¿Qué es Machine Learning ?

- - Machine Learning es una rama de la inteligencia artificial.
- - Las máquinas aprenden a reconocer patrones y tomar decisiones a partir de los datos en lugar de instrucciones explícitas.
- - modelos que puedan aprender de forma autónoma y mejorar su desempeño a medida que se les proporcionan más datos.

EXPERIENCE, TASK , PERFORMANCE

- - Experience E": Aprender de datos que se proporcionan
- - Task T": Tarea específica que se está intentando realizar.
    - reconocimiento de voz, detección de fraude o recomendación de productos.
- - performance P": Medición del rendimiento
- ** El modelo debe mejorar su capacidad para realizar la tarea T a medida que recibe más datos o experiencia E. Y esto se mide en términos de mejora en la medida de rendimiento P.

MACHINE LEARNING FLAVORS

- - Supervised Learning -> Las Respuestas correctas son dadas
- - Unsupervised Learning -> Las Respuestas correctas NO son dadas
- - Reinforcement Learning -> Maximizar una recompensa en un ambiente controlado

TRIBUS DEL MACHINE LEARNING

- - Symbolists: Símbolos, reglas y lógica para representar conocimiento -> Reglas y Árboles de decisión
- - Connectionists: Redes neuronales artificiales para aprender patrones a partir de los datos. -> Redes Neuronales
- - Bayesians: Estadísticas Bayesianas para modelar la incertidumbre y actualizar continuamente la probabilidad de los resultados a medida que se reciben nuevos datos. -> Naive Bayes y Markov
- - Evolutionaries: Esta tribu se inspira en la selección natural, evolucionan con el tiempo. -> Genetic Programs
- - Analogizers: Razonamiento basado en casos y la analogía para hacer predicciones a partir de datos similares en el pasado. -> Support Vectors

INGREDIENTES DEL MACHINE LEARNING

- Representación (Regres Lineales,Redes Neuronales,Regre Logística)
- - Evaluación (Función de Costo) ¿Cómo se elige TETA?
- - Optimizacion (Descenso de Gradiente)

REGULARIZACION (OVERFITTING)

- El overfitting ocurre cuando un modelo de aprendizaje automático se vuelve demasiado complejo y se ajusta demasiado a los datos de entrenamiento, perdiendo capacidad de generalización
    - Polinomios con grados muy altos
    - Lo optimo es tener low training error, low test error
        - Underfit -> high training error, high test error
        - Overfit -> low training error, high test error
    - DECISION BOUNDARY complejas causan overfitting
- Como no caemos ?
    - Usamos datos relevantes
    - Hacemos data augmentation
    - Seleccionamos los features que vamos a usar
    - Paramos antes en las redes neuronales
    - Regularización

- Regularización -> reducir el número de features aplicando una penalidad a parámetros con coeficientes altos.
    - Se agrega a las funciones de costo $+ \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$; donde lambda 0 causa overfitting y lambda grande underfitting

DESCENSO DE GRADIENTE (OPTIMIZACION)

- Es un algoritmo de optimización
- Aleatoriamente inicializa los parámetros TETA
- Cambia los parámetros hasta llegar al mínimo (Se repite hasta Converger)

$\theta_j := \theta_j - \alpha \frac{\delta}{\delta \theta_j} J(\theta_0, \theta_1)$; α Learning rate

- Es susceptible a mínimos locales ! SADDLE POINTS
- Tres tipos :
  - Batch GD -> Cada paso que se da se usan todos los training examples -> larga ejecución, no podemos guardar en memoria bases de datos grandes $\sum_{i=1}^{m = Training\ Examples}$
  - Stochastic GD -> Solo se usa un training sample por paso $\frac{1}{2}(H_\theta(X^{(i)}) - y^{(i)})^2$, fácil de guardar en memoria
  - Mini-batch GD -> Combinación de los dos anteriores $\sum_{i=1}^{m' < m}$

α LEARNING RATE

- Muy pequeño hace que la convergencia sea lenta
- Muy grande causa divergencia
- Valores que podemos usar = 0.0001, 0.001, 0.01, 0.1, 1

¿Por qué es malo Normal Equation a comparación de GD?

- Los dos debemos elegir ALPHA
- GD necesita de muchas iteraciones pero trabaja bien con N features grandes
- Normal Equation necesita computar $(X^T X)^{-1}$ lo hace lento para N features
- Para n> 10 000 usamos GD

- CONVERGENCIA MÁS RÁPIDA Y MEJOR
  - momentum -> problemas con mínimos locales como GD
  - AdaGrad -> llega a el minimo global
  - RMSProp -> llega a el minomo global
  - Adam -> llega al mínimo global =

REGRESIÓN LINEAL UNA VARIABLE (1 Feature)

- OJO TETA ES EL FEATURE PARA CUALQUIER REPRESENTACIÓN

1. Representación (Regresión Lineal simple)

$H_\theta = \theta_0 + \theta_1 X$, $H_\theta = \theta^T x$

2. Evaluacion (Mean Squared Error)

$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m = Training\ Examples} (H_\theta(X^{(i)}) - y^{(i)})^2$; (x(i), y(i)) → ith training example

3. Optimización (Descenso de Gradiente -> Regresión lineal es convexa no hay SADDLE POINTS)

$min(J(\theta_0, \theta_1))$

REGRESIÓN LINEAL MULTIVARIABLE

- Múltiples Features (Varias Entradas)
- $X^{(i)} \rightarrow input\ de\ features\ de\ la\ columna\ i \rightarrow es\ un\ vector\ OJO!$
  - $X_j^{(i)} \rightarrow valor\ del\ feature\ J\ del\ input\ de\ features\ de\ la\ columna\ i \rightarrow extraer\ del\ vector\ X^{(i)}\ el\ feature\ J$

1. Representación (Regresion Lineal multiple)

$H_\theta = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \ldots + \theta_n X_n$; $\theta_0\ es\ el\ BIAS$

2. Evaluacion (Mean Squared Error)

$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m=Training\ Examples} (H_\theta(X^{(i)}) - y^{(i)})^2$ -> MISMA QUE LA DE REGRESIÓN LINEAL SIMPLE

3. Optimización (Descenso de Gradiente)
   - $min(J(\theta_0,....,\theta_N))$ PERO LA REPETIMOS PARA N VECES θ_N

   - FEATURE SCALING -> TENEMOS QUE TENER EN CUENTA QUE TODAS LAS VARIABLES SE ENCUENTREN EN ESCALAS SIMILARES
     - Podemos usar:
       - Min - Max normalization $X_{norm} = \frac{X - min(x)}{max(x) - min(x)}$
       - Z-Score $X_{stand} = \frac{X - mean(x)}{std(x)}$; es la mejor que hay, se basa en la media y la desviación pero aun así podemos tener outliers(valores alejados)

REGRESIÓN LOGÍSTICA (CLASIFICAR)
- No predice un valor continuo pero más bien a qué clase pertenece nuestra muestra
- Puede ser o :
  - Es una clasificación Binaria 0 o 1
  - Clasificación multiclase 0,1,2,3… OJO!! PARA K FEATURES NECESITAMOS K REGRESIONES LOGÍSTICAS -> ELEGIMOS LA QUE MAXIMIZA LA PROBABILIDAD
- ¿Cuál es la diferencia con Regre Linear?
  - Linear es para cualquier número real
  - Logística es una probabilidad entre 0 y 1

1. Representacion (Sigmoid o Logistica)

$H_\theta = g(\theta^T x) = \frac{1}{1+e^{-z}} \rightarrow Llamada\ función\ sigmoide\ o\ logística$

- Esta función sigmoide nos entrega un DECISION BOUNDARY (puede ser lineal o no)
  - Son fronteras que separan las regiones de clasificación -> Diferentes THRESHOLDS CAMBIAN LAS PREDICCIONES (0.5,0,2…) -> pueden ser ajustadas dependiendo del Performance P

2. Evaluacion (Binary Cross Entropy BCE o Log Loss)

$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m=Training\ Examples} [y^{(i)} log((H_\theta(X^{(i)})) + (1 - y^{(i)}) log(1 - H_\theta(X^{(i)}))]$

- $\frac{1}{m} \sum_{i=1}^{m=Training\ Examples}$ -> Media
- Primer log es Cross Entropy $P(y = 1|x;\theta)$
- Segundo log es Cross Entropy $P(y = 0|x;\theta)$

3. Optimización (Descenso de Gradiente)

$min(J(\theta))$) -> muy similar a la regresión lineal pero la clave es que $H_\theta(x^i)$ ahora es la función sigmoide vs $H_\theta = \theta^T x$ de la lineal

REDES NEURONALES
- En bases de datos dimensionales con relaciones no lineares, la regresión logística causa overfitting
- Aparecen kas redes neuronales desde 1940, el primer problema que se encuentran en cómo modelar XOR, llegan las SVMs y aparece el segundo invierno, entonces se soluciona con las deep nets
- 3 tipos de inteligencia artificial -> narrow ANI , genera AGI 2040 l, superintelligence ASI 2060
- Neurona humana -> entrada (Dendritas) -> computación (núcleo) -> salida (Axón) - son llamadas unidades logísticas
  - Entrada -> pesos de los parámetros -> neurona (sumatoria de combinaciones lineales y función de activación) -> salida (H_θ(x))
  - Funciones de Activación

    - Sigmoid , Softmax(transforma vector de números a vector de probabilidades -> salida), Rectifier Linear Unit (ReLU), Linear(identidad), Leaky ReLU

- Las hidden layers transforman los problemas en problemas lineales, más layers mas features complejas
- One-hot encoding -> tenemos que transformar los labels a vectores
- FUNCIÓN DE COSTO -> CATEGORICAL CROSS ENTROPY LOSS A BCE SE LE SUMA $\sum_{K=1}^{K}$ DESPUÉS DE LA PRIMER SUMATORIA
- REGULARIZACIÓN -> SE LE SUMA A COSTO $+ \frac{\lambda}{2m} \sum_{I=1}^{L-1} \sum_{K=1}^{SL} \sum_{J=1}^{SL+1} (\theta_{jI}^{L})^2$; L = layers, SL = n de neuronas
- Backpropagation , nos ayuda a calcular las derivadas de manera más eficiente cuando hacemos optimización -> propagamos el error de la salida hacia las neuronas para la izquierda
    - $\frac{\delta}{\delta\theta_{ij}} J(\theta)$