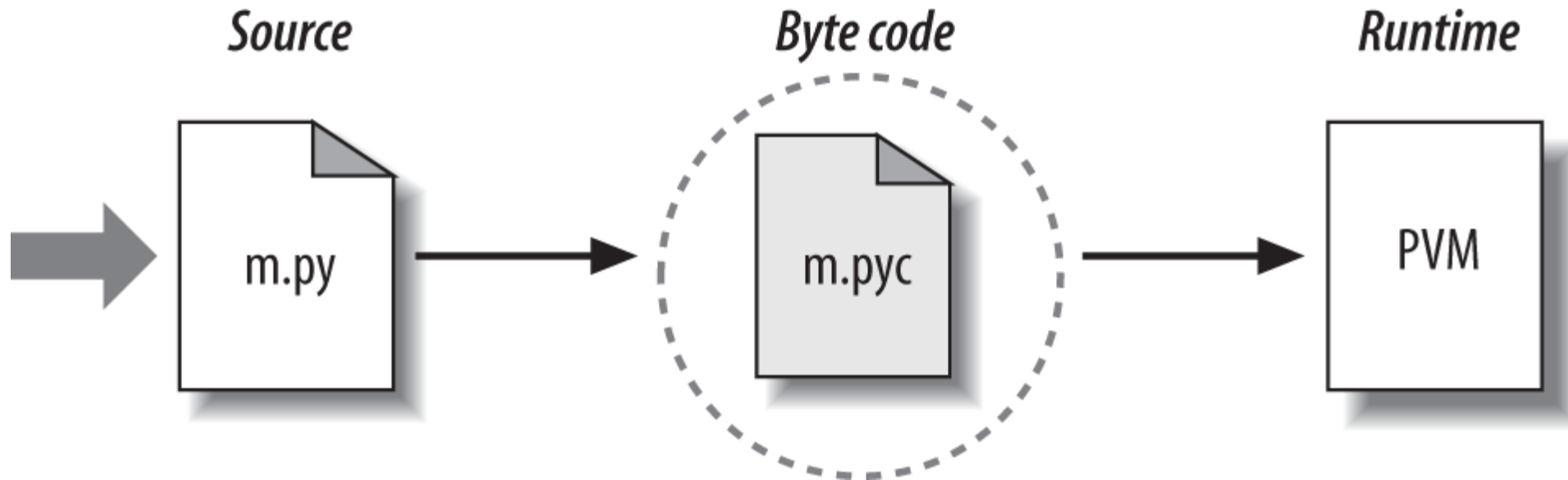


```

Select Command Prompt
C:\Users\ccekuma>python myscript.py

```

Understanding the execution model



Core Data Types

Object type	Example literals/creation
Numbers	<code>1234, 3.1415, 3+4j, 0b111, Decimal(), Fraction()</code>
Strings	<code>'spam', "Bob's", b'a\x01c', u'sp\xc4m'</code>
Lists	<code>[1, [2, 'three'], 4.5], list(range(10))</code>
Dictionaries	<code>{'food': 'spam', 'taste': 'yum'}, dict(hours=10)</code>
Tuples	<code>(1, 'spam', 4, 'U'), tuple('spam'), namedtuple</code>
Files	<code>open('eggs.txt'), open(r'C:\ham.bin', 'wb')</code>
Sets	<code>set('abc'), {'a', 'b', 'c'}</code>
Other core types	<code>Booleans, types, None</code>

Numbers

- Python provides implementations for:
 - Integers
 - Floating point numbers
 - Complex numbers
 - Decimals with precision
 - Sets
 - Rational numbers
 - As well as numeric operations like :
 - Addition (+)
 - Subtraction (-)
 - Exponentiation (**)
- And much more

Strings

- Strings are used to store textual information
- Strings are immutable
- One of the type of ***sequence*** others being Lists and tuples
- Strings being sequences, lets us access values by index
- The built in len function gives the length of any string
- Indexing on sequences can be forward or backward
- We can use slicing to take out a part of any string
- We can use addition operator (+) to concatenate two strings

Lists

Lists :

- Ordered Collection of arbitrary objects

```
my_list=[1, "Sam", ["list inside list"],1.25,True]
```

- Elements are accessed by index or offset

```
>>>my_list[3]
```

- Mutable Sequence and hence can be changed in place (See notes)

Dictionaries

Dictionaries:

- Collection of Key-value pairs

```
>>> my_dict={'name':"Alice", 'age':40}
```

- Accessed by Keys (and not indices)

```
>>> my_dict['name']  
"Alice"
```

- Mutable Mapping and hence can be changed in place (See notes)

Tuples

- Ordered collection of arbitrary objects

```
>>>my_tuple=(1, "Ben", "30",[85022133,84228117])
```

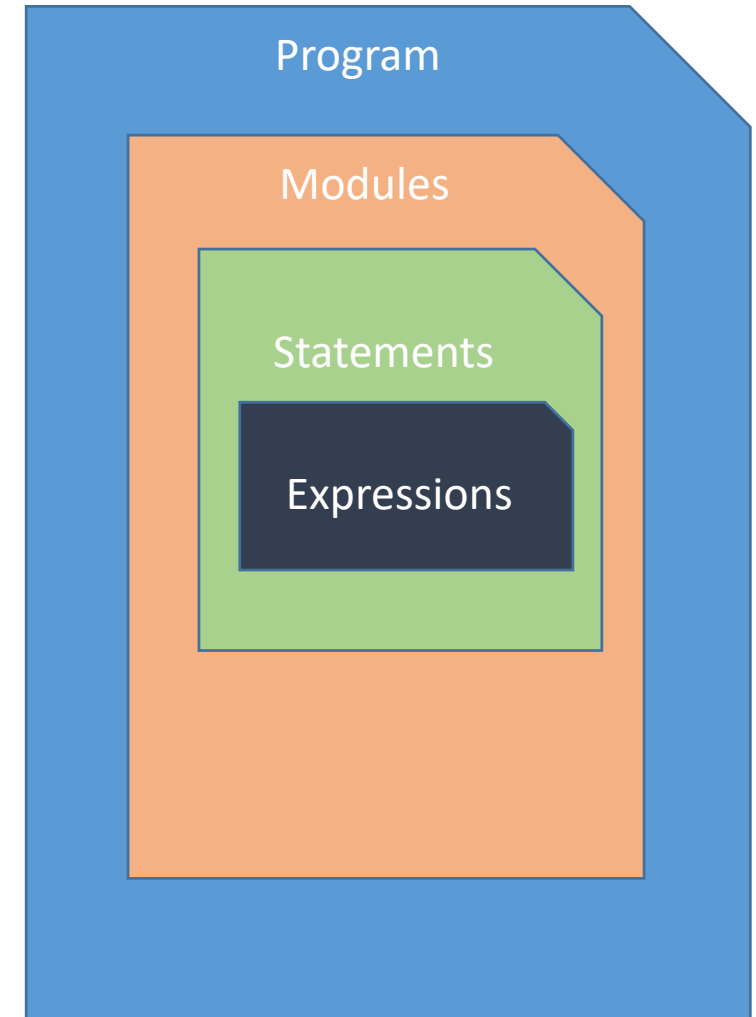
- Elements are accessed by index or offset

```
>>>my_tuple[1]  
'Ben'
```

- Immutable sequences and hence cannot be changed in place

Statements and Syntax

- Python program structure:
 1. Programs are composed of modules.
 2. Modules contain statements.
 3. *Statements contain expressions.*
 4. Expressions create and process objects



Statements and Syntax

A simple dice roller:

```
import random  
random.randint(1,6)
```



Statements and Syntax

`import random` ← Import statement

`random.randint(1,6)`

← Function inside the imported module

← Module name

Statements and Syntax

Statement	Role	Example
Assignment	Creating references	<code>a, b = 'good', 'bad'</code>
Calls and other expressions	Running functions	<code>log.write("spam, ham")</code>
print calls	Printing objects	<code>print('The Killer', joke)</code>
if/elif/else	Selecting actions	<code>if "python" in text: print(text)</code>
for/else	Iteration	<code>for x in mylist: print(x)</code>
while/else	General loops	<code>while X > Y: print('hello')</code>
pass	Empty placeholder	<code>while True: pass</code>
break	Loop exit	<code>while True: if exittest(): break</code>
continue	Loop continue	<code>while True: if skiptest(): continue</code>
def	Functions and methods	<code>def f(a, b, c=1, *d): print(a+b+c+d[0])</code>
return	Functions results	<code>def f(a, b, c=1, *d): return a+b+c+d[0]</code>
yield	Generator functions	<code>def gen(n): for i in n: yield i*2</code>
global	Namespaces	<code>x = 'old' def function(): global x, y; x = 'new'</code>
nonlocal	Namespaces (3.X)	<code>def outer(): x = 'old'</code>

Statement	Role	Example
		<code>def function(): nonlocal x; x = 'new'</code>
import	Module access	<code>import sys</code>
from	Attribute access	<code>from sys import stdin</code>
class	Building objects	<code>class Subclass(Superclass): staticData = [] def method(self): pass</code>
try/except/finally	Catching exceptions	<code>try: action() except: print('action error')</code>
raise	Triggering exceptions	<code>raise EndSearch(location)</code>
assert	Debugging checks	<code>assert X > Y, 'X too small'</code>
with/as	Context managers (3.X, 2.6+)	<code>with open('data') as myfile: process(myfile)</code>
del	Deleting references	<code>del data[k] del data[i:j] del obj.attr del variable</code>

Statements and Syntax

- Conditional Statements :
 - “If/elif/else” evaluates a condition and executes the enclosed box whether the evaluation returns true or false

```
if age > 18 :  
    print("adult")  
else:  
    print(" minor")
```

Statements and Syntax

- for/else

- `for` is a loop statement which is used to iterate over a sequence (like a list, dictionary, set or string)

```
for eachLetter in "AbRaCaDabRa":  
    print(eachLetter)
```

- `break` can be used to exit from a loop .

```
for eachNumber in [4,8,12,15,20,24]:  
  
    if eachNumber % 4 == 0:  
        print(str(eachNumber)+ " is a Multiple of 4")  
    else:  
        print(str(eachNumber)+" is a Non multiple ! Exiting!")  
        break
```

- `else` clause of for loop executes if there is no `break` statement encountered .

```
for eachNumber in [4,8,12,16,20,24]:  
  
    if eachNumber % 4 == 0:  
        print(str(eachNumber)+ " is a Multiple of 4")  
    else:  
        print(str(eachNumber)+" is a Non multiple ! Exiting!")  
        break  
  
else:  
    print("no non multiples of 4 found")
```

Statements and Syntax

- **while/else**

- `while` loop executes a set of statements as long as a condition is true.

```
i = 1
while i < 6:
    print(i)
    i += 1
```

- Same as `for` loop `while` loop exits if `break` is encountered
- `else` part of `while/else` executes if no `break` is encountered.

Statements and Syntax

- Pass is an empty placeholder
- Continue transfers the control back to the top of the loop
- Return sends the output of a function back to the function caller.
- Yield works just like return but for generator functions
- Try/except/finally are used for exception handling
- Raise is used to trigger exceptions
- With/as is used for context management (file handling etc)

Functions

- rather than having multiple redundant copies of an operation's code, we can factor it into a single function.(Code Reuse)
- If the operation must be changed later, we have only one copy to update in the function, not many scattered throughout the program.
- Functions let us split complex systems into manageable parts

Functions

- Create functions:
 - `def` is the keyword used to create functions

```
def name(arg1, arg2,... argN):  
    statements
```

- `return` statement can show up anywhere in a function body; when reached, it ends the function call and sends a result back to the caller.
- `yield` statement works just like `return` but remembers where it left off. It is used in Generators.

Functions

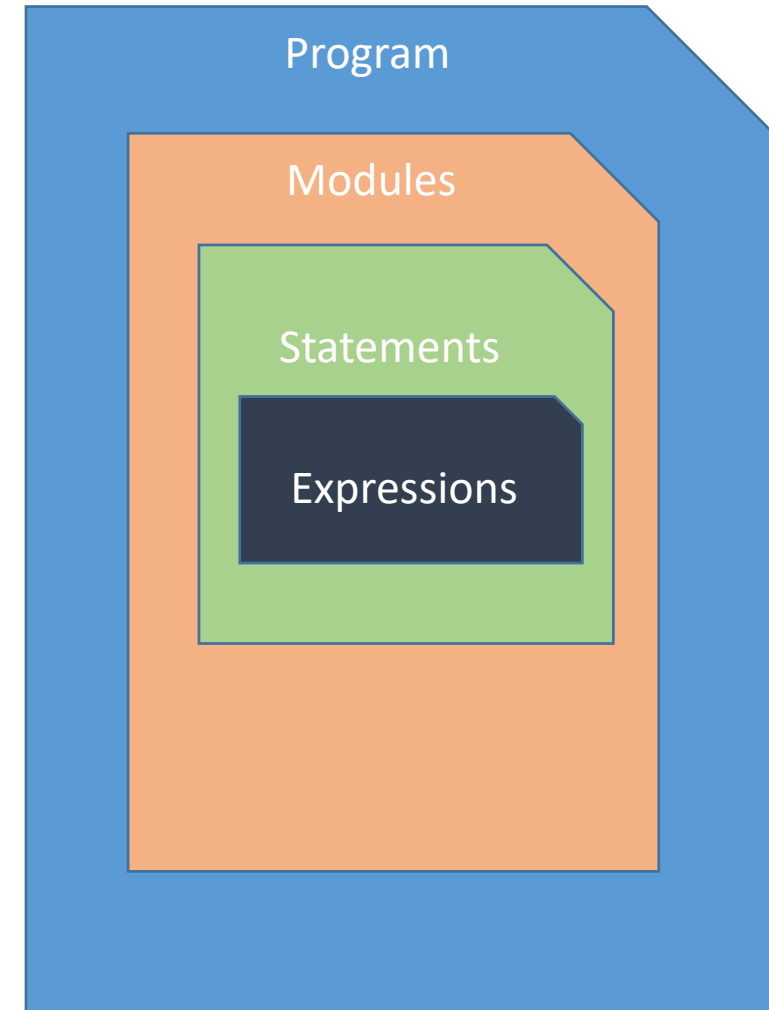
- Lambdas are anonymous functions
- Often used as a way to inline a function definition, or to defer execution of a piece of code
- Lambdas are expressions and not statements
- Lambda body is a single expression and not a block of statements
- Allows you to embed a function's definition within the code that uses it

Comprehensions

- Python supports the procedural, object-oriented, and function programming paradigms
- Comprehension is one such functional programming tool which helps us to combine functions in many powerful ways
- Type of comprehensions :
 - List comprehension
 - Set comprehension
 - Dictionary comprehension
 - Generator expressions

Modules

- Modules are the highest-level program organization unit which provides :
 - Packaged program code and data for reuse
 - Self containing namespaces to minimize variable name clashes
- Typically each python program file is a module
- Modules import other modules



Modules

- Importing a module
 - Import keyword lets us import a module as a whole
 - From keyword lets us import particular names from the module
- Purpose
 - Helps in code reuse
 - Helps in system namespace partitioning
 - Implementing shared services or Data

Packages

- Packages are basically folders containing multiple modules
- for organizing the files in a large system and tends to simplify module search path settings.
- With packages we no longer need to arrange unrelated files in a flat directory structure

Classes and Object Oriented Programming

- Classes are Python's main object-oriented programming (OOP) tool
- By using classes it becomes really easy to model real world relationships
- Instances of the same class have same behaviour.

e.g :

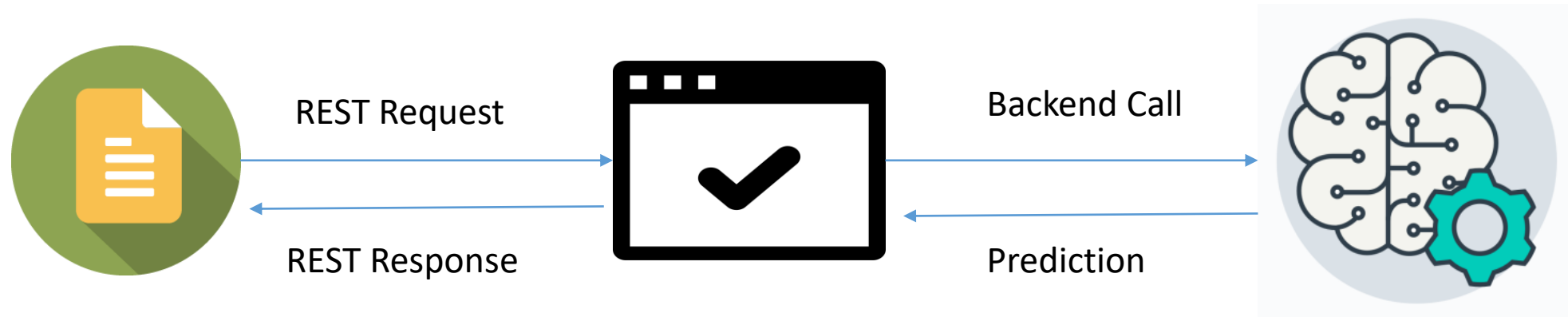
- a. a. PRIUS is kind of a car
- b. b. A PRIUS is composed of objects of other types of classes like wheel, engine, battery, power-train etc.



Classes and OOP

- The class statement creates a class object and assigns it a name
- Assignments inside class statements make class attributes
- Class attributes provide object state and behaviour
- Classes are customized by using Inheritance
- Addition of logic is done by changing subclass

Machine Learning Primer

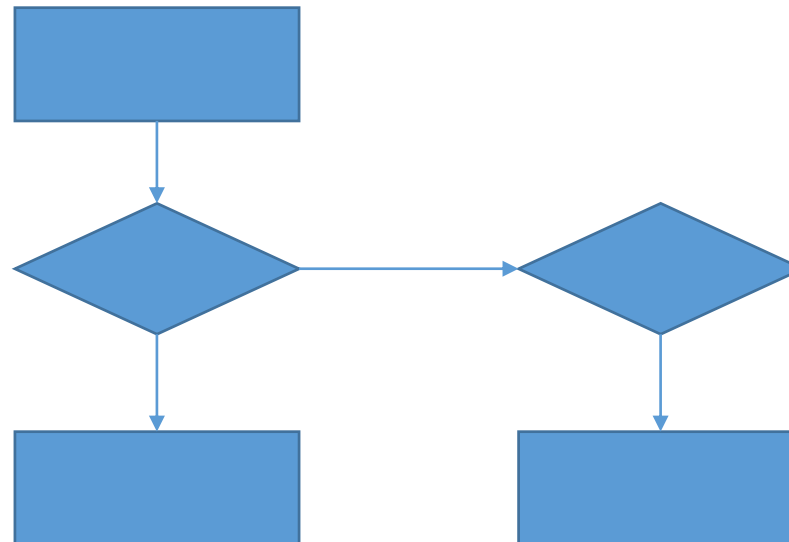


“A computer program is said to learn from experience E with respect to some task T and some performance measure P , if its performance on T , as measured by P , improves with experience E .”

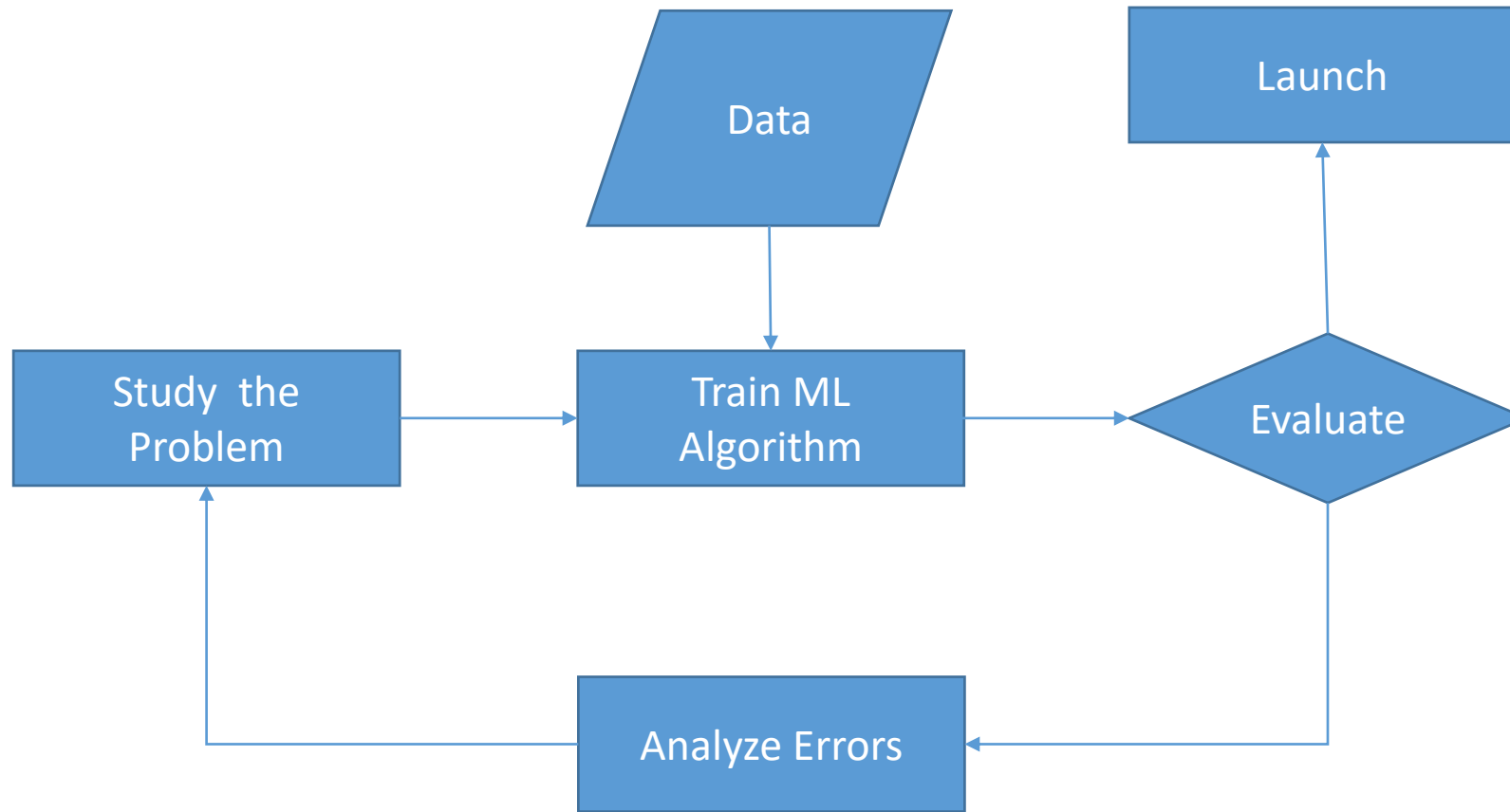
—Tom Mitchell, *1997*

Rule Based Systems :

- A combination of multiple binary statements or blocks of statements,
- after the evaluation of each statement or block, the system enters a state.

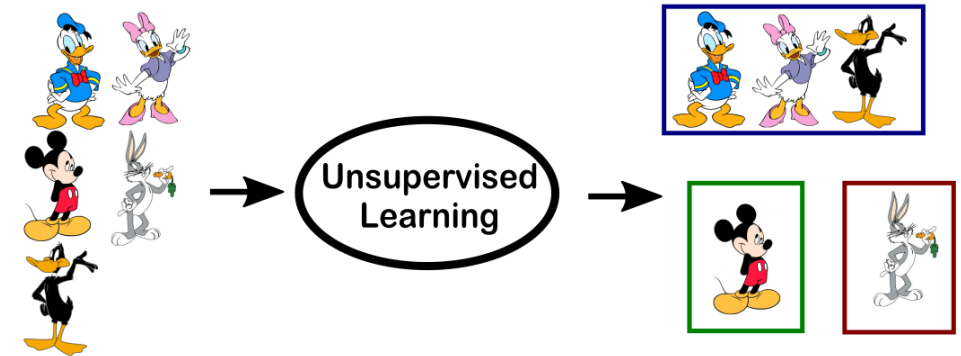
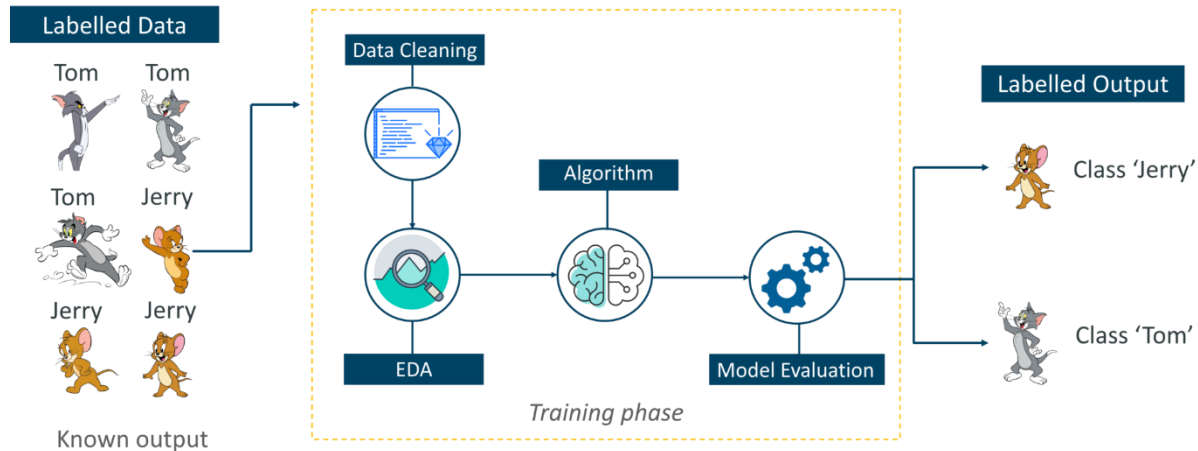


Machine Learning Concepts



Machine Learning Classification

Supervised vs Unsupervised



Machine Learning Classification

Online vs Batch learning

Online ML Systems

- In Online learning the data is fed either sequentially or in mini batches.
- Useful for systems which need to evolve fast.
- Useful for systems which receive data in a continuous flow.

Batch ML Systems

- The system learns incrementally on large chunks of data
- Training has to be done from scratch if new data has to be added
- Batch ML systems are more resource intensive

Machine Learning Classification

Instance Based vs Model Based

Instance based ML Systems

- System learns the examples and uses a similarity measure to generalize new cases

Model based ML Systems

- Instead of generalizing using a Similarity measure , a model is built from the data provided.
- The created model is then used to make predictions

Data, The Fuel

- Data is the fuel, Algorithm is the engine.
 - Machine Learning systems can go wrong either because of:
 - Wrong Choice of Algorithm
 - Data of bad quality
- Problems with data:
 - Insufficient Data
 - Non Representative data
 - Dirty Data
 - Data with Irrelevant features
 - Data with Insufficient features

Handling the Data

- The data should be divided into Training, Test and Validation datasets as a best practice.
- Preprocess the data to handle any rows that many have missing or incorrect values*
- Many ML algorithms prefer to work with numbers. So, for such cases a strategy to represent features as numbers is required.*
- Features in data might have very different scales. ML systems do not perform well in such cases. Thus Feature Scaling is required.*

Handling Text

- Encoding is the process that is used to assign numeric value to text attributes.
 - LabelEncoder *
 - Assigns an Integer value to each category of a particular feature
 - OneHotEncoder *
 - Creates one binary attribute per category .One attribute is hot(1) while others are cold(0)
 - LabelBinarizer *
 - Applies both the LabelEncoder and OneHotEncoder in one go.

Feature Scaling

Feature scaling is a method used to normalize the range of independent variables or features of data

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

MIN-MAX Scaling

$$x' = \frac{x - \bar{x}}{\sigma}$$

Standardization

Sklearn primer

Estimators. Any object that can estimate some parameters based on a dataset is called an *estimator*. It is performed by *fit()* method

Transformers. Some estimators (such as an imputer) can also transform a dataset; these are called *transformers*. It is performed by *transform()* method

Predictors : some estimators are capable of making predictions given a dataset; they are called *predictors*. It is performed by the *predict()* method.

Scikit-learn Deep Dive

- Estimator:
 - estimator interface is at the core of the library. It defines instantiation mechanisms of objects and exposes a ***fit()*** method for learning a model from training data.
 - All supervised and unsupervised learning algorithms (e.g., for classification, regression or clustering) are offered as objects implementing this interface
 - Machine learning tasks like feature extraction, feature selection or dimensionality reduction are also provided as estimators

Scikit-learn Deep Dive

- Transformers
 - some estimators implement a transformer interface which defines a **transform()** method
 - Preprocessing, feature selection, feature extraction and dimensionality reduction algorithms are all provided as transformers within the library
 - Every transformer allows **fit()** and **transform()** to be written as **fit_transform()** instead of being chained together
 - The combined **fit_transform()** method prevents repeated computations

Scikit-learn Deep Dive

- Predictor
 - Predictor interface extends the notion of an estimator by adding a predict method.
 - In the case of supervised learning estimators, this method typically returns the predicted labels or values computed by the model.
 - They provide a **score** function to assess their performance on a batch of input data

Scikit-learn Deep Dive

```
from sklearn.preprocessing import OneHotEncoder
```

Estimator (Transformer)

```
encoder = OneHotEncoder()
```

```
housing_cat_1hot = encoder.fit_transform(housing_cat_encoded.reshape(-1,1))
```

```
Print(housing_cat_1hot)
```

fit_transform() implementation

Scikit-learn Deep Dive

```
from sklearn.linear_model import LinearRegression
```

Estimator (Predictor)

```
lin_reg = LinearRegression()
```

fit ()

```
lin_reg.fit(data, labels)
```

predict() (Inference)

```
lin_reg.predict(some_test_data)
```

Objective Function

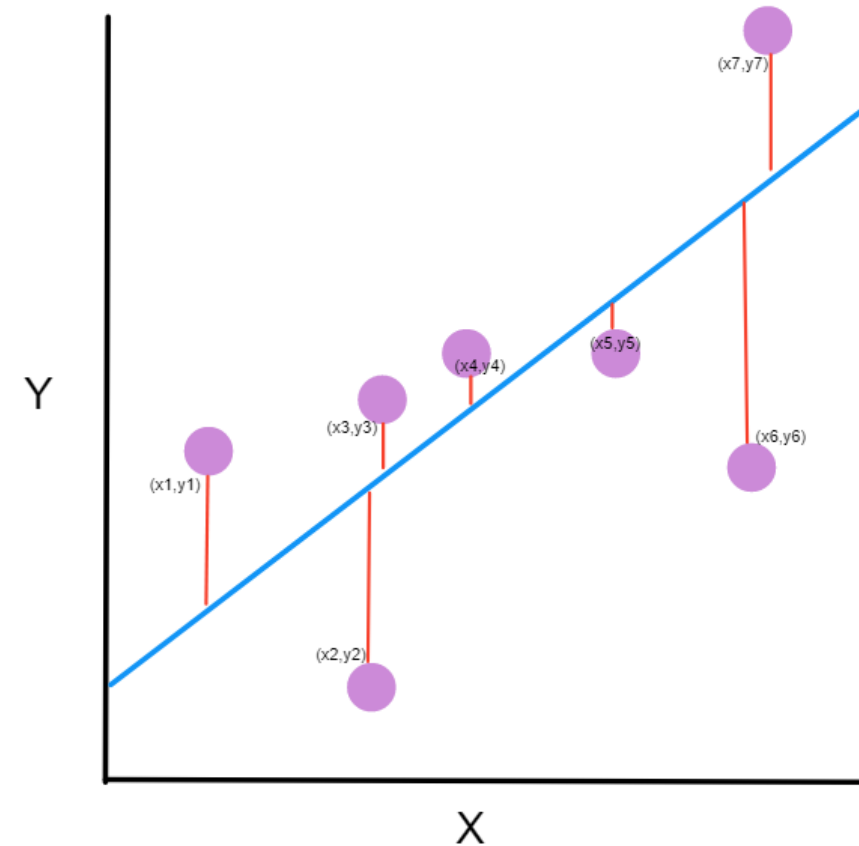
In Machine Learning **Objective Function** is the mathematical representation of the goal that we need to achieve.

For example : $h(x)=mx + c$ is the representation of a straight line. Since we try to draw a straight line in Linear Regression problem, we can take it as our objective function.

Understanding Error

Regression Problems

- The Idea is to find a line which fits best between the data points
- the **purple dots** are the points on the graph. Each point has an x-coordinate and a y-coordinate.
- The **blue line** is our prediction line. This is a line that passes through all the points and fits them in the best way. This line contains the predicted points.
- The **red line** between each purple point and the prediction line are the **errors**. Each error is the distance from the point to its predicted point.



RMSE

Root Mean Square Error

$$\text{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m} \sum_{i=1}^m \left(\underset{\text{Error}}{h(\mathbf{x}^{(i)})} - y^{(i)} \right)^2}$$

Root function

Mean of m terms

MAE

Mean Absolute Error

$$\text{MAE}(\mathbf{X}, h) = \frac{1}{m} \sum_{i=1}^m \left| h(\mathbf{x}^{(i)}) - y^{(i)} \right|$$

Modulus Function
for absolute value

Mean of m terms

Error

** RMSE is more sensitive to outliers in the data. Why?

Norms are measures of distance between the target and predicted values.

RMSE corresponds to the Euclidean Norm, also know as the ℓ_2 norm.

MAE corresponds to the Manhattan Norm , also know as the ℓ_1 norm.

General form of norm :

$$\| \mathbf{v} \|_k = \left(|v_0|^k + |v_1|^k + \cdots + |v_n|^k \right)^{\frac{1}{k}} \cdot \ell_0$$

Machine Learning Algorithms

Supervised ML Algorithms

- k-Nearest Neighbors
- Linear Regression
- Logistic Regression
- Support Vector Machines (SVMs)
- Decision Trees and Random Forests

Unsupervised ML Algorithms

- Clustering
 - k-Means
 - Hierarchical Cluster Analysis (HCA)
 - Expectation Maximization
- Visualization and dimensionality reduction
 - Principal Component Analysis (PCA)
 - Kernel PCA
 - Locally-Linear Embedding (LLE)
 - t-distributed Stochastic Neighbour Embedding (t-SNE)

Logistic Regression:

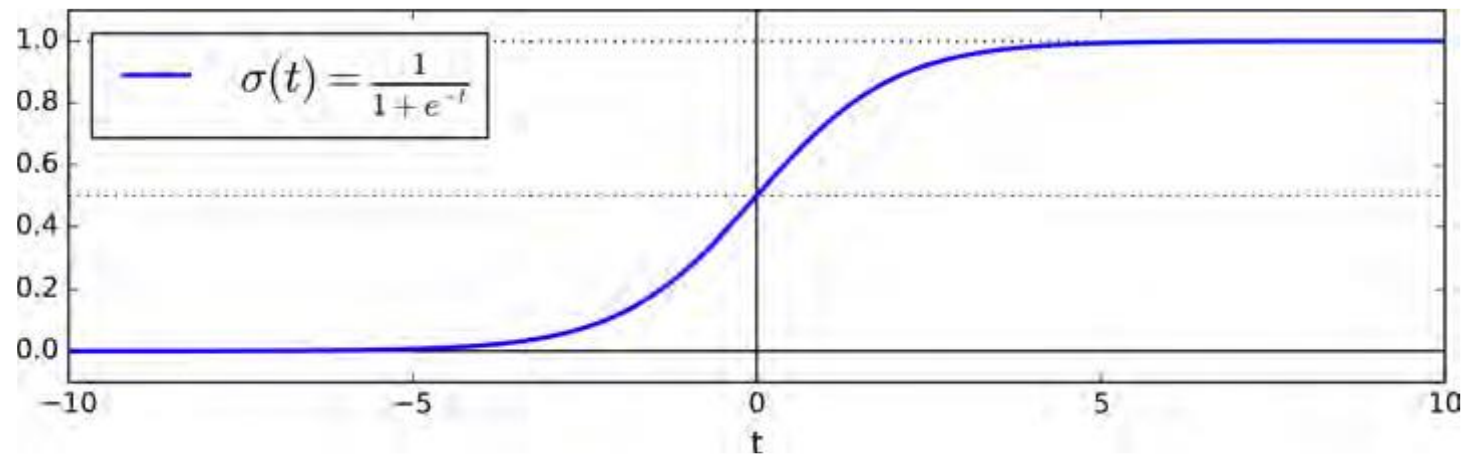
- Used to estimate the probability that an instance belongs to a particular class.
- If the estimated probability is greater than 50%, then the model predicts that the instance belongs to that class (called the positive class, labeled “1”), or else it predicts that it does not (i.e., it belongs to the negative class, labeled “0”). This makes it a binary classifier.

$$\hat{p} = h_{\theta}(\mathbf{x}) = \sigma(\theta^T \cdot \mathbf{x})$$

Logistic Regression :

- The logistic—also called the *logit*, noted $\sigma(\cdot)$ —is a *sigmoid function* (i.e., S-shaped) that outputs a number between 0 and 1

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$



Machine Learning Algorithms

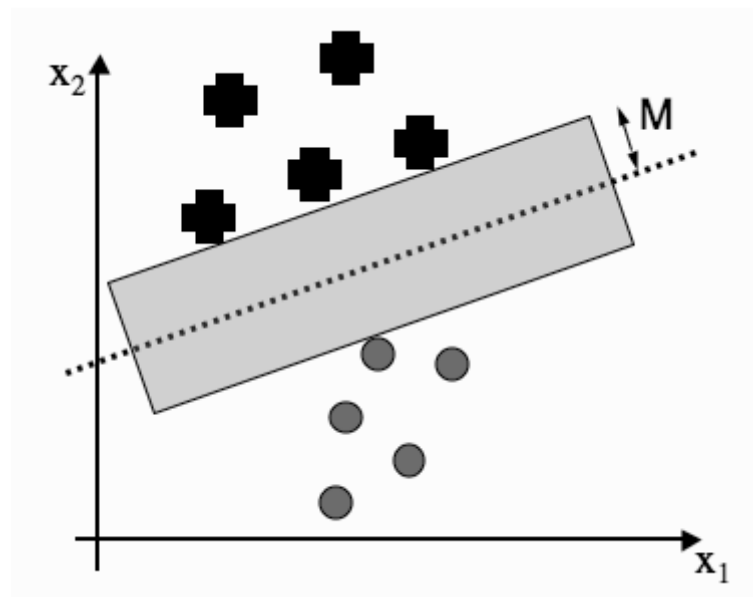
Logistic Regression:

The Logistic Regression Model prediction can be written as :

$$\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5, \\ 1 & \text{if } \hat{p} \geq 0.5. \end{cases}$$

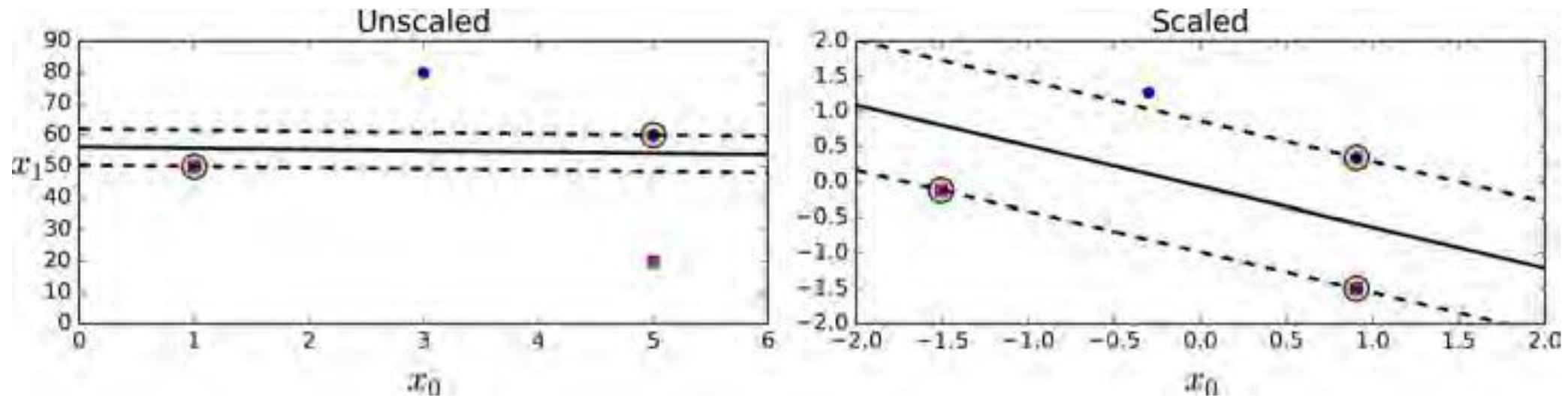
Support Vector Machine

- A *Support Vector Machine* (SVM) is a very powerful and versatile Machine Learning model, capable of performing linear or nonlinear classification, regression, and even outlier detection



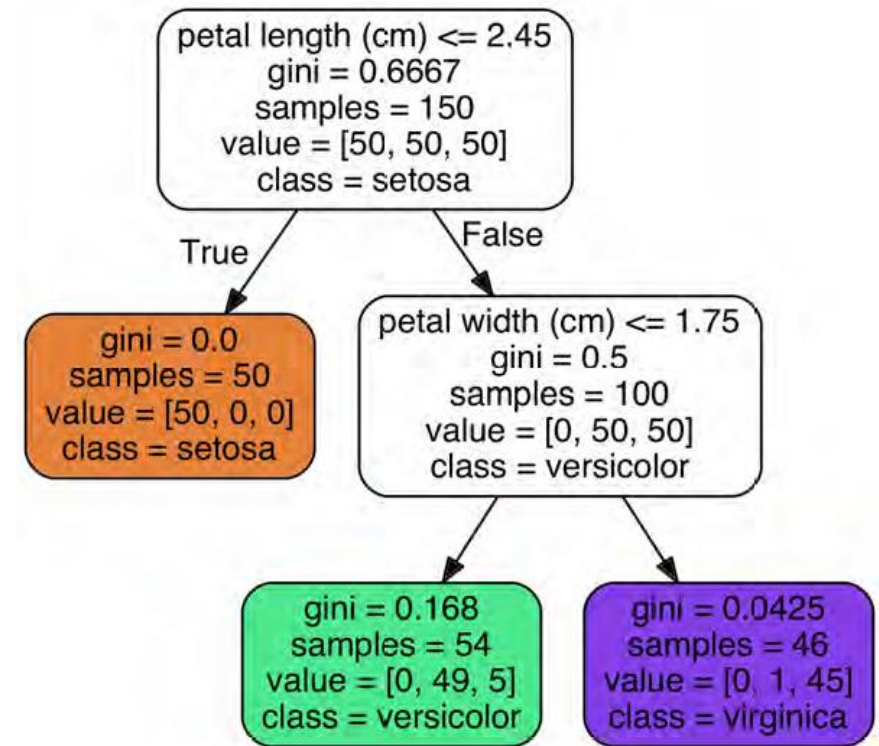
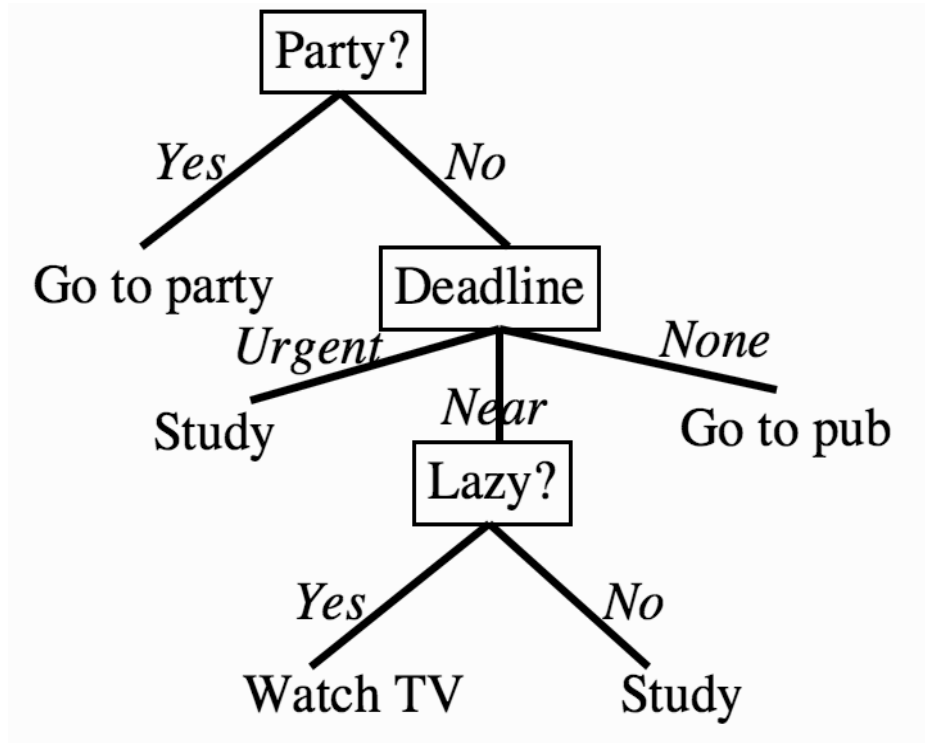
Machine Learning Algorithms

Support Vector Machines:
SVMs are sensitive to the feature scales



Machine Learning Algorithms

Decision Trees :



Decision Trees

- the algorithms build the tree in a greedy manner starting at the root, choosing the most **informative** feature at each step
- At each stage, you choose a feature that gives you the most information given what you know already
- Decision Trees make very few assumptions about the training data (as opposed to linear models, which may assume that the data is linear, for example)

Q & A

Questions ?

Thank You !