THE BRIDGE

CAREER READINESS

ENTREVISTAS TÉCNICAS

DATA SCIENCE

Simulacro de prueba técnica

Preguntas teóricas

1. Define con tus palabras los siguientes conceptos:

a. Data Science

b. Machine Learning

c. IA

d. Big Data

2. Elige una de las siguientes preguntas y respóndela:

a. ¿Qué es un intervalo de confianza?

b. ¿Qué tipos de datos en Python conoces?

3. Explica los conceptos de entrenamiento, test y validación cruzada.

4. Explica en pseudocódigo los algoritmos KNN y KMeans. Similitudes y diferencias.

5. ¿Por qué las tapas de las alcantarillas tienen forma circular? (Pregunta real realizada por Google)

Ejercicios prácticos - Python

1. Convierte una lista en un string.

2. Convierte una lista en una tupla.

3. Convierte una lista en un set.

4. Cuenta cuántas veces aparece un elemento en una lista.

5. Resuelve el ejercicio 4 con un for dentro de una lista.

6. Genera un número aleatorio.

7. Suma todos los números de 1 a 100 en una línea

8. Usando http://www.pythontutor.com/ ejecuta el siguiente código y explica por qué no se ve el valor 3 de x nunca

```
x = [1,2,3,4,5]
for a in x:
    print(a)
    if a == 2:
        x.remove(1)
```

9. Escribe un generador en Python que devuelva "a", "b", "c" y pruébalo usando next()

Preguntas extra - Teóricas

1. ¿Qué harías primero?, ¿eliminar *outliers* y después tratar nulos o tratar nulos y después quitar *outliers*? Explica tu respuesta (orden de los pasos y qué tratamiento aplicarías)

2. Explícame el problema de *bias* vs varianza

3. Explícame el teorema central del límite.

4. Explícame la matriz de confusión

5. Diferencia entre regresión y clasificación.

6. Si quisieras predecir un target de números enteros (no continuos), ¿qué harías?

7. ¿Qué es el coeficiente *bias* en un regresor, para qué sirve y cómo se interpreta?

8. Con una regresión lineal siempre se obtiene una recta. Justifica si estás de acuerdo o no.

9. Los coeficientes de una regresión me indican la importancia de las variables. Justifica si estás de acuerdo o no.

10. ¿Qué es el *p-valor*?

11. ¿Qué tipos de join conoces? Explícalos.

12. Tienes demasiadas variables para tu algoritmo, ¿qué harías?

13. Explícame el descenso por gradiente

14. Tenemos un problema de clasificación binaria donde el 98% de los valores de la target para el entrenamiento son 0 y solo el 2% son 1. ¿Qué harías?

15. ¿Qué es la dummy trap variable?

16. ¿Qué es la regularización?

17. Explica una red neuronal convolucional.

18. ¿Qué es Flask? ¿Una API Rest? ¿El modelo cliente-servidor?

19. ¿Diferencia entre Git y GitHub? Comenta los principios de Git

20. ¿Qué es Spark? Estructuras de datos, diferencia entre el Spark Context y el Spark Session, arquitectura de Spark y qué operaciones se realizan.

21. Cuenta el proceso de un proyecto de ML desde que tienes los datos hasta el final.

Preguntas extra - SQL

En esta solución del asesinato en la ciudad SQL:
https://gist.github.com/bearloga/cfc8099223d1dace2604c8737dcbb4c3

¿qué hace?
WHERE INSTR(name, 'Annabel') > 0 AND address_street_name = 'Franklin Ave'

¿y qué hace?
SELECT person_id, COUNT(1) AS n_checkins

¿y qué hace?
AND id REGEXP '^48Z'

Ejemplo entrevista técnica:

Ejemplo de prueba de Python resuelta por un ingeniero de Airbnb:
https://interviewing.io/mocks/airbnb-python-two-sum

ENTREVISTAS TÉCNICAS DATA SCIENCE

SOLUCIONES

¡OJO! NO VALE MIRARLAS ANTES DE INTENTARLO

SOLUCIONES

Preguntas teóricas

1. Define con tus palabras los siguientes conceptos:

- **Data Science**: es la disciplina que junta informática y matemáticas para extraer información desde los datos. El procesado y modelado de los datos dependerá del caso de uso (negocio).
- **Machine Learning:** Son los modelos matemáticos donde la máquina aprende el patrón con datos anteriores. Se busca que la máquina aprenda a clasificar, a predecir un número, a hacer una predicción temporal, a tratar con imágenes, audio, vídeo... a identificar patrones, relaciones más o menos complejas, automáticamente desde los datos.
- **IA:** Inteligencia artificial, se basa en el Deep Learning, que son algoritmos que se fundamentan en las redes neuronales. La inteligencia artificial es el concepto más amplio de imitación de la inteligencia humana: lenguaje natural, imagen, sonido, vídeo... El DL extrae las características de los datos automáticamente desde el modelado.
- **Big Data** es trabajar :
  - con muchos datos
  - no estructurados o con muchos formatos
  - con respuestas muy rápidas

2. Elige una de las siguientes preguntas y respóndela:

- **¿Qué es un intervalo de confianza?**

Un intervalo de confianza te dice el nivel de seguridad que tienes, matemáticamente, en estimar que un valor está dentro de un rango.

- **¿Qué tipos de datos en Python conoces?**

str, int, float, bool, datetime, list, dictionary, set, None
Todo en Python son objetos. Los tipos de las variables son distintos, según estéis trabajando con Python directamente, con Pandas o con NumPy, por ejemplo.

Pandas dtype	Python type	NumPy type	Usage
object	str or mixed	string_, unicode_, mixed types	Text or mixed numeric and non-numeric values
int64	int	int_, int8, int16, int32, int64, uint8, uint16, uint32, uint64	Integer numbers
float64	float	float_, float16, float32, float64	Floating point numbers
bool	bool	bool_	True/False values
datetime64	datetime	datetime64[ns]	Date and time values
timedelta[ns]	NA	NA	Differences between two datetimes
category	NA	NA	Finite list of text values

3. Explica los conceptos de entrenamiento, test y validación cruzada.

- **Entrenamiento**: conjunto de datos del que se aprende minimizando la función de coste del modelo
- **Test**: conjunto de datos donde compruebo cómo funcionaría mi modelo con datos que se comportan como nuevos. NO puedo aprender de test
- **Validación cruzada**: dentro de *train* (entrenamiento), separo en distintos *k-folds* (grupos) y utilizo un *fold* para testear y el resto para entrenar. Obtengo k scorings y calculo su media para estimar el *scoring* de utilizar ese método para entrenar. Utilizo validación cruzada con varios modelos para elegir el mejor. Una vez elegido, puedo entrenar con *train* y luego testear con test. Si no quiero que haya contaminación en la validación cruzada tengo que usar pipelines y procesados para que se use el procesado desde cero en cada *fold* de la validación. Siempre se puede validar pero a veces no se puede realizar la validación cruzada. Entonces, simplemente dividimos el *train* en dos bloques sin barajar (como en las series temporales o en el *XGBoost* por el boosting).

4. Explica en pseudocódigo los algoritmos KNN y KMeans. Similitudes y diferencias.

- **K-Means**

```
Input:
    D= {t1, t2, …. Tn }   // Set of elements
    K                     // Number of desired clusters
Output:
    K                     // Set of clusters
K-Means algorithm:
   Assign initial values for m1, m2,…. mk
   repeat
     assign each item ti    to the clusters which has the closest mean;
     calculate new mean for each cluster;
   until convergence criteria is met;
```

- **KNN**

```
k-Nearest Neighbor
Classify (X, Y, x) // X: training data, Y: class labels of X, x: unknown sample
for i = 1 to m do
  Compute distance d(X_i, x)
end for
Compute set I containing indices for the k smallest distances d(X_i, x).
return majority label for {Y_i where i ∈ I}
```

- **Similitudes**: ambos son algoritmos de clustering (agrupamiento)
- **Diferencias**: KNN es supervisado (conozco el target Y), KMeans es no supervisado(no conozco el target Y)

5. ¿Por qué las tapas de las alcantarillas tienen forma circular? (Pregunta real realizada por Google)

La tapa es circular para no caer en el agujero. Al tener forma de círculo, si dejas la tapa plana paralela al agujero y la giras, no se va a caer dentro del agujero.
Si pones la tapa en vertical con respecto al agujero, tampoco se cae y si la giras en vertical, tampoco se cae. Gires como gires la tapa, tienes una esfera y nunca se va a caer dentro del agujero.
Una tapa cuadrada, si la pones en vertical, se cae dentro del agujero si la colocas en la diagonal del agujero.

Ejercicios prácticos - Python

1. Convierte una lista en un string.

```
weekdays = ['sun','mon','tue','wed','thu','fri','sat']
listAsString = ' '.join(weekdays)
print(listAsString)
```

2. Convierte una lista en una tupla.

```
weekdays = ['sun','mon','tue','wed','thu','fri','sat']
listAsTuple = tuple(weekdays)
print(listAsTuple)
```

3. Convierte una lista en un set.

```
weekdays = ['sun','mon','tue','wed','thu','fri','sat','sun','tue']
listAsSet = set(weekdays)
print(listAsSet)
```

4. Cuenta cuántas veces aparece un elemento en una lista.

```
weekdays = ['sun','mon','tue','wed','thu','fri','sun','mon','mon']
print(weekdays.count('mon'))
```

5. Resuelve el ejercicio 4 con un for dentro de una lista.

```
weekdays = ['sun','mon','tue','wed','thu','fri','sun','mon','mon']
print([[x,weekdays.count(x)] for x in set(weekdays)])
```

6. Genera un número aleatorio.

#1. random() – This command returns a floating point number, between 0 and 1.

#2. uniform(X, Y) – It returns a floating point number between the values given as X and Y.

#3. randint(X, Y) – This command returns a random integer between the values given as X and Y.

7. Suma todos los números de 1 a 100 en una línea

```
print sum(range(1,101))
```

8. Usando http://www.pythontutor.com/ ejecuta el siguiente código y explica por qué no se ve el valor 3 de x nunca

```
x = [1,2,3,4,5]
for a in x:
    print(a)
    if a == 2:
        x.remove(1)
```

Solución:

a va ir valiendo 1,2,4 y 5

Cuando a vale 2, se quita 1 elemento, x se cambia a 2,3,4,5 (se quita el primer elemento que vale 1).

El último índice del valor 2 era el índice 1. En la siguiente iteración se leerá el índice 2 y ahora el índice 2 se corresponde con el valor 4.

Seguirá imprimiendo los elementos hasta que detecte que la lista se ha acabado. Por eso no vamos a ver el valor 3, porque remove ha hecho que su índice se mueva una posición a la izquierda, pero la posición que recorre la lista sigue incrementándose 1 hasta que detecta el final de la lista.

9. Escribe un generador en Python que devuelva "a", "b", "c" y pruébalo usando next()

```
def mi_generador():
    yield "a"
    yield "b"
    yield "c"

gen = mi_generador()

print(next(gen))
print(next(gen))
print(next(gen))
```

Preguntas extra - Teóricas

1. ¿Qué harías primero?, ¿eliminar *outliers* y después tratar nulos o tratar nulos y después quitar *outliers*? Explica tu respuesta (orden de los pasos y qué tratamiento aplicarías)

Primero rellenamos los nulos con la mediana, porque la mediana no se ve afectada por los *outliers*. Después eliminamos los *outliers*, por ejemplo quedándonos con el rango intercuartílico con unos márgenes, que tiene los datos más concentrados.

Si relleno los nulos con la media estoy metiendo ruido de los *outliers* aunque luego los quite. Si empiezo quitando los *outliers* no tengo en cuenta los nulos (que sí quiero tratar) para detectar los *outliers*.

2. Explícame el problema de *bias* vs varianza

En *underfitting* tengo un *bias* bajo (el modelo no sobreajusta) pero una varianza demasiado alta (puede bajar la varianza para que el target se parezca más al resultado del modelo). En *overfitting* tengo una varianza muy baja (lo buscado por el punto anterior) pero el *bias* se me dispara (me he aprendido los datos de entrenamiento de memoria). Lo ideal es la situación intermedia, que se busca mirando el error de test, pues el error de *train* siempre disminuye cuanto más ajustamos el modelo, pero el error de test disminuye al alejarnos del *underfitting* pero se dispara al entrar en *overfitting*, pues estamos testeando con datos NO usados en el entrenamiento.

3. Explícame el teorema central del límite.

Cuando tenemos la suma de muchas variables aleatorias, la variable resultante tiene distribución gaussiana. Esto es muy importante, porque si tenemos una variable que es combinación de otras variables, o por *Feature Engineering*, acabamos teniendo variables gaussianas. Gracias al teorema central del límite aparece tan frecuentemente la campana de Gauss frente a otras distribuciones.

Preguntas extra - Teóricas

4. Explícame la matriz de confusión

		Predicción	
		Positivos	**Negativos**
Observación	**Positivos**	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

- **VP** es la cantidad de positivos que fueron clasificados correctamente como positivos por el modelo.
- **VN** es la cantidad de negativos que fueron clasificados correctamente como negativos por el modelo.
- **FN** es la cantidad de positivos que fueron clasificados incorrectamente como negativos.
- **FP** es la cantidad de negativos que fueron clasificados incorrectamente como positivos.

5. Diferencia entre regresión y clasificación.

La regresión predice un número continuo, la clasificación predice categorías.

6. Si quisieras predecir un target de números enteros (no continuos), ¿qué harías?

Una regresión y el resultado final, me quedaría con el entero más cercano.

7. ¿Qué es el coeficiente *bias* en un regresor, para qué sirve y cómo se interpreta?

Es un término de la ecuación de regresión que consigue cambiar la componente continua, subir o bajar la línea de la regresión. Es el valor de salida que se obtiene cuando todas las entradas son nulas. Muchas veces no tiene sentido que todas las variables de entrada sean cero. Por ejemplo: puede salir que cuando la edad es cero, la cantidad de hijos que se predice es -2,5. Lo que importa es que para los valores donde se va a mover la entrada del modelo, la salida sea adecuada. Sin el coeficiente *bias* siempre tendríamos una línea que pasa por el origen de coordenadas y con esa restricción no vamos a conseguir minimizar el error de predicción como teniendo el cruce con ordenadas "libre".

Preguntas extra - Teóricas

8. Con una regresión lineal siempre se obtiene una recta. Justifica si estás de acuerdo o no.

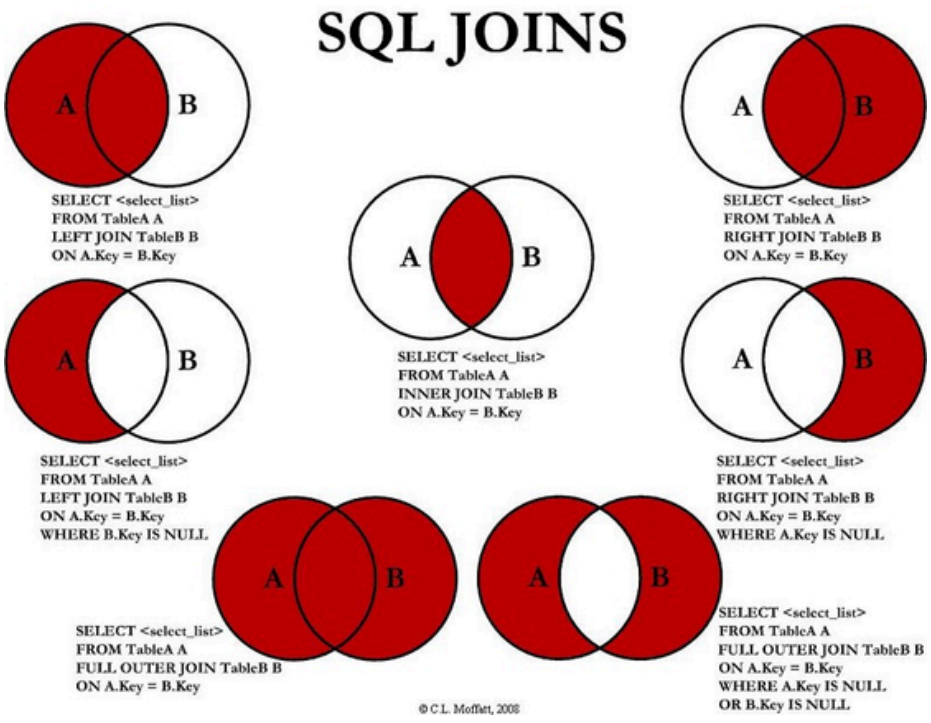
No, una regresión lineal con extensión polinómica, es lineal para calcular los coeficientes pero como obtengo y = w0 + w1 x1 + w2 x1^2 + w3 x1^3 + ... la línea de regresión no es una recta.

9. Los coeficientes de una regresión me indican la importancia de las variables. Justifica si estás de acuerdo o no.

Solo si las variables están escaladas o estandarizadas, entonces sí, porque no afecta la escala inicial y un coeficiente mayor en x2 que en x3 indica que subiendo o bajando lo mismo en x2 y en x3, el efecto en x2 hace que la predicción saliente varíe más, luego es más importante para llegar a ajustar la salida del modelo (la misma variación en la entrada produce un impacto mayor en el target).

10. ¿Qué es el *p-valor*?
Es un estadístico que me indica cómo es de probable que la hipótesis nula sea verdadera o no. Se aplica en el contraste de hipótesis para decidir si estadísticamente la H0 es muy probable o tiene la probabilidad de ser cierta muy remota.

11. ¿Qué tipos de join conoces? Explícalos.

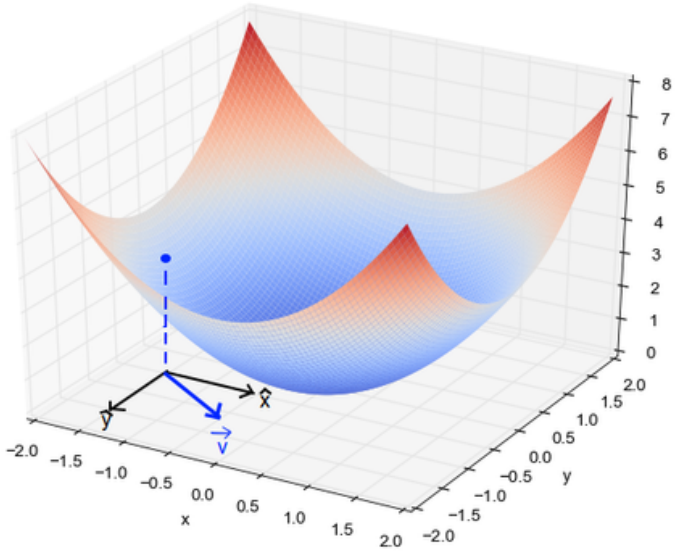


12. Tienes demasiadas variables para tu algoritmo, ¿qué harías?

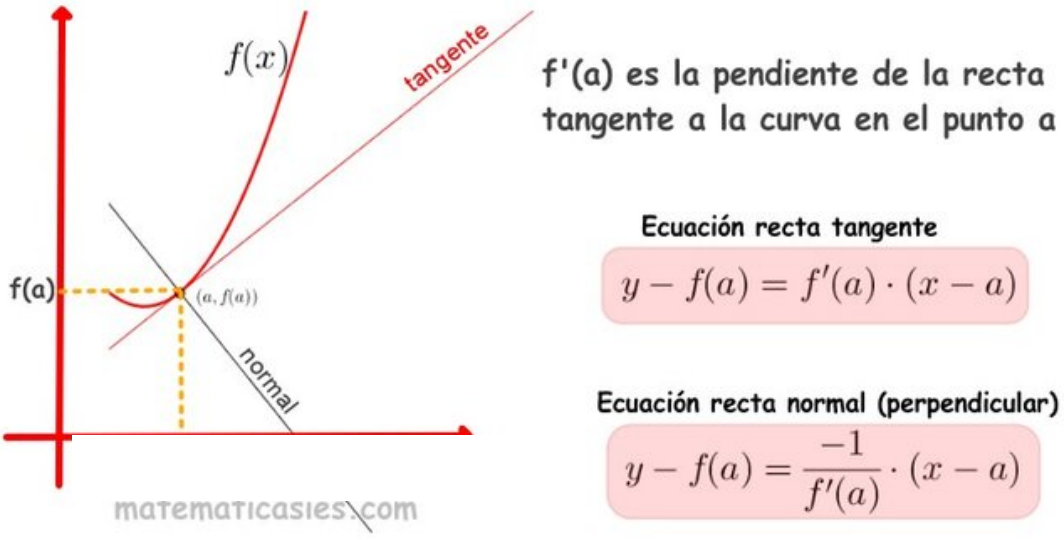
Puedo hacer un PCA para disminuir el número de variables viendo qué cantidad de información pierdo en la varianza, aunque perderé en explicabilidad. Puedo aplicar un selectKbest basado en Chi cuadrado u otro test para quedarme con las más relevantes. Puedo basarme en métodos que calcular la importancia de las variables (aunque no vaya luego a utilizar esos métodos) para conocer las más importantes, como con algoritmos de árboles y sus splittings. Puedo basarme simplemente en la varianza de las variables (cuanto más variabilidad tiene una entrada, más información contiene).

13. Explícame el descenso por gradiente

En modelos paramétricos de ML existe una función de coste (error de predicción) que depende de los parámetros del modelo.
Supongamos y = b + w1 x con su función de coste J



Lo que quiero es encontrar qué w y qué b minimiza mi error de predicción y esos valores dependen de mis datos de entrada (en este caso, de x).
Función de coste depende de *X train*, en este caso solo x1. Utilizaré todas las muestras de x1.
¿Y cómo consigo encontrar el punto más bajo de la función de coste? Mi algoritmo empieza en un par (w,b) y calcula el error (la altura hasta el plano curvado). En la siguiente iteración va a mover la coordenada (w,b) un poquito (el paso) y vuelve a mirar la altura. Me interesa moverme en la dirección donde baje más rápidamente. Yo no sé a priori dónde está el mínimo del plano curvado (si lo supiera, no tendría que hacer todo esto). Pero SÍ puedo saber justo en el punto (w,b) donde me encuentro, en esa altura, con derivadas parciales, hacia dónde me tengo que mover para bajar con la mayor pendiente posible (las derivadas nos dan pendientes en puntos de curvas y son derivadas parciales porque quiero derivar parcialmente respecto a w y parcialmente respecto a b).



La derivada de f(x) en x=a tiene un valor. Ese valor significa cuánto vale la pendiente de esa recta tangente. Si quiero descender lo más rápido desde f(x=a) tendré que moverme por la recta que tiene de punto x=a y de pendiente MENOS la derivada (si cojo el positivo, asciendo lo más rápido posible).

Llevado a la función de coste con 2 parámetros de antes:

Parto de un (w,b)
Calculo con el descenso por gradiente (el gradiente son derivadas parciales) la dirección de bajada más rápida desde ese punto, pero yo SOLO puedo saber, dado un (w,b), la altura (el coste) y a esa altura una "brújula" de hacia dónde mover (w,b). Imaginemos que me dice que mueva w –> w = w + 0.01 (paso) * 0.02 (movimiento proyectado en w) y que mueva b –> b = b + 0.01 (paso) * (-0.07) (movimiento proyectado en b). Así en cada iteración nos vamos desplazando por la superficie hacia su mínimo.

14. Tenemos un problema de clasificación binaria donde el 98% de los valores de la target para el entrenamiento son 0 y solo el 2% son 1. ¿Qué harías?

Balancear las clases. Si tengo muchos registros, elimino registros con target 0 para tener 50%-50% de ceros y unos. Si no tengo muchos registros, creo sintéticamente unos artificiales basándome en los unos que ya existen.

15. ¿Qué es la dummy trap variable?

La trampa de la variable ficticia ocurre cuando dos o más variables ficticias creadas mediante la codificación *one-hot* están altamente correlacionadas (multicolineales). Esto significa que una variable puede predecirse a partir de las otras, lo que dificulta interpretar los coeficientes predichos en modelos de regresión. En otras palabras, el efecto individual de las variables ficticias en el modelo de predicción no puede interpretarse bien debido a la multicolinealidad.

Al utilizar el método de codificación *one-hot*, se crea una nueva variable ficticia para cada variable categórica para representar la presencia (1) o ausencia (0) de la variable categórica. Por ejemplo, si la especie de árbol es una variable categórica compuesta por los valores pino o roble, entonces la especie de árbol puede representarse como una variable ficticia convirtiendo cada variable en un vector *one-hot*. Esto significa que se obtiene una columna separada para cada categoría, donde la primera columna representa si el árbol es pino y la segunda columna representa si el árbol es roble. Cada columna contendrá un 0 o un 1 si el árbol en cuestión es de la especie de la columna. Estas dos columnas son multicolineales ya que si un árbol es pino, entonces sabemos que no es roble y viceversa.

+ info

16. ¿Qué es la regularización?

La regularización es un conjunto de técnicas utilizadas en el aprendizaje automático y la estadística para prevenir el sobreajuste (*overfitting*) en modelos predictivos. El sobreajuste ocurre cuando un modelo se ajusta demasiado bien a los datos de entrenamiento, capturando el ruido en los datos en lugar de las verdaderas relaciones subyacentes. Esto puede resultar en un rendimiento deficiente del modelo cuando se aplica a nuevos datos. La regularización introduce una penalización adicional en la función de pérdida del modelo para limitar la complejidad del modelo, lo que ayuda a evitar el sobreajuste. Hay varias técnicas de regularización comunes, como la regularización L1 (*Lasso*), la regularización L2 (*Ridge*) y la *Elastic Net*, cada una de las cuales penaliza de manera diferente los coeficientes del modelo. Estas técnicas de regularización funcionan al agregar términos de penalización a la función de pérdida durante el entrenamiento del modelo, lo que fomenta coeficientes más pequeños y, por lo tanto, modelos más simples y generalizables.

En resumen, la regularización es una estrategia clave para mejorar la capacidad de generalización de los modelos predictivos al evitar el sobreajuste y mantener un equilibrio entre el sesgo y la varianza del modelo.

+ info

17. Explica una red neuronal convolucional.

Una red neuronal convolucional (CNN por sus siglas en inglés, Convolutional Neural Network) es un tipo especializado de red neuronal artificial diseñada específicamente para procesar datos que tienen una estructura de cuadrícula, como imágenes o datos de series temporales. Las CNN han demostrado ser muy efectivas en tareas de visión por computadora, como reconocimiento de imágenes, clasificación de objetos, segmentación semántica, entre otras.

La arquitectura de una CNN se basa en capas de convolución, que son capaces de extraer automáticamente características relevantes de los datos de entrada a través de filtros o kernels. Estos filtros se aplican de manera local a regiones de la entrada, lo que permite que la red capture patrones espaciales en la información, como bordes, texturas o formas.

Además de las capas de convolución, las CNN suelen incluir capas de submuestreo (o pooling) para reducir la dimensionalidad de las características extraídas, así como capas de activación no lineales, como ReLU (Rectified Linear Unit), para introducir no linealidades en el modelo.

Las CNN suelen seguir una arquitectura en cascada, donde las capas convolucionales y de submuestreo se alternan varias veces antes de conectarlas a capas completamente conectadas, que son responsables de realizar la clasificación o regresión final. Esta estructura jerárquica permite que las CNN aprendan representaciones jerárquicas de los datos de entrada, desde características simples en las primeras capas hasta características más abstractas y complejas en las capas más profundas.

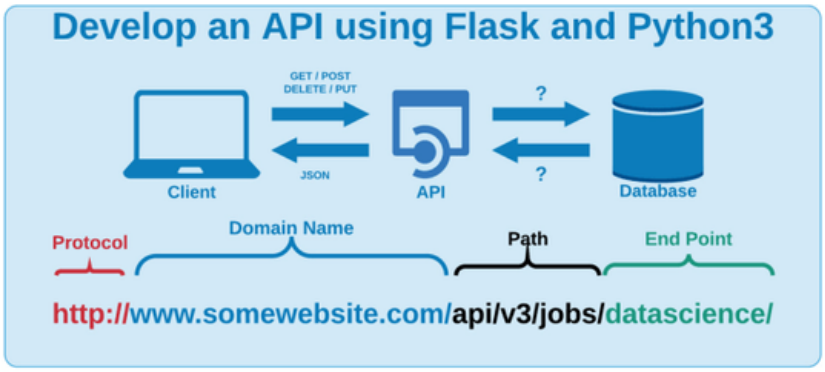
+ info

18. ¿Qué es Flask? ¿Una API Rest? ¿El modelo cliente-servidor?

Flask es un popular framework de desarrollo web en Python que se utiliza para crear aplicaciones web de manera rápida y sencilla. Es conocido por ser ligero, flexible y fácil de aprender, lo que lo hace ideal para desarrolladores que desean construir aplicaciones web simples o complejas.

Aunque Flask es comúnmente utilizado para desarrollar aplicaciones web, incluyendo aplicaciones CRUD (Create, Read, Update, Delete), no está limitado únicamente a este propósito. Flask se puede utilizar para construir diversos tipos de aplicaciones, incluyendo APIs RESTful.

Una API REST (Representational State Transfer) es una interfaz de programación de aplicaciones que sigue los principios de REST, una arquitectura de red que se basa en estándares web como HTTP, URL y JSON. Las API RESTful permiten a los clientes interactuar con el servidor mediante solicitudes HTTP, como GET, POST, PUT y DELETE, para realizar operaciones en recursos específicos.
Flask se puede utilizar para construir y servir una API RESTful, donde las rutas URL se mapean a funciones Python que manejan las solicitudes HTTP y devuelven respuestas en formato JSON u otros formatos de datos. Esto permite la comunicación eficiente entre aplicaciones cliente y servidor a través de la web.
El modelo cliente-servidor es un paradigma de arquitectura de software en el que las aplicaciones están divididas en dos partes distintas: el cliente, que es la interfaz de usuario o la aplicación que solicita servicios, y el servidor, que es la aplicación que provee esos servicios. Flask, al ser un framework de desarrollo web, facilita la implementación del modelo cliente-servidor al permitir la creación rápida y eficiente de la parte del servidor de una aplicación web, mientras que el cliente puede ser cualquier aplicación que consuma los servicios proporcionados por el servidor, ya sea un navegador web, una aplicación móvil u otra aplicación basada en la web.



19. ¿Diferencia entre Git y GitHub? Comenta los principios de Git
Git y GitHub están estrechamente relacionados pero tienen funciones distintas:
Git:
1. Sistema de Control de Versiones Distribuido: Git es un sistema de control de versiones distribuido que permite a los desarrolladores rastrear los cambios en el código fuente a lo largo del tiempo. Cada desarrollador tiene una copia completa del repositorio en su máquina local, lo que permite trabajar de forma independiente y fusionar cambios de manera eficiente.
2. Funcionamiento Local: Git funciona principalmente en tu computadora local. Puedes iniciar un repositorio Git en cualquier carpeta de tu máquina, lo que te permite rastrear los cambios en tus proyectos sin necesidad de una conexión a Internet.

3. Gestión de Versiones y Ramas: Git te permite gestionar las versiones de tu código y trabajar con ramas (branches) para desarrollar nuevas características o arreglar problemas sin afectar la rama principal del proyecto.

GitHub:

1. Plataforma de Alojamiento de Repositorios: GitHub es una plataforma en línea que aloja repositorios Git de forma remota. Permite a los desarrolladores colaborar en proyectos, realizar seguimiento de problemas (issues), gestionar solicitudes de extracción (pull requests), y mucho más.

2. Colaboración y Comunidad: GitHub fomenta la colaboración entre desarrolladores al proporcionar herramientas para revisar y comentar el código de otros, así como para gestionar proyectos de software de manera conjunta.

3. Gestión de Proyectos: GitHub ofrece funcionalidades adicionales como tableros (boards) de proyecto, seguimiento de errores, wikis, y acciones (actions) automatizadas para facilitar la gestión y el desarrollo de proyectos de software.

Principios de Git:

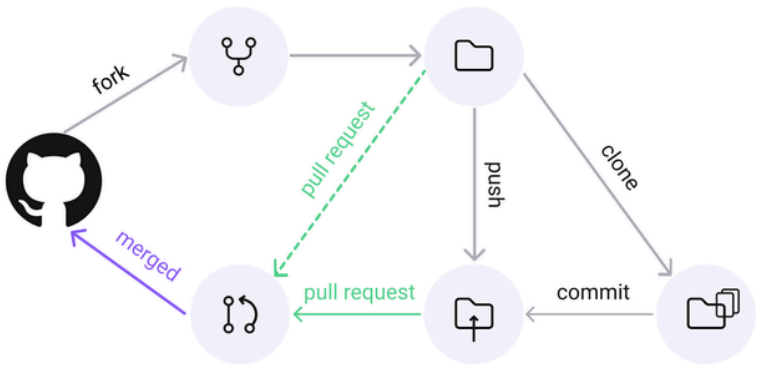
1. Sistema de Control de Versiones: Git es un sistema que rastrea los cambios en los archivos de un proyecto a lo largo del tiempo, lo que permite volver a versiones anteriores del código si es necesario.

2. Distribuido: Cada desarrollador tiene una copia completa del repositorio en su máquina local, lo que facilita el trabajo colaborativo y la gestión de versiones.

3. Ramas (Branches): Git permite trabajar con ramas independientes del código principal para desarrollar nuevas características o solucionar problemas sin interferir con el código estable.

4. Historial de Cambios: Git mantiene un historial completo de todos los cambios realizados en el código, lo que facilita la identificación de quién hizo un cambio específico y cuándo se realizó.

5. Rápido y Eficiente: Git es rápido y eficiente en el manejo de grandes proyectos y grandes cantidades de datos, lo que lo hace ideal para proyectos de cualquier tamaño.



20. ¿Qué es Spark? Estructuras de datos, diferencia entre el Spark Context y el Spark Session, arquitectura de Spark y qué operaciones se realizan.

Spark es un potente framework de procesamiento distribuido y análisis de datos diseñado para manejar grandes volúmenes de datos de manera eficiente en clústeres de computadoras. Fue desarrollado originalmente en la Universidad de California, Berkeley, y es mantenido por Apache Software Foundation en el proyecto Apache Spark.

Estructuras de datos en Spark:

1. **Resilient Distributed Dataset (RDD):** Es la estructura de datos fundamental en Spark. Es una colección inmutable y distribuida de objetos que pueden ser procesados en paralelo.
2. **DataFrame:** Es una estructura de datos tabular similar a una tabla en una base de datos relacional. Proporciona una abstracción más alta que RDD y es optimizada para el procesamiento de datos estructurados.
3. **Dataset:** Es una API de alto nivel que proporciona una interfaz tipo DataFrame, pero con la ventaja de poder aprovechar el sistema de tipos de datos estático de Java o Scala.

Diferencia entre Spark Context y Spark Session:

- **Spark Context:** Era el punto de entrada principal en versiones anteriores de Spark, utilizado para conectarse a un clúster Spark y crear RDDs. Sin embargo, con la introducción de DataFrames y Datasets, el Spark Context ha sido reemplazado por la Spark Session en las versiones más recientes de Spark.
- **Spark Session:** Es el punto de entrada principal en las aplicaciones de Spark a partir de la versión 2.0. Unifica los puntos de entrada anteriores (SparkContext, SQLContext, HiveContext) y proporciona una API unificada para trabajar con RDDs, DataFrames y Datasets.

Arquitectura de Spark:

- **Driver**: Es el proceso principal que coordina las operaciones en un programa de Spark y mantiene la información sobre la aplicación.
- Executor: Son procesos que se ejecutan en los nodos del clúster y realizan tareas específicas asignadas por el driver.
- Cluster Manager: Es el componente responsable de asignar recursos (CPU, memoria) a las aplicaciones Spark y administrar los nodos del clúster.
- Spark Core: Es el corazón de Spark, que proporciona la funcionalidad básica para el procesamiento distribuido, incluyendo la programación de tareas, la gestión de memoria y la coordinación entre los nodos del clúster.

Operaciones realizadas en Spark:

- **Transformaciones:** Operaciones que transforman un RDD, DataFrame o Dataset en otro, como map, filter, groupBy, etc.
- Acciones: Operaciones que devuelven un resultado al programa principal, como collect, count, saveAsTextFile, etc.
- **Persistencia de datos:** Guardar los resultados intermedios en memoria o en disco para reutilizarlos en futuras operaciones, utilizando cache o persist.
- **Procesamiento en batch y en tiempo real:** Spark es capaz de procesar datos tanto en modo batch como en tiempo real utilizando componentes como Spark Streaming, Structured Streaming, y más.

21. Cuenta el proceso de un proyecto de ML desde que tienes los datos hasta el final.

1. Obtención de datos:
   - Identifica las fuentes de datos relevantes para tu problema.
   - Recopila los datos necesarios, ya sea desde bases de datos, archivos, APIs u otras fuentes.
   - Realiza tareas de limpieza de datos para eliminar valores atípicos, datos faltantes o inconsistencias.
2. Exploración y análisis de datos:
   - Explora y visualiza los datos para comprender su estructura y características.
   - Identifica patrones, tendencias y relaciones entre las variables.
   - Realiza análisis estadísticos y gráficos descriptivos para obtener información sobre los datos.
3. Preprocesamiento de datos:
   - Realiza tareas de preprocesamiento como normalización, estandarización, codificación de variables categóricas, etc.
   - Divide los datos en conjuntos de entrenamiento, validación y prueba para evaluar el rendimiento del modelo de manera adecuada.
4. Selección y entrenamiento del modelo:
   - Selecciona el algoritmo de ML adecuado para tu problema, teniendo en cuenta factores como el tipo de datos, el tamaño del conjunto de datos y el objetivo del proyecto.
   - Entrena el modelo utilizando el conjunto de entrenamiento, ajustando los hiperparámetros según sea necesario.
   - Utiliza técnicas como la validación cruzada para evaluar el rendimiento del modelo y evitar el sobreajuste.

5. Evaluación del modelo:
    - Evalúa el modelo utilizando el conjunto de validación y/o prueba para medir su rendimiento en datos no vistos.
    - Utiliza métricas de evaluación adecuadas para tu problema, como precisión, recall, F1-score, etc.
    - Realiza análisis de errores para identificar áreas de mejora y posibles ajustes en el modelo.
6. Ajuste del modelo (si es necesario):
    - Realiza ajustes en el modelo según los resultados de la evaluación, como cambiar los hiperparámetros, probar diferentes algoritmos o realizar ingeniería de características adicional.
7. Despliegue del modelo:
    - Implementa el modelo en producción, integrándolo en sistemas existentes si es necesario.
    - Realiza pruebas adicionales para asegurarte de que el modelo funcione correctamente en un entorno de producción.
8. Monitorización y mantenimiento:
    - Monitoriza el rendimiento del modelo en producción y realiza ajustes si es necesario.
    - Mantén el modelo actualizado con nuevos datos y cambiantes condiciones del mundo real.
    - Realiza mantenimiento regular para asegurarte de que el modelo sigue siendo relevante y preciso.

ENTREVISTAS TÉCNICAS DATA SCIENCE

RECURSOS Y MATERIAL EXTRA

SIGUE PRACTICANDO Y APRENDIENDO

ROADMAP

- Python
- SQL
- AI and Data Scientist
- Data Analyst
- MLOps
- Promp Engineering

InterviewBit

- Python
- Data Analyst
- Data Science
- ETL Testing
- Deep Learning
- NLP
- Algorithm
- Data Modelling
- NumPY
- Statistics
- Linear Regression
- Pandas

freeCodeCamp

- Google Interview Question Guide: Delete Reoccurring Characters with Python
- SQL
- Mock Data Science Job Interview
- Data Science Interview Questions for Beginners
- Common SQL Interview Questions: Your Database Cheat Sheet
- How to Prepare for Data Analyst Job Interviews

Mock Interview

- Python Interview with an Airbnb engineer