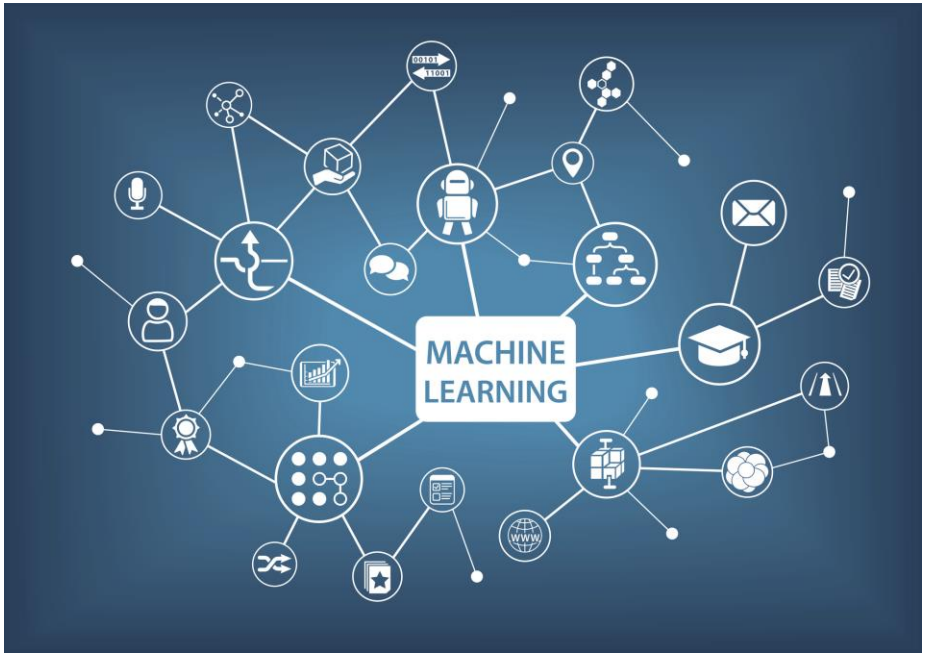
Módulo 4: Aprendizaje Automático

Temario de la clase 1
9 de agosto del 2024
Modelos lineales para regresión

- Introducción al Machine Learning
- Tipos de aprendizaje
- Modelos lineales
- Función de error
- Método del descenso del gradiente
- Generalización
- Problema del sobreajuste y subajuste

Machine Learning

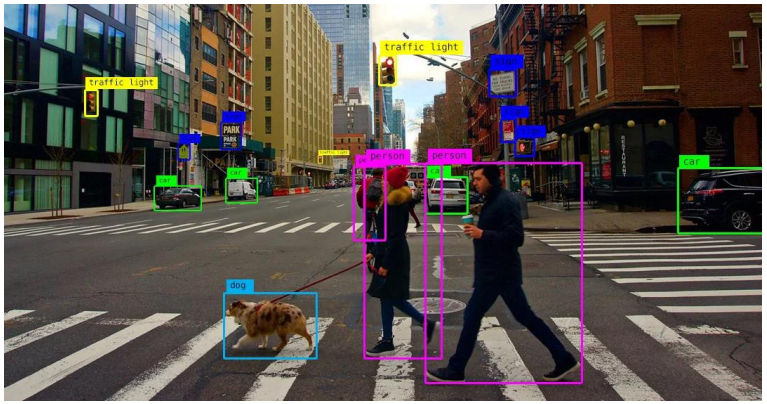
El **Machine Learning** (aprendizaje automático) es una rama de la inteligencia artificial que permite a las máquinas **aprender de los datos** y **mejorar su desempeño** en tareas específicas sin ser explícitamente programadas para ello. En lugar de seguir reglas rígidas establecidas por programadores, un sistema de machine learning identifica patrones en grandes volúmenes de datos y utiliza esos patrones para hacer predicciones o tomar decisiones.



Machine Learning

Aplicaciones del ML que *no* pueden realizarse con métodos clásicos de programación.

- **Procesamiento de lenguaje natural** (Word, Gmail, ChatGPT)
- **Reconocimiento de voz**
- **Visión por computadora** (reconocimiento de imágenes, detección de objetos)
- **Sistemas de recomendación** (Google, Netflix, Spotify, Youtube, Mercado Libre, publicidades)



Machine Learning

Los algoritmos más ampliamente usados son:

- **Regresión lineal**
- Regresión logística
- Máquinas de soporte vectori
- Clasificador Bayesiano
- Árboles de decisión
- Bosques aleatorios
- Redes neuronales
- Análisis discriminante lineal
- K-vecinos más próximos

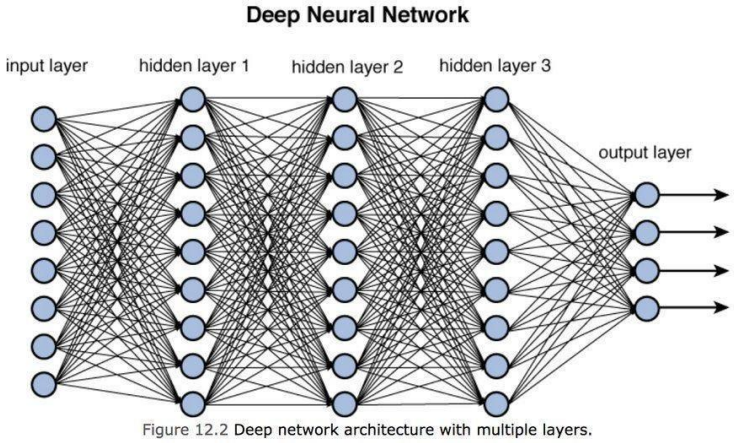
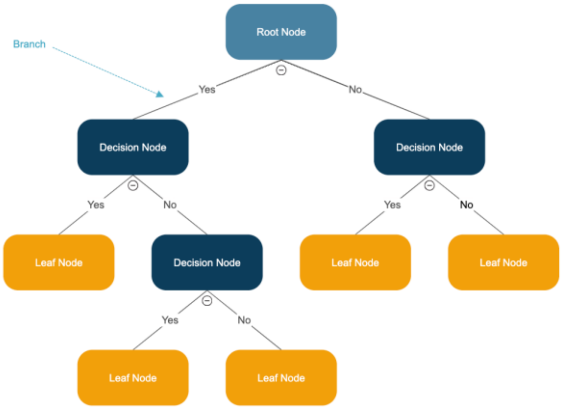


Figure 12.2 Deep network architecture with multiple layers.



Machine Learning

En general, un problema de ML puede considerarse como perteneciente a dos clases: problema de regresión o problema de clasificación.

- **Regresión**: el resultado es un valor numérico que se busca estimar o predecir. Ejemplo: predecir la cantidad de lluvia en una región el mes que viene, el tiempo que va a permanecer un paciente en el hospital, determinar el valor de una casa, analizar la variación del tamaño de una población o el precio de una acción en la bolsa.

- **Clasificación**: el resultado indica la probabilidad de que una determinada muestra pertenezca a un conjunto de categorías o agrupaciones. Si solamente se tienen dos categorías, la clasificación se denomina binaria. Ejemplo: identificar un correo como deseado o como spam, identificar los caracteres en una imagen.

Machine Learning

No importa con que tipo de algoritmo o problema estemos tratando, los componentes básicos son siempre los mismos:

- Los **datos** a partir de los cuales aprendemos.
- Un **modelo** que se encarga de transformar los datos.
- Una **función de error** que cuantifica que tan bien (o que tan mal) el modelo trabaja.
- Un **algoritmo** que ajusta los parámetros del modelo para disminuir el error y mejorar la performance. Esto se denomina entrenamiento del modelo.

- Una vez que el modelo está entrenado, es capaz de dar resultados o predicciones a entradas de datos que antes no ha visto.

Paradigmas del Machine Learning

Aprendizaje Supervisado:

- En este enfoque, el modelo es entrenado con un conjunto de datos etiquetados, donde las respuestas correctas están ya disponibles. El objetivo es que el modelo aprenda a mapear las entradas a las salidas correctas. Ejemplos comunes incluyen la clasificación de imágenes y la regresión.

Aprendizaje No Supervisado:

- Aquí, el modelo trabaja con datos no etiquetados. El objetivo es descubrir la estructura subyacente o los patrones en los datos. Ejemplos incluyen el clustering (agrupación) y la reducción de dimensionalidad.

Paradigmas del Machine Learning

Aprendizaje por Refuerzo:

- En este enfoque, un agente aprende a tomar decisiones a través de la interacción con un entorno. El agente recibe recompensas o castigos en función de sus acciones y ajusta su estrategia para maximizar las recompensas a lo largo del tiempo. Este enfoque es común en aplicaciones como juegos o robótica.

Componentes del aprendizaje supervisado

Entrenamiento

- Durante el entrenamiento, el modelo de machine learning procesa los datos de entrada y ajusta sus parámetros para minimizar el error o maximizar la precisión. Este proceso se realiza utilizando una función de pérdida que mide qué tan bien está funcionando el modelo.

Validación

- Una vez que el modelo ha sido entrenado, se evalúa para determinar su desempeño. Esto se hace generalmente utilizando un conjunto de datos separado que no fue utilizado durante el entrenamiento. Las métricas comunes de evaluación incluyen la precisión, la exactitud, el recall, entre otros.

Componentes del aprendizaje supervisado

Entonces, nuestros **datos disponibles** o *dataset* se van a dividir en dos grupos:

- Los datos usados para el entrenamiento del modelo.
- Los datos utilizados para la validación del modelo.

Es necesario evaluar el modelo teniendo en cuenta su performace sobre los datos que no ha visto durante el entrenamiento para determinar si realmente ha aprendido.

La forma de dividir y aprovechar los datos disponibles es una cuestión a tener en cuenta a la hora de entrenar un modelo.

Regresión lineal

- Es la herramienta mas simple para resolver problemas de regresión. Establece que el resultado del modelo se obtiene a partir de la suma ponderada de las entradas o ***inputs***.
- Ecuación general de un modelo de regresión lineal con *d* entradas:

$$\hat{y} = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b$$

$$\hat{y} = \sum_{i=1}^{d} w_i x_i + b = \boldsymbol{w}^T \boldsymbol{x} + b$$

- $\hat{\boldsymbol{y}}$ representa la salida estimada por el modelo, el ***output***.
- Aquí ***w*** y ***x*** representan vectores columna que contienen los inputs y los pesos correspondientes a cada ejemplo.
- El parámetro **b** representa el bias o valor que tomaría la estimación cuando todas las inputs son cero.

Regresión lineal

- Si tenemos un conjunto de datos con n observaciones o ejemplos del fenómeno, podemos armar una matriz **X** de dimensión nxd para representar los datos.

$$\hat{\mathbf{y}} = \boldsymbol{X}\boldsymbol{w} + b$$

- La matriz X vendría a representar nuestro dataset, el cual importaremos a nuestro entorno de trabajo.

Regresión lineal: ejemplo

- Supongamos que queremos calcular el valor de una casa y tenemos a disposición distintos datos como ser:

    - El tamaño o área de la casa en metros cuadrados.
    - El número de habitaciones.
    - El número de baños.
    - La edad de la construcción.
    - Un coeficiente de la calidad de la construcción.

- Entonces la variable dependiente precio será una función del conjunto de variables independientes nombradas anteriormente.
- A las variables independientes se las suele llamar **características** o *features*.
- A la variable dependiente se la suele llamar **objetivo** o *target*.

Regresión lineal: ejemplo

- Supongamos que queremos calcular el valor de una casa y tenemos a disposición distintos datos como ser:

  - El tamaño o área de la casa en metros cuadrados.
  - El número de habitaciones.
  - El número de baños.
  - La edad de la construcción.
  - Un coeficiente de la calidad de la construcción.

$$\hat{y} = w_1 x_{area} + w_2 x_{hab} + w_2 x_{baño} + w_2 x_{const} + w_d x_{coef} + b$$

$$\hat{y} = \sum_{i=1}^{d} w_i x_i + b = \boldsymbol{w}^T \boldsymbol{x} + b$$

Regresión lineal: solución analítica

- Si tomamos un modelo simple de regresión lineal, es posible encontrar soluciones analíticas que satisfacen la condición de que el error total sea nulo o mínimo para un conjunto de parámetros.
- Para un conjunto de datos con ejemplos y features, podemos construir la ecuación del modelo.
- Donde si las filas de la matriz X son linealmente independientes, la solución analítica para los parámetros w viene dada por:

$$\boldsymbol{w} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

- Sin embargo, los modelos de ML como las redes neuronales divergen rápidamente de estos modelos simples, y por lo tanto las soluciones analíticas son imposibles de encontrar en la enorme mayoría de los casos.

Discusión sobre la naturaleza del ML

- Sin embargo, el paradigma del ML establece que no es necesario encontrar soluciones **exactas** para un determinado problema, sino que es deseable encontrar soluciones **prácticas** que se puedan aplicar y cuya aproximación sea suficiente en la mayoría de los casos.

- Además, como veremos mas adelante, entrenar un modelo para que tenga un error mínimo respecto de los datos que recibe durante el proceso de optimización puede llegar a ser **muy indeseable**.

- También debemos tener en cuenta que los datos con los que contamos no son exactos, sino que contienen **errores** de medición, de observación, sistemáticos o sesgados. Por lo tanto es innecesario buscar generar un modelo que devuelva dichos valores *exactamente*. Asumimos que el error (también llamado ruido) tiene una distribución normal de media cero.

Función de pérdida o loss function

- En ML, los algoritmos actualizan los valores de los parámetros del modelo teniendo como referencia si el mismo comete un error mayor o menor al deseado. Por eso es importante cuantificar con precisión el error cometido por el modelo.
- La **función de pérdida** compara la salida del modelo contra el valor real que debería tomar, el cual es conocido por nosotros por medio del set de datos.
- No existe una única función de pérdida, y la misma depende de si el problema es de regresión o clasificación.
- Otras denominaciones para la función de pérdida son **función de coste** o **función de error**.

Función de pérdida o loss function

- En un problema de regresión, donde tenemos el valor real de la salida y, junto con el valor estimado por el modelo ŷ, podemos calcular el error como el cuadrado de la diferencia.
- La magnitud del error dependerá de los parámetros del modelo.
- La expresión del error cuadrático para la i-esima muestra del dataset será:

$$l^{(i)}(w,b) = \frac{1}{2}(y^{(i)} - \hat{y}^{(i)})^2$$

- Podemos calcular el **error medio cuadrático** (MSE, *mean squared error*) cometido por el modelo sobre todos los puntos del dataset como:

$$L(w,b) = \frac{1}{n}\sum_{i=1}^{n} l^{(i)}(w,b) = \frac{1}{2n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$$

Función de pérdida o loss function

- Puesto que queremos que nuestro modelo tenga el menor error posible, la función de perdida es un parámetro que se busca **minimizar**.
- Ya que el error es causado por los parámetros introducidos en el modelo, la optimización del valor de la función de pérdida se realiza modificando dichos parámetros, lo cual se puede hacer de manera iterativa.
- En cada paso de la iteración, el valor actualizado de un parámetro se calcula teniendo en cuenta el valor de la función de pérdida en el paso anterior.

Descenso del gradiente

- En gran parte, el ML se encarga de optimizar el ajuste de un modelo a un conjunto de datos.

- El **método del descenso del gradiente** (*gradient descent*) es un proceso iterativo que busca aproximarse paso a paso hacia una solución óptima.

- La idea principal del método consiste en reducir iterativamente el error, actualizando los parámetros en la dirección que progresivamente disminuya la magnitud de la función de pérdida.

Descenso del gradiente

- Iniciamos el método asignando valores aleatorios a los parámetros del modelo. Intuitivamente, esto nos daría un error importante en la performance inicial del mismo.
- Para tener una idea de como el valor del error puede disminuir (o aumentar) debemos tener el cuenta la derivada de la función de coste.
- Se introduce también el concepto de **learning rate (η)** o tasa de aprendizaje, que regula el tamaño de las actualizaciones.
- Los parámetros se actualizan en base al tamaño del gradiente (derivada) de la función de coste respecto a cada parámetro y el valor del learning rate.

Descenso del gradiente

- En un modelo con parámetros **w** y **b** a optimizar podemos calcular el nuevo valor del parámetro en función del gradiente de la función de pérdida:

$$w_{i+1} = w_i - \eta \nabla_w (w, b)$$

$$b_{i+1} = b_i - \eta \nabla_b (w, b)$$

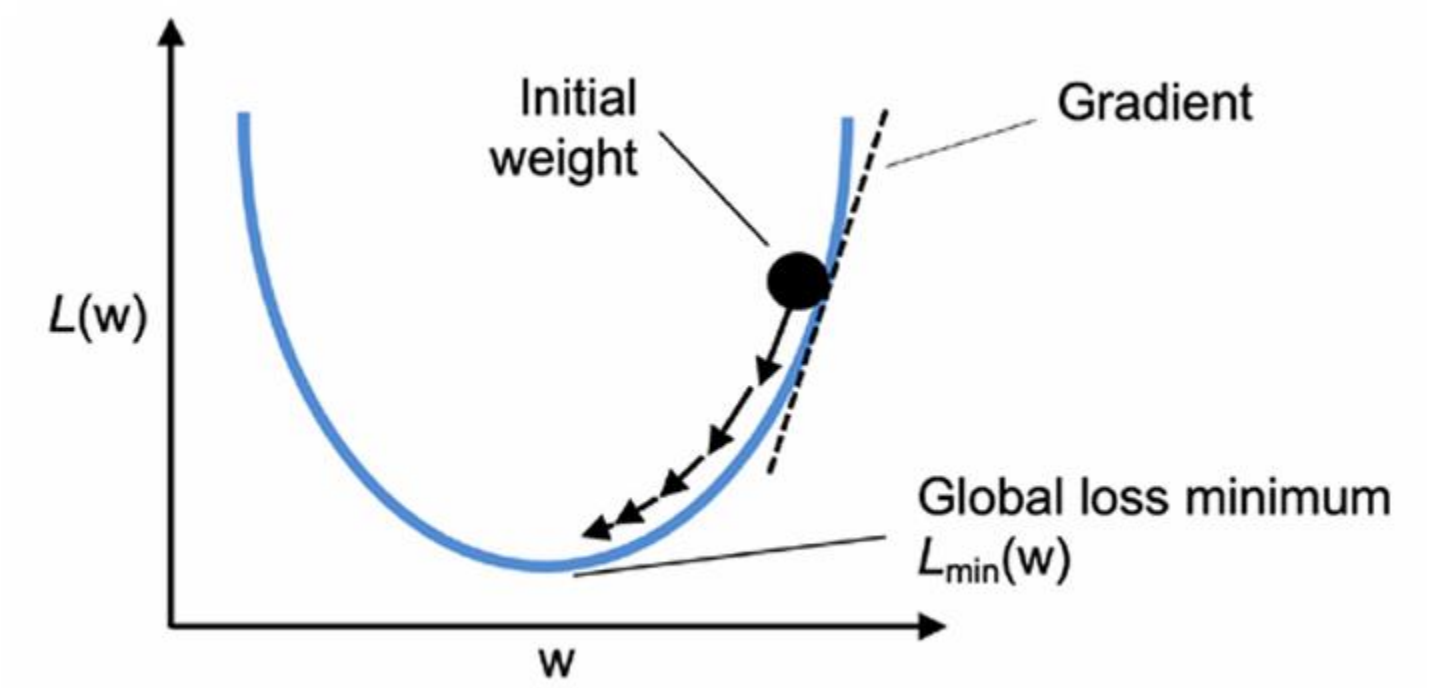
- O lo que es equivalente:

$$w_{i+1} = w_i - \eta \frac{\partial}{\partial w} L(w, b)$$

$$b_{i+1} = b_i - \eta \frac{\partial}{\partial b} L(w, b)$$

Descenso del gradiente

- El proceso se itera un determinado número de veces o hasta que se cumpla alguna condición.



Descenso del gradiente: ejemplo para 1 parámetro

- Supongamos que tenemos un modelo simplificado que sigue la ecuación:

$$\hat{y} = ax$$

- Vamos a optimizar el parámetro α utilizando descenso del gradiente.
- Puesto que:

$$L(a) = \frac{1}{2}(y - ax)^2$$

- Tenemos que:

$$\frac{\partial L(a)}{\partial a} = ax^2 - xdy = x(ax - y) = \nabla_a(a)$$

Descenso del gradiente: ejemplo para 1 parámetro

- Si tenemos el siguiente set de datos:

x	1.7	2.3	5	0.2	4.3
y	4.4	5.7	12.3	0.7	10.8

- Calculamos el gradiente de la función de coste para los datos asumiendo un valor inicial α = 0, y un learning rate η = 0.09.

$$\nabla_a(a) = \sum_{i=1}^{5} x(ax - y)$$

$$a_{i+1} = a_i - \eta \nabla_a(a)$$

$$L(a) = \frac{1}{10} \sum_{i=1}^{5} (y^{(i)} - ax^{(i)})^2$$

Descenso del gradiente: ejemplo para 1 parámetro

x	1.7	2.3	5	0.2	4.3
y	4.4	5.7	12.3	0.7	10.8

- Calculamos el error inicial por asumir α = 0.

$$L(a=0) = \left[\begin{matrix}(4.4-0)^2+(5.7-0)^2+(12.3-0)^2 \\ +(0.7-0)^2+(10.8-0)^2)\end{matrix}\right] * \frac{1}{10}$$

$$= 32.027$$

Descenso del gradiente: ejemplo para 1 parámetro

x	1.7	2.3	5	0.2	4.3
y	4.4	5.7	12.3	0.7	10.8

- El step inicial es:

$$Step = \frac{1}{5} * 0.09 \left[\begin{matrix} 1.7(0-4.4) + 2.3(0-5.7) + 5(0-12.3) \\ +0.2(0-0.7) + 4.3(0-10.8 \end{matrix} \right]$$
$$= -2.05872$$

Con lo que el nuevo valor de α es:

$$a = 0 - (-2.05872) = 2.05872$$

El cual genera un error de:

$$L(a = 2.05872) = 0.96429$$

Descenso del gradiente: ejemplo para 1 parámetro

x	1.7	2.3	5	0.2	4.3
y	4.4	5.7	12.3	0.7	10.8

- El nuevo step es:

$$0.018\begin{bmatrix} 1.7(2.05872 * 1.7 - 4.4) + 2.3(2.05872 * 2.3 - 5.7) \\ +5(2.05872 * 5 - 12.3) \\ +0.2(2.05872 * 0.2 - 0.7) + 4.3(2.05872 * 4.3 - 10.8 \end{bmatrix}$$
$$= -0.355417$$

Con lo que el nuevo valor de α es:

$$a = 2.05872 - (-0.355417) = 2.41413$$

El cual genera un error de:

$$L(a = 2.41413) = 0.03848$$

Descenso del gradiente: ejemplo para 1 parámetro

x	1.7	2.3	5	0.2	4.3
y	4.4	5.7	12.3	0.7	10.8

- Realizando dos iteraciones más llegamos a los siguientes resultados:

	a	L(a)	Step
Cond. Inicial	0	32.027	--
Iteración 1	2.05872	0.96429	-2.05872
Iteración 2	2.41413	0.03848	-0.35541
Iteración 3	2.47549	0.01088	-0.06135
Iteración 4	2.48608	0.01006	-0.0105

Descenso del gradiente: ejemplo para 1 parámetro

- Los valores del dataset se habían calculado con un valor de α igual a 2.5 con una componente de ruido superpuesta. Con lo que vemos que el método se aproxima rápidamente al valor real.

	a	L(a)	Step
Cond. Inicial	0	32.027	--
Iteración 1	2.05872	0.96429	-2.05872
Iteración 2	2.41413	0.03848	-0.35541
Iteración 3	2.47549	0.01088	-0.06135
Iteración 4	2.48608	0.01006	-0.0105

Descenso del gradiente: ejemplo para 2 parámetros

- Si ahora consideramos un modelo lineal con dos parámetros a optimizar (la pendiente y la ordenada al origen) tenemos:

$$\hat{y} = ax + b$$

$$L(a,b) = \frac{1}{2}(y - ax - b)^2$$

$$\frac{\partial L(a,b)}{\partial a} = x(ax + b - y)$$

$$\frac{\partial L(a,b)}{\partial b} = ax + b - y$$

$$\nabla(a,b) = \left(\frac{\partial L(a,b)}{\partial a}, \frac{\partial L(a,b)}{\partial b}\right)$$

Descenso del gradiente: ejemplo para 2 parámetros

- En este caso los pasos a seguir son similares a los descriptos anteriormente:

1. Asignar valores iniciales a los parámetros.
2. Obtener el error inicial.
3. Calcular el gradiente de la función de error respecto a cada parámetro: tendremos tantas derivadas como parámetros a optimizar.
4. Con el valor del gradiente y el learning rate, calcular el step para cada parámetro.
5. Con el valor actual del parámetro y el step, calcular el nuevo valor del parámetro.
6. Calcular el nuevo error e iterar.

Descenso del gradiente estocástico (SGD)

- El algoritmo descrito anteriormente tiene la desventaja de que si el set de entrenamiento es grande, por ejemplo, si tiene 1.000.000 de puntos, entonces en cada iteración se deberán computar 1.000.000 de derivadas para poder hacer una actualización.
- El número de cálculos también aumenta si en cada paso queremos actualizar más de un parámetro a la vez. Por ejemplo si queremos actualizar 10 parámetros deberíamos calcular 10.000.000 de derivadas.
- Entonces, cuanto el dataset es muy grande esto no es computacionalmente eficiente, sobre todo porque puede haber redundancia en los datos.
- El entrenamiento del modelo se hace muy lentamente.

Descenso del gradiente estocástico (SGD)

- Por ello, en lugar de utilizar todo el dataset para hacer una actualización de uno o más parámetros, el método del descenso del gradiente se modifica para realizar una actualización luego de haber tenido en cuenta una sola muestra tomada al azar.
- En este caso, el método se denomina **descenso del gradiente estocástico** (*stochastic gradient descent*).
- Entonces el gradiente se "aproxima" teniendo en cuenta una sola muestra, lo que es una estrategia efectiva.
- Es más rápido ya que actualizamos los parámetros calculando el gradiente sobre un solo ejemplo del dataset.

Descenso del gradiente estocástico (SGD)

- Una desventaja de la aplicación de este método es que los procesadores realizan más rápidamente la acción de multiplicar y sumar números que la de mover elementos a distintas partes de la memoria.
- Esto hace que analizar varias muestras al mismo tiempo pueda ser más computacionalmente eficiente que analizar una muestra a la vez.

Descenso del gradiente estocástico por mini bloques

- Por lo tanto, una solución intermedia se encuentra mediante el análisis del gradiente utilizando mini bloques de observaciones (*mini batch stochastic gradient descent*).
- El dataset completo se divide en un determinado número de bloques que contienen el mismo número de inputs y la actualización de los parámetros se realiza en base al gradiente calculado por cada bloque.
- El tamaño del bloque puede variar dependiendo del tamaño del dataset o de la memoria, aunque se suelen tomar potencias de 2.

Librerías

- Cualquier algoritmo de ML requiere la necesidad de realizar una gran cantidad de cálculos numéricos de forma reiterativa, muchas veces con grandes cantidades de datos. Por ej: multiplicación de vectores y matrices.
- El uso de algoritmos eficientes reduce el costo computacional de dichos cálculos, por lo que es necesario que los mismos estén optimizados en cada etapa de la aplicación del mismo.
- Existen librerías especializadas para ML que se pueden utilizar en Python. Entre ellas tenemos PyTorch (2016) y TensorFlow (2015).

Parámetros e hiperparámetros

- Los valores que componen el modelo en si y que son calculados **por el algoritmo** mediante procesos de optimización se denominan parámetros.
- Los valores que son definidos **por el usuario** y que van a determinar el funcionamiento del algoritmo se denominan hiperparámetros.

- **Parámetros:**
  - ❖ Pesos
  - ❖ Bias
  - ❖ Gradientes

- **Hiperparámetros:**
  - ❖ Tipo de modelo y estructura
  - ❖ Función de error
  - ❖ Tasa de aprendizaje
  - ❖ Número de iteraciones
  - ❖ Optimizador

Generalización

- El objetivo es descubrir patrones que generalicen.
- Trabajar con muestras finitas, implica el riesgo de que ajustemos nuestros datos de entrenamiento y descubramos que no pudimos descubrir un patrón generalizable.
- Existe un fenómeno de ajustar más a los datos de entrenamiento que a la distribución subyacente: **overfitting.**

Error de entrenamiento y de generalización

- En el aprendizaje supervisado, los datos de entrenamiento y los datos de prueba se extraen independientemente de distribuciones idénticas: *supuesto IID*.

- El supuesto dice que los datos de entrenamiento tomados de la distribución P(X,Y) deberían ser capaces de predecir los datos de prueba generados por una distribución Q(X,Y) diferente.

Error de entrenamiento y de generalización

- El error de entrenamiento es una estadística calculada sobre el conjunto de datos de entrenamiento.
- El error de generalización es una expectativa tomada con respecto a la distribución subyacente.
- Se estima el error de generalización aplicando nuestro modelo a un conjunto de pruebas independiente constituido por una selección aleatoria de ejemplos *X'* y los targets *y'* que no se incluyeron en nuestro conjunto de entrenamiento.
- Esto es, aplicar la misma fórmula que se usó para calcular el error de entrenamiento empírico pero a un conjunto de prueba *X'*, e *y'* desconocidos.

Error de entrenamiento y de generalización

- Cuando evaluamos nuestro clasificador en el conjunto de prueba, estamos trabajando con un clasificador fijo (no depende de la muestra del conjunto de prueba) y, por lo tanto, estimar su error es simplemente el problema de la estimación de la media.

- El modelo con el que terminamos depende explícitamente de la selección del conjunto de entrenamiento y, por lo tanto, *el error de entrenamiento será en general una estimación sesgada del error verdadero de la población subyacente*.

- La cuestión central de la generalización es entonces cuándo deberíamos esperar que el error de entrenamiento esté cerca del error de población (y, por tanto, del error de generalización).

Complejidad del modelo

- Con modelos más complejos y/o menos ejemplos, el error de entrenamiento disminuye pero la brecha de generalización crece.

- Rango de valores que pueden tomar los parámetros.
- Un modelo cuyos parámetros puedan tomar valores arbitrarios sería **más complejo**.

- Modelo es capaz de ajustar etiquetas arbitrarias:

❖ Un error de entrenamiento bajo no implica necesariamente un error de generalización bajo.

❖ Tampoco implica necesariamente un alto error de generalización.

Sobreajuste y subajuste

- **Caso 1**: el error de entrenamiento y el error de validación son sustanciales pero hay una pequeña brecha entre ellos.

- Si el modelo no puede reducir el error de entrenamiento, entonces el modelo es demasiado simple para capturar el patrón que se está tratando de modelar.

- La brecha entre ambos errores es pequeña, podemos suponer que un modelo más complejo ajustaría mejor.

- Este fenómeno se conoce como **subajuste** o *underfitting*.

Sobreajuste y subajuste

- **Caso 2:** el error de entrenamiento es significativamente menor que el error de validación.
- Este fenómeno se denomina **sobreajuste** o *overfitting*.
- No siempre significa una desventaja o un problema.
- Si el error de entrenamiento es cero, entonces la brecha de generalización es exactamente igual al error de generalización y sólo podemos avanzar reduciendo la brecha.

Ajuste de curva polinomial

- Intentamos encontrar el polinomio de grado d, para estimar el target $\hat{y}$.

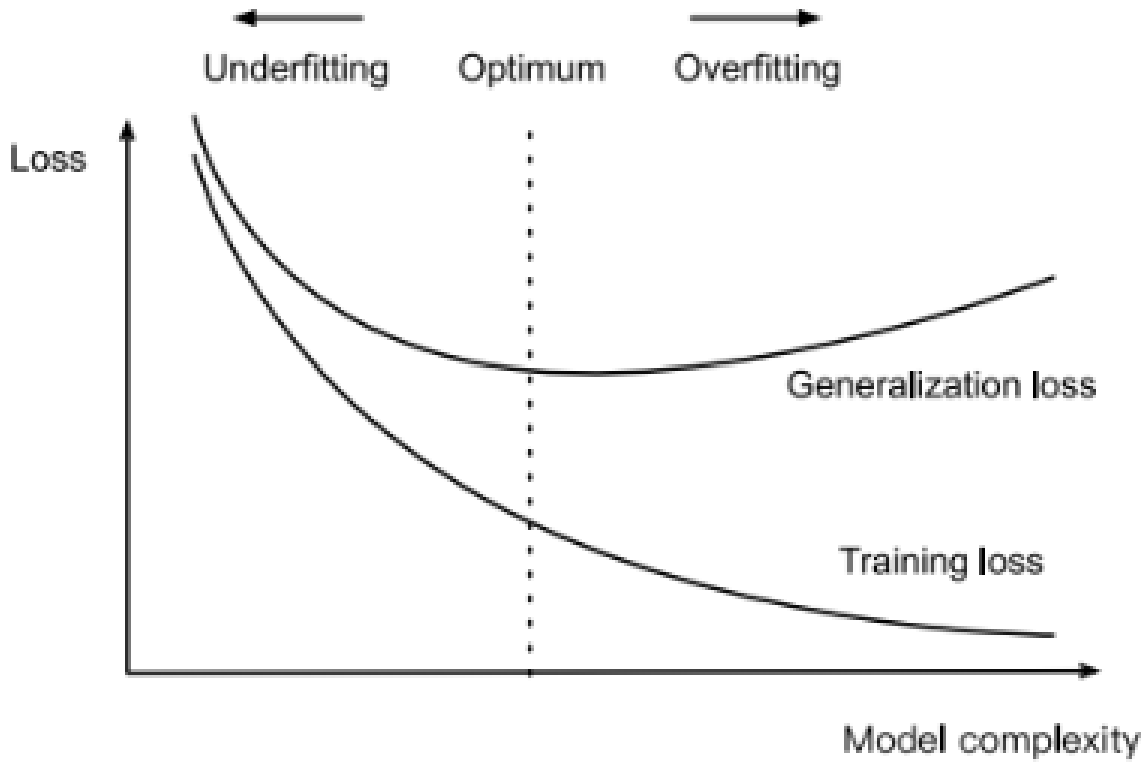
$$\hat{y} = \sum_{i=0}^{d} w_i x^i$$

- Es un problema de regresión lineal donde las features están dadas por las potencias de x, los pesos del modelo están dados por wi y el sesgo está dado por w0 ya que x0 = 1 para todo x.

- Es problema de regresión lineal, entonces se usa el error al cuadrado como función de pérdida.

Ajuste de curva polinomial

- Relación entre el grado del polinomio (complejidad del modelo) y el underfitting y overfitting:



Selección del modelo

1. Selección del Modelo
- Evaluar múltiples modelos con diferentes arquitecturas, objetivos, características y tasas de aprendizaje.
  - La selección del modelo se realiza tras evaluar varias configuraciones.

2. Conjunto de Pruebas

3. Uso Adecuado de los Datos de Prueba

4. División de Datos

FIN DE LA CLASE 1