Introducción a la Inteligencia Artificial

Clase 1

Régimen de aprobación

- Asistencia
- 1 entregable individual (TP)
- 1 trabajo grupal (TG)
- **Nota =** 0.4 TP + 0.6 TG

→ se entrega 1 semana dsp. clase 8
Fecha límite (26/08/23)

→ se entrega clase 8

Cronograma

(C2 - C5) Ap. supervisado
(C6 - C7) Ap. no sup

Clase 1: Introducción a AI. Python y Numpy para AI.
Clase 2: PCA y K-mean. (C6)
Clase 3: Aprendizaje estadístico (Regresión Lineal). Esperanza Condicional, ECM, Máxima Verosimilitud. (C2)
Clase 4: Bayes y estimadores puntuales (bias, variance, etc). (C3)
Clase 5: Gradientes, Optimización, Hiperparámetros y Regularización. (C4)
Clase 6: Regresión Logística. Clasificación binaria. Softmax. (C5)
Clase 7: Algoritmos no supervisados. Expectation Maximization.
Clase 8: Exposición de los trabajos + Semana para entregar trabajo individual.

Dinámica esperada para las clases (idealmente, puede fallar):

- 50 minutos de teoría
- 10 minutos de descanso
- 50 minutos de ejemplos y ejercicios
- 10 minutos de descanso
- 60 minutos de programación

Herramientas

- **Lenguaje de programación**:
  - Python ^3.8
  - Herramienta pip para instalar librerias de codigo y dependencias (Virtualenv) virtual environments
- **Librerías**:
  - Numpy
  - SciPy
  - statsmodels
  - sklearn
  - Matplotlib
- **Entornos de trabajo:**
  - Jupyter-notebooks, Google Colab, IDE, iPython

↳ Framework basado en IPython

La bibliografía es solo a modo de sugerencia y no será obligatorio el uso de dicho material. El curso está diseñado para ser completamente autocontenido.

- The Elements of Statistical Learning | Trevor Hastie | Springer (Re sandísimo)
- An Introduction to Statistical Learning | Gareth James | Springer (más amigable)
- Deep Learning | Ian Goodfellow | https://www.deeplearningbook.org/
- Stanford | CS229T/STATS231: Statistical Learning Theory
- Mathematics for Machine Learning | Deisenroth, Faisal, Ong
- Artificial Intelligence, A Modern Approach | Stuart J. Russell, Peter Norvig

Índice

1. Introducción a la materia
2. Definición de AI
3. Clasificación de algoritmos de AI
4. Herramientas matemáticas para AI
5. Herramientas de programación para AI
6. Introducción a Python
7. Introducción a Numpy
8. Ejercicios básicos Numpy
9. Ejercicio de aplicación (K-means)
10. Bibliografía

Definición de Artificial Intelligence (AI)

Pensar humanamente	Pensar racionalmente
"AI is the automation of activities that we associate with human thinking, activities such as decision-making, problem solving, and learning" (Bellman, 1978)	"AI is the study of the computations that make it possible to perceive, reason, and act." (Winston, 1992)"
Actuar humanamente	**Actuar racionalmente**
"AI is the study of how to make computers do things at which, at the moment, people are better." (Rich and Knight, 1991)	"AI is concerned with intelligent behavior in artifacts." (Nilsson, 1998)

proceso del pensamiento

proceso del comportamiento

comparación de éxito contra el resultado humano

comparación de éxito contra el resultado ideal

Definición de Artificial Intelligence (AI)

- Test de Turing propuesto por Alan Turing en 1950.

"Una computadora pasa el Test de Turing si un interrogante humano no puede diferenciar si las respuestas provienen de otra persona o de una computadora"

- El Test prohíbe la interacción física directa entre el interrogante y la computadora, ya que según el test simular físicamente un humano es innecesario para definir inteligencia.

- Pasar el Test de Turing rigurosamente requiere mucho trabajo:
  - Natural Language Processing
  - Knowledge Representation
  - Automated Reasoning

IBM Watson Jeopardy! challenge

Definición de Machine Learning (ML) (clásico)

"Es la capacidad de un agente de Inteligencia Artificial de mejorar su rendimiento en futuras tareas después de hacer observaciones en el mundo donde el algoritmo se desempeña."

¿Por qué el aprendizaje a partir de la observación es importante?

Porque es imposible…

1. anticipar todas las situaciones en la que se encontrará el agente.
2. anticipar todos los cambios en el tiempo que se producirán.
3. dar una solución sin usar el aprendizaje por observaciones.

Definición de Machine Learning supervisado

"El agente de AI observa ejemplos de pares entrada-salida y aprende una función que mapea la entrada a la salida."

En términos matemáticos, a grandes rasgos:

Observar ejemplos del vector aleatorio $\vec{X}$, $\bar{Y}$

$\bar{X}$ es el dato de entrada
$\bar{Y}$ es el valor real

Aprender a predecir Y a partir de observaciones del vector aleatorio X, estimando la función densidad de probabilidad de Y condicionada al vector aleatorio X.

$$f_{Y|\vec{X}}(y|\vec{x})$$

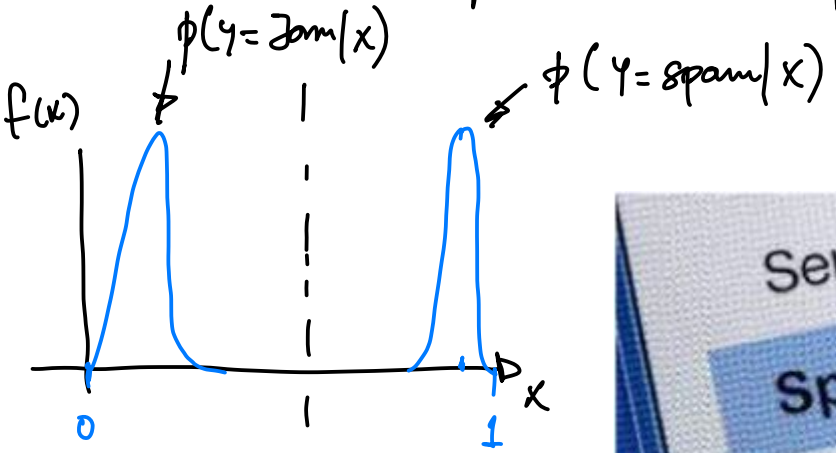
$\hat{f}_{Y|X}$ ← estimación

Machine Learning supervisado - Ejemplos

Detección de objetos - Segmentación de imagenes.

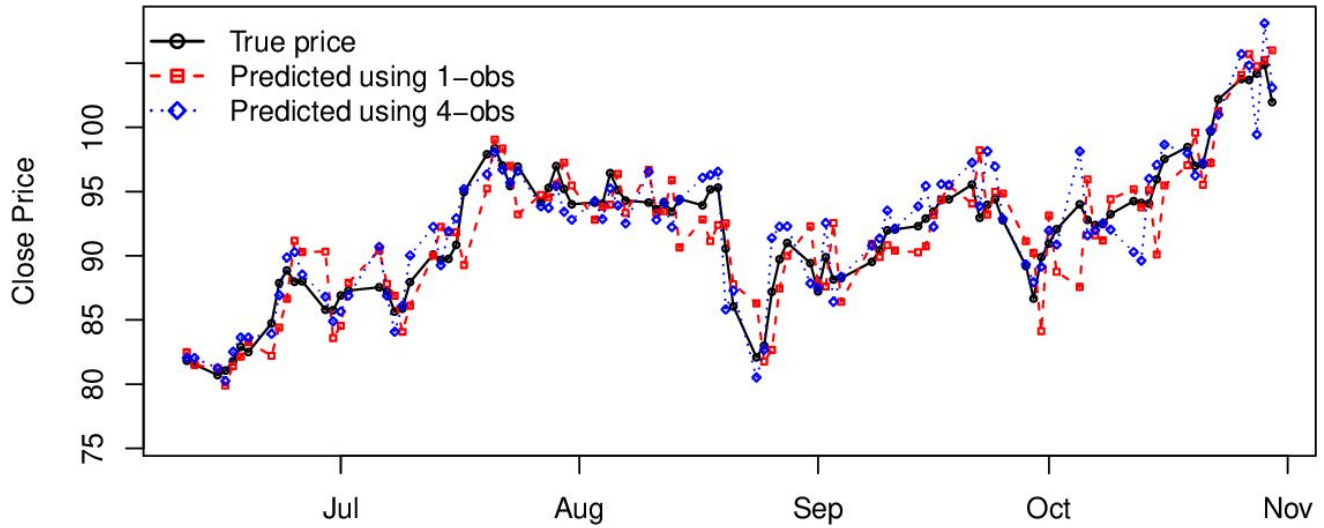
Machine Learning supervisado - Ejemplos

Filtro de spam



Machine Learning supervisado - Ejemplos

Precios de activos financieros $\vec{X}(t) \rightarrow$ series temporales



Definición de Machine Learning no supervisado

"El agente de AI observa datos y aprende patrones sin necesidad de utilizar feedback explícito."

En términos matemáticos, a grandes rasgos:

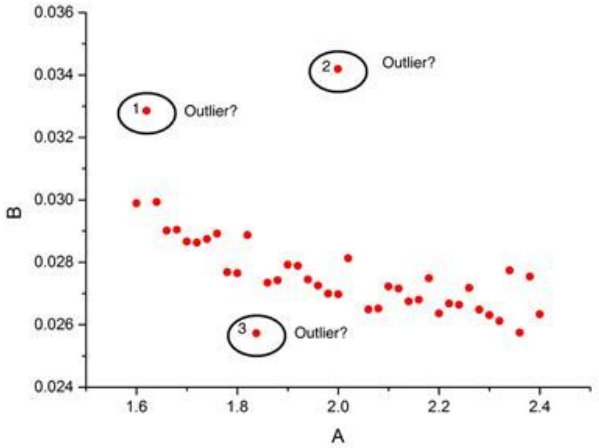
Observar ejemplos del vector aleatorio $\vec{X}$

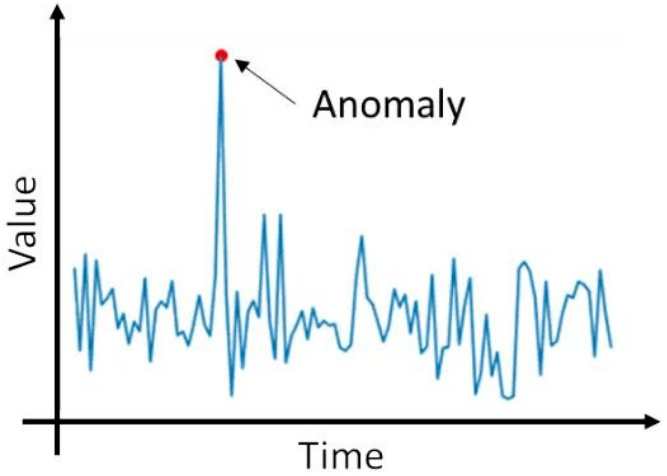
Aprender implícitamente o explícitamente la función de densidad aproximada del vector aleatorio X:

$$f_{\vec{X}}(\vec{x}) \qquad \hat{f}_{\vec{X}}$$

Machine Learning no supervisado - Ejemplos

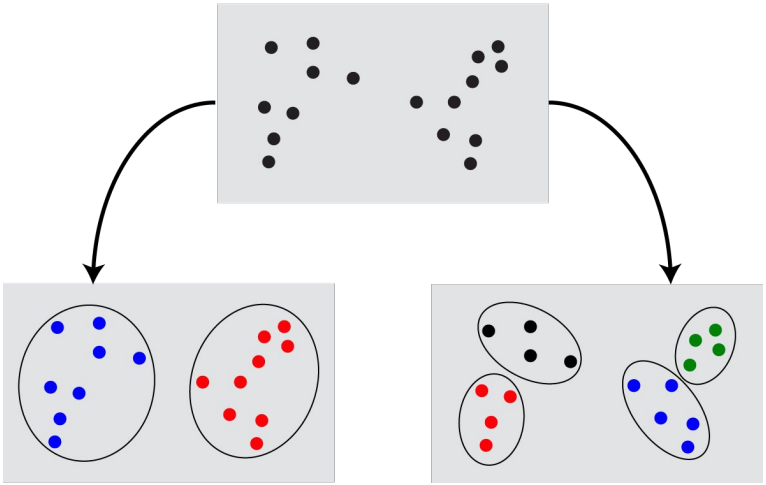
Detección de anomalías





Machine Learning no supervisado - Ejemplos

Segmentación / Clustering



Machine Learning no supervisado - Ejemplos

Sistemas de recomendación

Relación entre modelos supervisados y no supervisados (a grandes rasgos)

Podemos resolver el problema no-supervisado como n problemas supervisados.

$$f_{X_2,X_1}(x_2,x_1) = f_{X_2|X_1}(x_2|x_1)f_{X_1}(x_1)$$

$$f_{X_3,X_2,X_1}(x_3,x_2,x_1) = f_{X_3|X_2,X_1}(x_3|x_2,x_1)f_{X_2,X_1}(x_2,x_1)$$

$$f_{X_3,X_2,X_1}(x_3,x_2,x_1) = f_{X_3|X_2,X_1}(x_3|x_2,x_1)f_{X_2|X_1}(x_2|x_1)f_{X_1}(x_1)$$

$$f_{\vec{X}}(\vec{x}) = \prod_{i=1}^{n} f_{X_i|X_1,...,X_{i-1}}(x_i|x_1,...,x_{i-1})$$

Podemos resolver el problema supervisado a partir de uno no supervisado.

$$f_{Y|\vec{X}}(y|\vec{x}) = \frac{f_{Y,\vec{X}}(y,\vec{x})}{f_{\vec{X}}(\vec{x})}$$

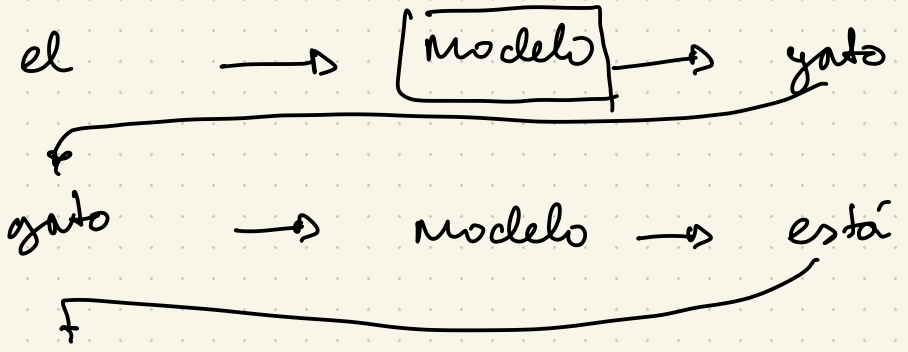
Definición de Machine Learning semi-supervisado

"El agente de AI combina técnicas no-supervisadas y supervisadas para resolver problemas."

Casos donde no se tiene la variable target para todas las muestras.

Clasificación → Pseudo-etiquetas (Propagación de etiquetas)
Regresión →Reducción de dimensionalidad

<begin>



modelo → <end>

<begin> el gato esta sobre la mesa <end>

Definición de Reinforcement Learning (RL)

"El agente de AI aprende a partir de feedback explícito obtenido a través de una función de recompensas y castigos. El agente conoce las acciones que puede ejecutar y a medida que ejecuta las acciones aprende con el objetivo de maximizar la recompensa."

RL - Ejemplo

Definición de Deep Learning

"Deep Learning es una técnica para problemas de ML supervisados, que agrega capas y parámetros para aprender complejos mapeos de entrada-salida, utilizando modelos no lineales."

Ian Goodfellow

En general, requiere grandes volúmenes de datos y poder de cómputo

MLP: Multilayer Perceptron

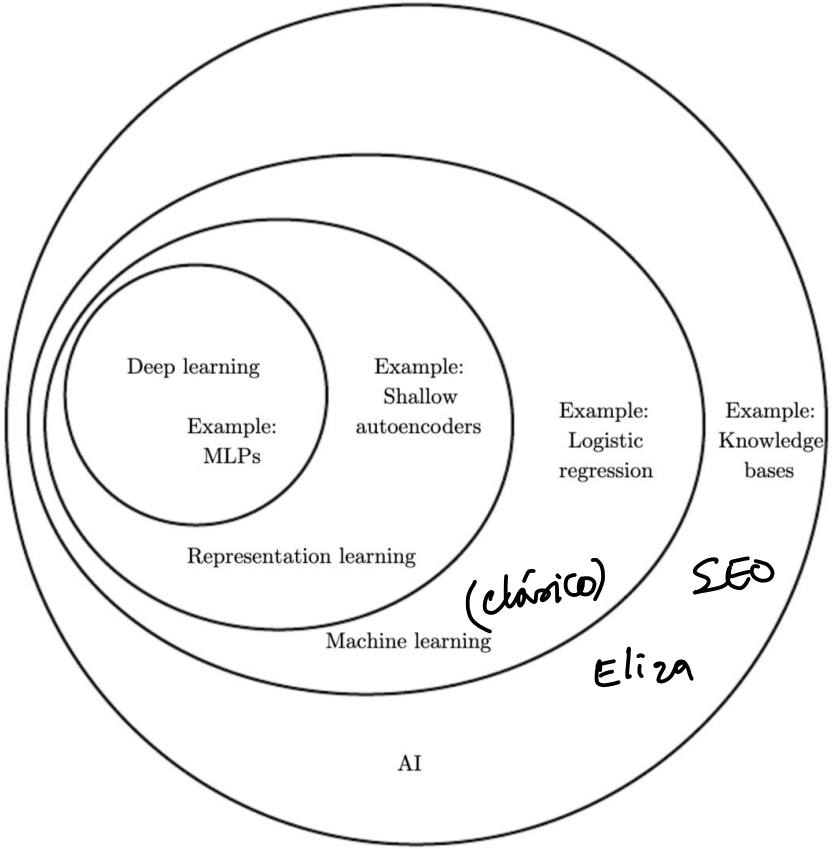
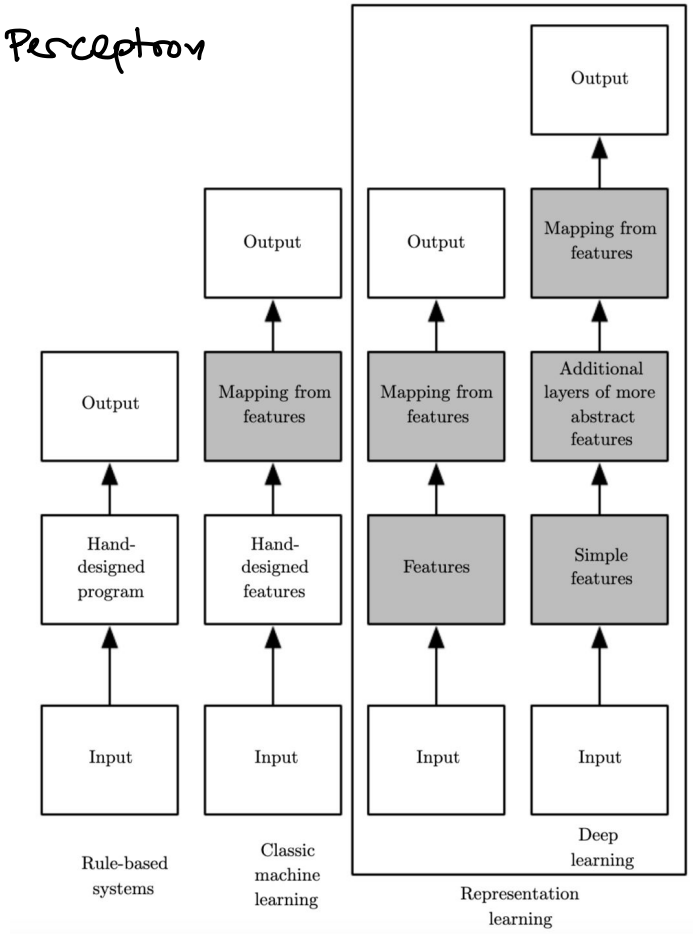


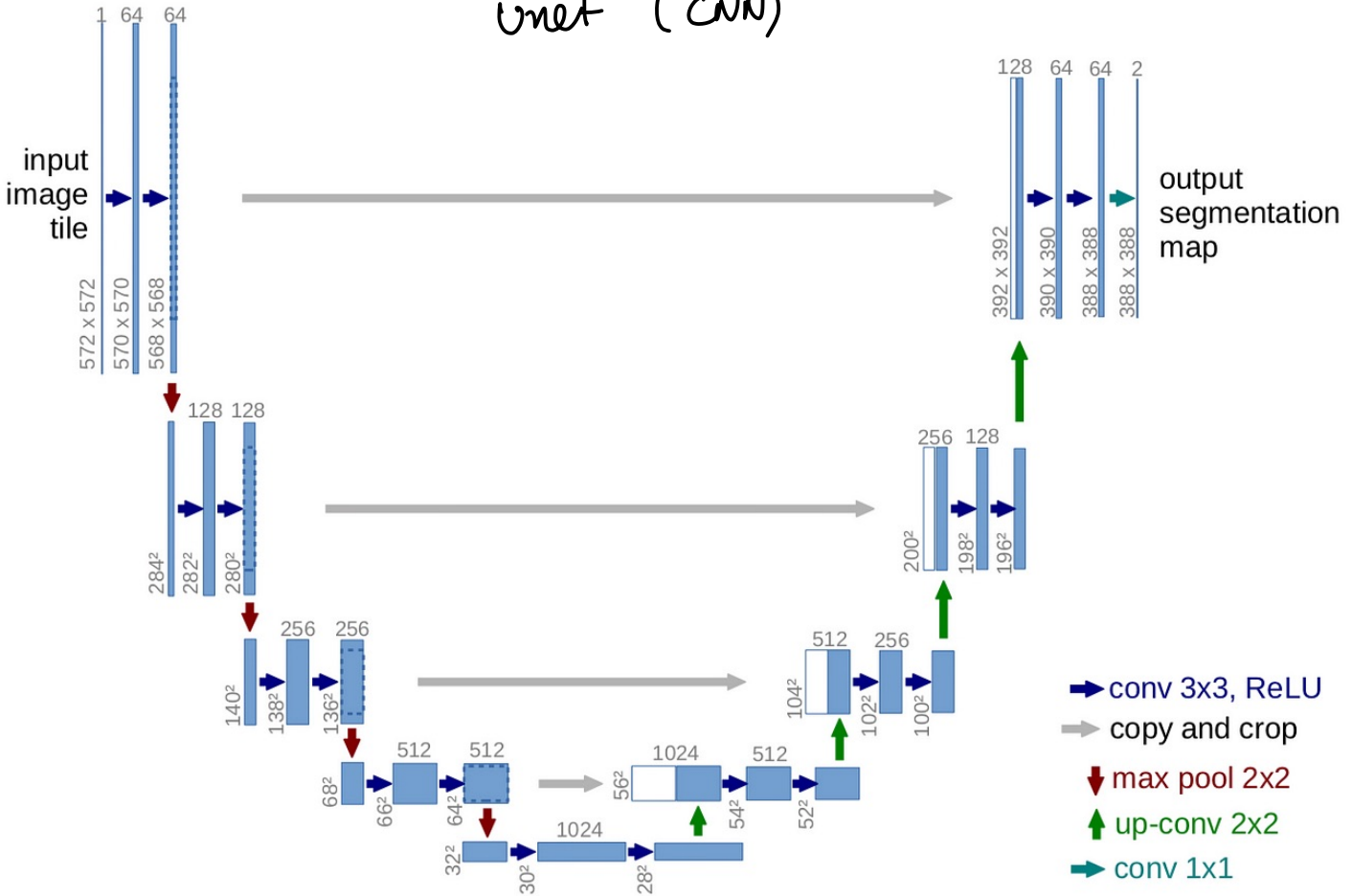
Diagrama de Venn de algoritmos



Bloques que se aprenden en Deep Learning

unet (CNN)

x

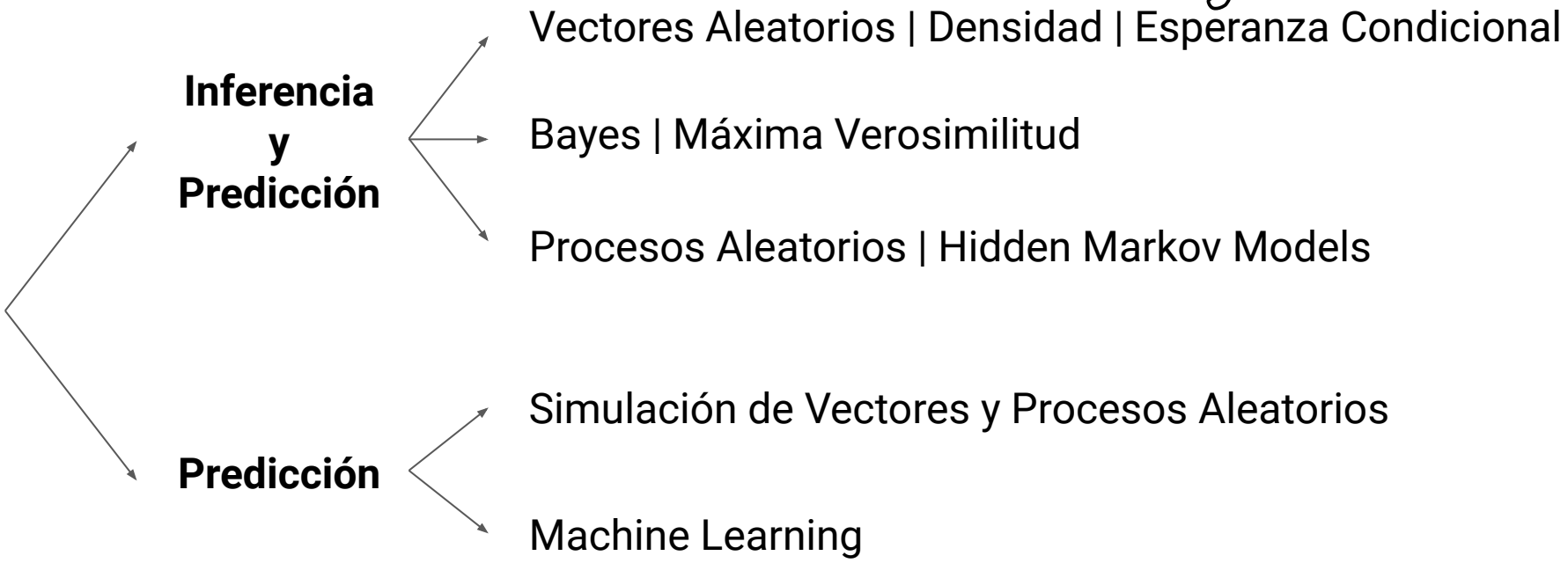


$x \in \mathbb{R}$

Estrategias

$f_{X,Y}$ inferir $\rightarrow$ $\hat{f}_{XY}$ encontrar el $\hat{f}$ $L(f - \hat{f})$

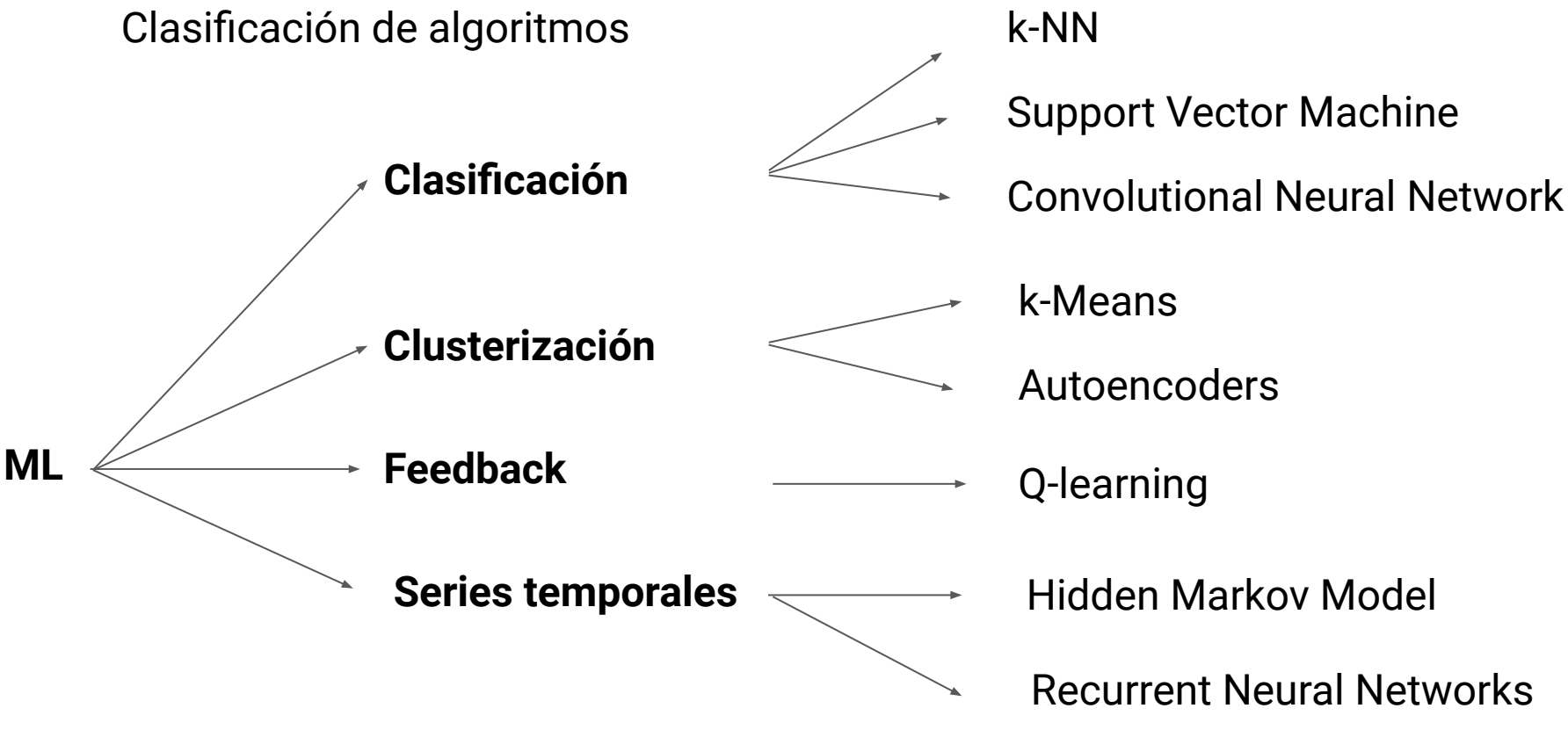
predecir $\rightarrow$ g_{XY} encontrar $\hat{y} = g_{XY}(x) \rightarrow \min_g \|y - \hat{y}\|^2$



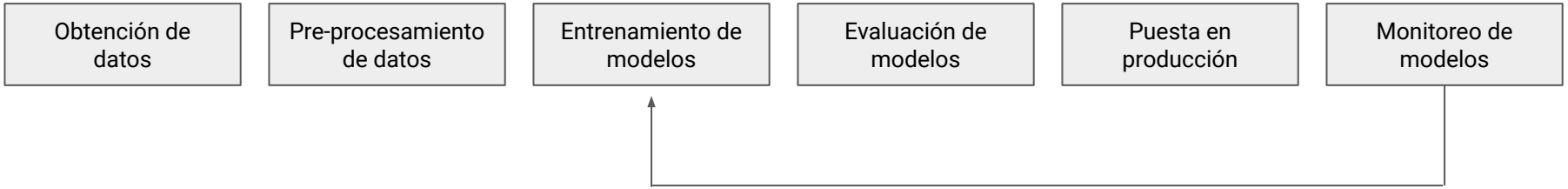
Clasificación de algoritmos de AI | ¿Cuándo usar cada estrategia?

X es un vector aleatorio, Y es un vector aleatorio, n es cantidad features de X

Datos	Modelo	Problema	Estrategia
Conozco densidad de X, n chico	Conozco relación entre X e Y	Inferencia y predicción	Teoría de probabilidad
Conozco densidad de X, n chico	Conozco Y si condiciono a X	Inferencia y predicción	Esperanza condicional
Conozco familia de X, n chico	Conozco Y si condiciono a X	Inferencia y predicción	Teoria de Bayes
Conozco familia de X(t), n chico	Conozco modelo del proceso	Inferencia y predicción	Procesos aleatorios
Conozco familia de X(t), n grande	Conozco modelo del proceso	Predicción	Simulación
No conozco X, n grande	No conozco relación entre X e Y	Predicción	Machine Learning supervisado
No conozco X, n grande	-	Predicción	Machine Learning no supervisado


Clasificación de algoritmos
ML
Clasificación
Clusterización
Feedback
Series temporales
k-NN
Support Vector Machine
Convolutional Neural Network
k-Means
Autoencoders
Q-learning
Hidden Markov Model
Recurrent Neural Networks

Entendimiento del problema



Data Engineer

Data Scientist

AI Researcher

Machine Learning Engineer

MLOps

“When you’re fundraising, it’s AI. When you’re hiring, it’s ML. When you’re implementing, it’s linear regression. When you’re debugging, it’s printf().”

Baron Schwartz

Herramientas de Análisis Matemático

- Optimización
  - Gradiente
  - Gradiente estocástico
  - Gradiente mini-batch
  - Optimización convexa y no-convexa
  - Mínimo/Máximo locales y absolutos

- Algebra lineal
  - Normas y regularización
  - Factorización de matrices
  - Descomposición en Valores Singulares
  - Análisis de Componentes Principales

- Pre-procesamiento
  - Transformada de Fourier

Herramientas de Probabilidad y Estadística

- Aprendizaje estadístico
  - Variables Aleatorias, Función Densidad y Función Distribución de Probabilidad
  - Vectores Aleatorios, Densidad Conjunta
  - Esperanza Condicional
  - Enfoque Bayesiano

- Simulación
  - Procesos estocásticos
  - Hidden Markov Model

- Análisis de datos
  - Estimadores Puntuales
  - Estimadores por Intervalos
  - Test de Hipótesis
  - Test de Bondad de Ajuste
  - Teorema Central del Límite

Herramientas de programación para AI

- Análisis y desarrollo de modelos de ML:
  - Scikit-learn
  - Deep Learning:
    - PyTorch | TensorFlow | Keras
- Modelos de ML en producción:
  - Numpy
- ML Distribuido:
  - Apache Spark


Python
Features
Easy to Write
Easy to Understand
Object-Oriented
Robust Standard Libraries
Supports Various Programming Paradigms
Support for Interactive Mode
Dynamically Typed and Type Checking
Databases and GUI Programming
Extensible
Portable
Scalable
Integrated
Automatic Garbage Collection
Free and Open Source

Tipos de datos básicos | Lista de tipos inmutables y mutables

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

Tipos de datos básicos | Alias

En Python, TODO ES UN OBJETO

```
#  Alias
num_1 = 10  # creates a new object "o"
print(hex(id(num_1)))  # print the memory address of the object "o" -> '0x103be3cc0'
num_2 = num_1  # creates an alias to the object "o"
print(hex(id(num_2)))  # '0x103be3cc0'

# Immutable data types
a = 10
print(hex(id(a)))  # 0x103be4800
a = 20
print(hex(id(a)))  # 0x103be5480
                   # another memory address is created because the object is immutable
a = a + 1
print(hex(id(a)))  # 0x103be54a0
```

Tipos de datos básicos | Tipos inmutables

```python
# Immutable data types in methods
def power(base, exponent):
    base = 20
    print(hex(id(base)))  # 0x103be3e00
    return base ** exponent

a = 10
exponent = 2
print(hex(id(a)))  # 0x103be3cc0
print(power(a, exponent))
print(hex(id(a)))  # 0x103be3cc0


# Everything is an object (also the numbers)
z = 201
w = 201
print(hex(id(z)))  # 0x103be54a0
print(hex(id(w)))  # 0x103be54a0
```

```python
# Tuple
t_1 = (1, ['a', 'b'])
t_2 = t_1
t_2[1].append('c')
print(t_1)  # (1, ['a', 'b', 'c'])
print(t_2)  # (1, ['a', 'b', 'c'])
# t_2[1] = ['d'] not a valid operation
```

Tipos de datos básicos | Tipos mutables

```
# Mutable types
list_1 = list([1, 2, 3, 4])
list_2 = list_1
print(list_2)  # [1, 2, 3, 4]
print(hex(id(list_1)))  # 0x105f05a00
print(hex(id(list_2)))  # 0x105f05a00
list_1.pop()
list_2.pop()
print(list_1) # [1, 2]
print(list_2) # [1, 2]
print(hex(id(list_1))) # 0x105f05a00
```

Tipos de datos básicos | Tipos mutables

```python
# List
list_1 = [1, 2, 3, 4]
print(type(list_1))  # <class 'list'>
list_1.append(5)
for elm in list_1:
    print(list_1)

# Dict
dict_1 = {
    "key_1": 1,
    "key_2": 2,
}
print(type(dict_1))  # <class 'dict'>
print(dict_1.get("key_4", None))  # None
for key, value in dict_1.items():
    print("{}: {}".format(key, value))

# Set
set_1 = set([1, 2, 3, 4, 1])  # <class 'set'>
print(type(set_1))
for item in set_1:
    print(item)
```

List y dict comprehension

```
# List comprehension
list_1 = [1, 2, 3, 4, 5, 6]
list_2 = [item for item in list_1]
print(hex(id(list_1)))  # 0x10600fa50
print(hex(id(list_2)))  # 0x1060896e0

list_3 = [item for item in list_2 if item % 2 == 0]
print(list_3)  # [2, 4, 6]
print(hex(id(list_3)))  # 0x106020be0


# Set comprehension
list_1 = ['a',  'b', 'b', 'c', 'c', 'c']
set_1 = {value for value in enumerate(list_1)}
# {'a', 'b', 'c'}


# Dict comprehension
list_1 = ['a', 'b', 'c', 'd', 'e', 'f']
dict_1 = {key: value for key, value in enumerate(list_1)}
# {0: 'a', 1: 'b', 2: 'c', 3: 'd', 4: 'e', 5: 'f'}
```

Tipos de datos básicos | Tipos mutables | Complejidad temporal

List

Operation	Average Case
Copy	O(n)
Append[1]	O(1)
Pop last	O(1)
Pop intermediate	O(k)
Insert	O(n)
Get Item	O(1)
Set Item	O(1)
Delete Item	O(n)
Iteration	O(n)
Get Slice	O(k)
Del Slice	O(n)
Set Slice	O(k+n)
Extend[1]	O(k)
Sort	O(n log n)
Multiply	O(nk)
x in s	O(n)
min(s), max(s)	O(n)
Get Length	O(1)

Dict

Operation	Average Case	Amortized Worst Case
k in d	O(1)	O(n)
Copy[2]	O(n)	O(n)
Get Item	O(1)	O(n)
Set Item[1]	O(1)	O(n)
Delete Item	O(1)	O(n)
Iteration[2]	O(n)	O(n)

Set

Operation	Average case	Worst Case
x in s	O(1)	O(n)
Union slt	O(len(s)+len(t))	
Intersection s&t	O(min(len(s), len(t))	O(len(s) * len(t))
Multiple intersection s1&s2&..&sn		(n-1)*O(l) where l is max(len(s1),..,len(sn))
Difference s-t	O(len(s))	
s.difference_update(t)	O(len(t))	
Symmetric Difference s^t	O(len(s))	O(len(s) * len(t))
s.symmetric_difference_update(t)	O(len(t))	O(len(t) * len(s))

Control de flujo

```
# Flow control with if

a = True
if a:
   print('Hello world!')

b = False
if a and b:
   print('Hello world!')

c = 1
if a == 1:
   print('1')
elif a == 2:
   print('2')
else:
   print('3')
```

```
# Flow control with for

list_1 = [1, 2, 3, 4, 5, 6]

for item in list_1:
   print(item)

for i, item in enumerate(list_1):
   print("{}: {}".format(i, item))
```

Clases

```
# Class
class Dog(object):

    def __init__(self, name):
        self.name = name

    def print_name(self):
        print(self.name)


# Object
dog_obj = Dog('Buck')
dog_obj.print_name()
```

__new__ y __init__

```
class Dog(object):

    def __new__(cls, name):
        print("__new__")
        return super(Dog, cls).__new__(cls)

    def __init__(self, name):
        print("__init__")
        self.name = name

    def print_name(self):
        print(self.name)


dog_obj = Dog('Buck')
dog_obj.print_name()

# __new__
# __init__
# Buck
```

```
class Dog(object):

    def __new__(cls, name):
        print("__new__")
        return super(Dog, cls).__new__(cls)

    def __init__(self, name):
        print("__init__")
        self.name = name

    def print_name(self):
        print(self.name)


dog_obj_1 = Dog("Buck")
dog_obj_2 = Dog("Carmy")
print(hex(id(dog_obj_1)))  # 0x102ac3090
print(hex(id(dog_obj_2)))  # 0x102ac36d0
```

```
class Dog_2(object):

    instance = None

    def __new__(cls, name):
        if Dog_2.instance is None:
            print("__new__ object created")
            Dog_2.instance = super(Dog_2, cls).__new__(cls)
            return Dog_2.instance
        else:
            return Dog_2.instance

    def __init__(self, name):
        print("__init__")
        self.name = name

    def print_name(self):
        print(self.name)


dog_obj_1 = Dog_2("Buck")
dog_obj_2 = Dog_2("Carmy")
print(hex(id(dog_obj_1)))  # 0x102aed090
print(hex(id(dog_obj_2)))  # 0x102aed090
```

Pickle

```
# save object in file
import pickl
dog_1 = Dog('Buck')
with open('dataset.pkl', 'wb') as file:
   pickle.dump(dog_1, file, protocol=pickle.HIGHEST_PROTOCOL)
```

```
# load object from file
with open('dataset.pkl', 'rb') as file:
   dog_2 = pickle.load(file)
dog_2.print_name()
```

Numpy

Ver Jupyter Notebooks en Clase

Ejercicio #1 | Operaciones matriciales

Dada una matriz en formato numpy array, donde cada fila de la matriz representa un vector matemático, se requiere computar las normas l0, l1, l2, l-infinity, según la siguientes definiciones:

l0 = número de elementos diferentes a cero en el vector

$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}.$$

$$\|\mathbf{x}\|_\infty := \max_i |x_i|.$$

Todas las operaciones debe ser vectorizadas.

Ejercicio #2 | Sorting

Data una matriz en formato numpy array, donde cada fila de la matriz representa un vector matemático, se requiere computar la norma l2 de cada vector.

Una vez obtenida la norma l2 de cada vector, se debe ordenar las normas de mayor a menor.

Finalmente, obtener la matriz original ordenada por fila según la norma l2.

Todas las operaciones debe ser vectorizadas.

Ejercicio #3 | Indexing

Construir un índice para identificadores de usuarios.
Identificadores de usuarios = users_id = [15, 12, 14, 10, 1, 2, 1]
Índice de usuarios = users_id = [0, 1, 2, 3, 4, 5, 4]
Objetivo: construir id2idx e idx2id.
Crear una clase. El índice debe construirse en el constructor.
Armar métodos de instancia “get_users_id” y “get_users_idx”

```
id2idx =  [-1   4   5  -1  -1  -1  -1  -1  -1  -1   3  -1   1  -1   2   0]
           0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
```

id2idx[15] -> 0 | id2idx[12] -> 1 | id2idx[3] -> -1

idx2id[0] -> 15 | idx2id[4] -> 1

Ejercicio #4 | Precision, recall y accuracy

En clasificación contamos con dos arreglos, la “verdad” y la “predicción”. Cada elemento de los arreglos pueden tomar dos valores, “True” (representado por 1) y “False” (representado por 0). Entonces podemos definir 4 variables:

- True Positive (TP): la verdad es 1 y la producción es 1.
- True Negative (TN): la verdad es 0 y la predicción es 0.
- False Negative (FN): la verdad es 1 y la predicción es 0.
- False Positive (FP): la verdad es 0 y la producción es 1.

Se definen las siguientes métricas:

- Precision = TP / (TP + FP)
- Recall = TP / (TP + FN)
- Accuracy = (TP + TN) / (TP + TN + FP + FN)

Suponer que se tienen 2 arreglos:
truth = [1,1,0,1,1,1,0,0,0,1]
prediction = [1,1,1,1,0,0,1,1,0,0]

Calcular las 3 métricas con Numpy y operaciones vectorizadas.

Ejercicio #5 | Average query precision

En information retrieval o search engines, en general contamos con queries “q” y para cada “q” una lista de documentos que son verdaderamente relevantes. Para evaluar search engine, es común utilizar la métrica average query precision. Dado un search engine y una lista de queries “q” para evaluación, podemos obtener los siguientes resultados:

q_id = [1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4]
predicted_rank = [0, 1, 2, 3, 0, 1, 2, 0, 1, 2, 3, 4, 0, 1, 2, 3]
truth_relevance = [T, F, T, F, T, T, T, F, F, F, F, F, T, F, F, T]

precision para q_id 1 = 2 / 4
precision para q_id 2 = 3 / 3
precision para q_id 3 = 0 / 5
precision para q_id 4 = 2 / 4

average query precision = ((2/4) + (3/3) + (0/5) + (2/4)) / 4

Calcular la métricas con Numpy y operaciones vectorizadas.

Ejercicio #6 | Distancia a centroides

Dada una nube de puntos X y centroides C, obtener la distancia entre cada vector X y los centroides utilizando operaciones vectorizadas y broadcasting en NumPy. Utilizar como referencia los siguientes valores:

- X = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
- C = [[1, 0, 0], [0, 1, 1]]

Ejercicio #7 | Enunciado

Obtener para cada fila en X, el índice de la fila en C con distancia euclídea más pequeña.
Es decir, decir para cada fila en X a qué cluster pertenece en C.
Por ejemplo, si el resultado anterior fue:

```
# [[ 3.60555128  8.36660027 13.45362405]
#  [ 2.44948974  7.54983444 12.72792206]]
```

El programa debería devolver [1, 1, 1]

Hint: utilizar np.argmin

Ejercicio #8 | Implementacion basica de K-means en Numpy

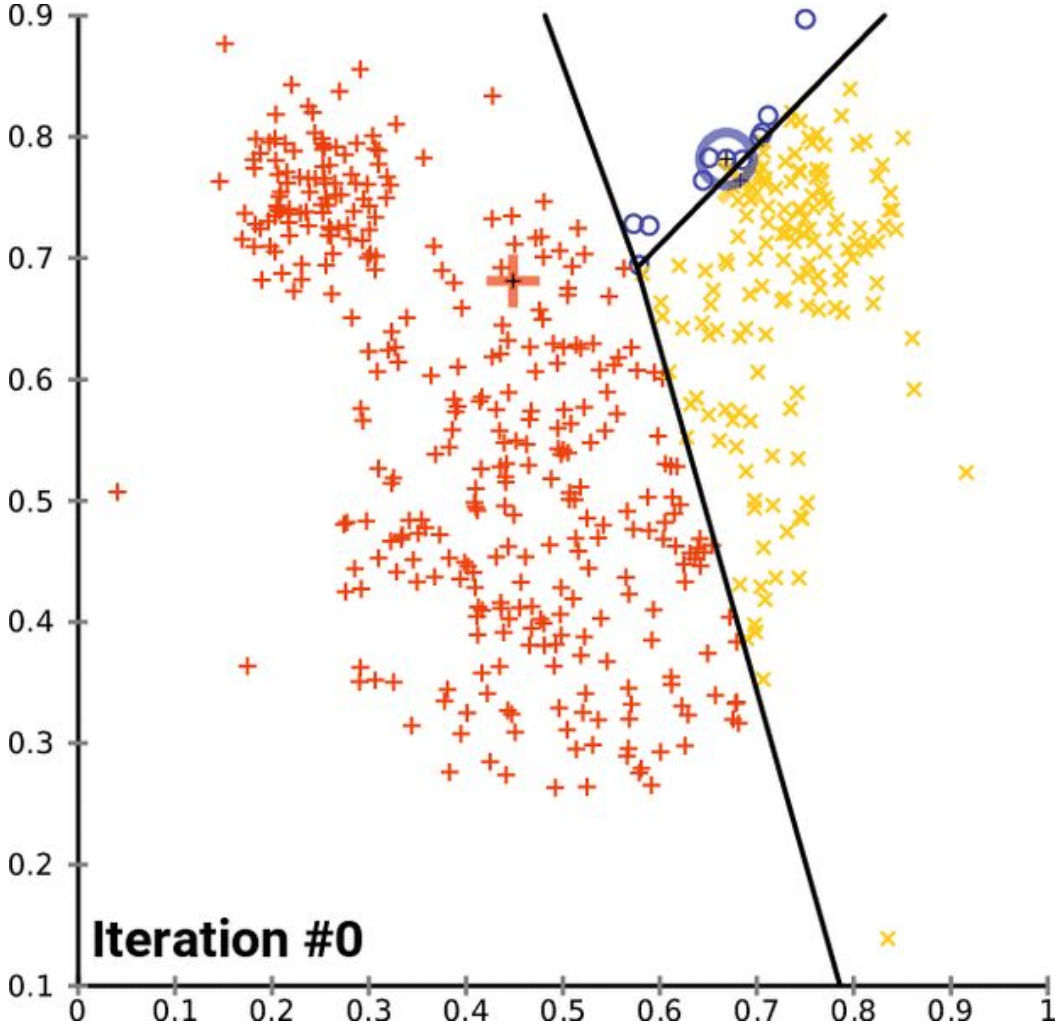
K-means es uno de los algoritmos más básicos en Machine Learning no supervisado. Es un algoritmo de clusterización, que agrupa los datos que comparten características similares. Recordemos que entendemos datos como n realizaciones del vector aleatorio X.

El algoritmo K-means funciona de la siguiente manera:

1. El usuario selecciona la cantidad de clusters a crear (n).
2. Se seleccionan n elementos aleatorios de X como posiciones iniciales del los centroides C.
3. Se calcula la distancia entre todos los puntos en X y todos los puntos en C.
4. Para cada punto en X se selecciona el centroide más cercano de C.
5. Se recalculan los centroides C a partir de usar las filas de X que pertenecen a cada centroide.
6. Se itera entre 3 y 5 una cantidad fija de veces o hasta que la posición de los centroides no cambie.

Implementar la función *def k_means(X, n)* de manera tal que al finalizar devuelva la posición de los centroides y a qué cluster pertenece cada fila de X.

Hint: para (2) utilizar funciones de np.random, para (3) y (4) usar los ejercicios anteriores, para (5) es válido utilizar un for. Iterar 10 veces entre (3) y (5).


0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Iteration #0

Ejercicio #9 | Computar todas las métricas con __call__

En problemas de machine learning, es muy común que para cada predicción que obtenemos en nuestro dataset de verificación y evaluación, almacenemos en arreglos de numpy array el resultado de dicha predicción, junto con el valor verdadero y parámetros auxiliares (como el ranking de la predicción y el query id).

Luego de obtener todas las predicciones, podemos utilizar la información almacenada en los arreglos de numpy, para calcular todas las métricas que queremos medir en nuestro sistema.

Una buena práctica para implementar esto en Python, es crear clases que hereden de una clase Metric “base” y que cada métrica implemente el método __call__.

Utilizar herencia, operador __call__, kwargs, para escribir un programa que permita calcular todas las métricas de los ejercicios anteriores mediante un for. Cuál es la ventaja de resolver el problema utilizando éstas herramientas?

Bibliografía

- The Elements of Statistical Learning | Trevor Hastie | Springer
- An Introduction to Statistical Learning | Gareth James | Springer
- Deep Learning | Ian Goodfellow | https://www.deeplearningbook.org/
- Stanford | CS229T/STATS231: Statistical Learning Theory | http://web.stanford.edu/class/cs229t/

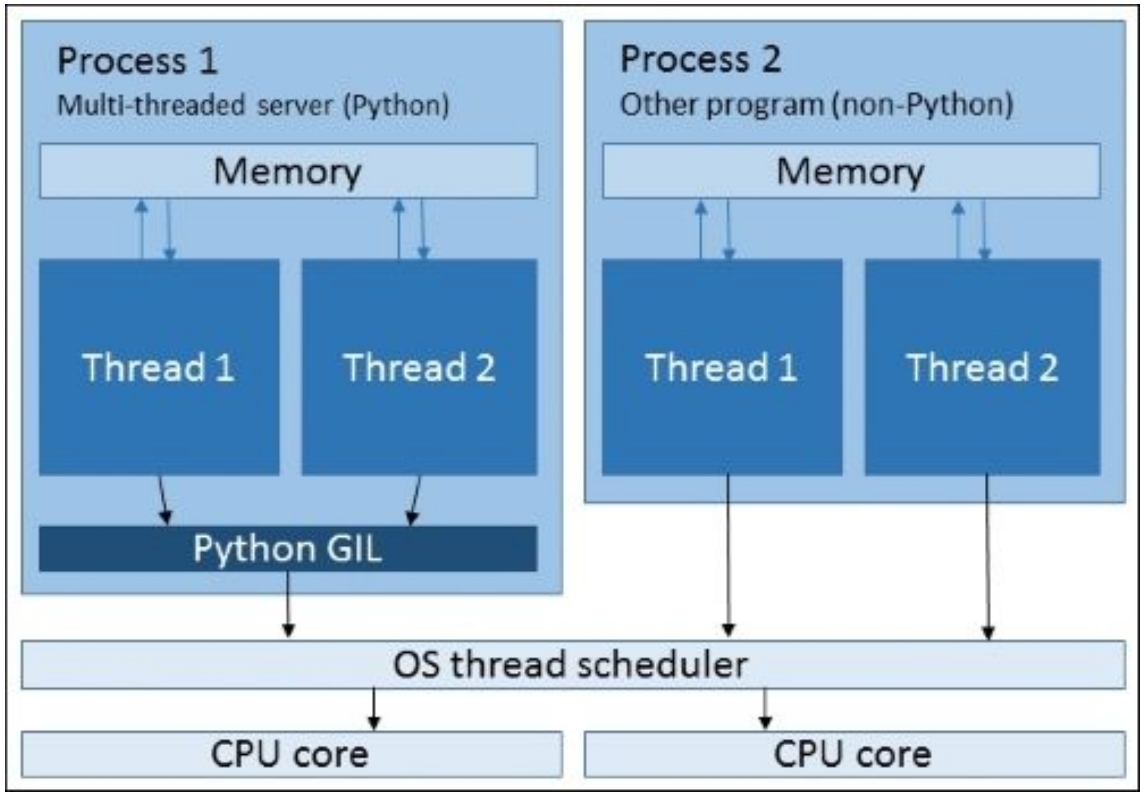
GIL, Threads y Procesos

- **GIL: Global Interpreter Lock.** Es un mutex/lock que permite que un solo thread de python mantenga el control del intérprete. Esto significa que por más que escribamos un programa en Python que sea multi-thread, si el programa es CPU-intensivo, va a comportarse como si fuese single-thread. Por ejemplo, si estamos haciendo una operación intensiva de procesamiento de imagen, el thread ejecutando dicha tarea no permitirá que otros threads se ejecuten. En programas del tipo IO-intensivo, no es un problema porque los threads están la mayoría del tiempo esperando datos (base de datos, protocolos de comunicación, etc.).

- **Multi-threading:** Python usa un único procesador, pero hay muchos threads ejecutándose concurrentemente. Debido al GIL, threading es útil en programas IO-intensivos, pero no es útil en programas CPU-intensivos. Esto va a tener un impacto interesante en problemas de Machine Learning y servidores HTTP/HTTPs como uWSGI. En general, los problemas de Machine Learning son CPU-intensivos. **¿Cómo podemos escalar un servidor con modelos de Machine Learning?**

- **Multi-processing:** Python usa múltiples procesadores del servidor, y cada procesador responde a su propio GIL. Esto soluciona en gran medida el problema de programas CPU-intensivos, pero no es simple compartir la memoria. En general, los modelos de Machine Learning están representados por grandes matrices en memoria. **¿Cómo podemos escalar la lectura y escritura de estas matrices?**

GIL, Threads y Procesos



GIL, Threads y Procesos

- **¿Cómo podemos escalar un servidor con modelos de Machine Learning?**
  - Podemos agregar más procesos.

- **¿Cómo podemos escalar la lectura de matrices para modelos de Machine Learning?**
  - En sistemas operativos basados en Linux, la creación de procesos hijos desde un proceso padre, utiliza la metodología copy-on-write. Esto significa, que la memoria entre procesos no va a ser copiada, siempre y cuando no editemos la memoria desde el thread hijo.

- **¿Cómo podemos escalar la escritura de matrices para modelos de Machine Learning?**
  - Es muy común necesitar editar las matrices en tiempo-real para actualizar y mejorar los modelos. Esta operación suele ser bastante compleja, y la mayoría de las soluciones en el mercado solucionan este problema a través de solo entrenar los modelos en producción, cada un periodo determinado (entrenar nuevo modelo -> eliminar modelo viejo -> subir modelo nuevo).
    - Alternativa 1: copiar la memoria y usar patrones pub/sub.
    - Alternativa 2: utilizar caché como Redis.
    - Alternativa 3: machine-learning distribuido.