Review: Machine learning and neurons

Sección orientada a ‘calentamiento’.

Lo que será cubierto:

- ¿Qué es ML?
  - Definición más técnica
- Clasificación y regresión.
- Neuronas.
- ¿Qué significa ‘aprender’ en ML?
- Código para poner lo aprendido en práctica.

¿Qué es Machine Learning (ML)?

Una definición común que podemos encontrar sería:

> “El "Machine Learning" o aprendizaje automático es un subcampo de la inteligencia artificial que se centra en enseñar a las computadoras a aprender y mejorar automáticamente a partir de la experiencia, sin ser programadas explícitamente. Esto se logra mediante algoritmos y modelos matemáticos que permiten a las máquinas analizar grandes cantidades de datos, identificar patrones y hacer predicciones o tomar decisiones basadas en esos datos.”

El problema con este tipo de definiciones es que es muy abstracta y vaga, una definición más realista y aterrizada a la realidad puede ser:

Un efoque ‘as dumb as possible’ y más realista es el pensar el Machine Learning como un problema de goemetría.

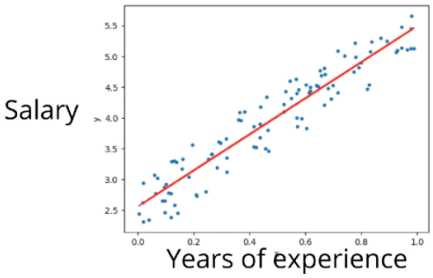
Ej:

Empezemos con un ejemplo de regresión.

La tarea de una regresión es ajustar una recta o curva. Geometría se trata de rectas, curvas, planos, circulos, etc. Esto es el porque algunos estadistic@s dirían “ML no es más que un ajuste de curva glorificado”.

Por lo que, para el ejemplo, imagina que eres un científico de datos en Linkedin, y quieres modelar como se mueven los salarios con respecto a los años de experiencia de un profesional de un rubro en particular, esto se vería de la siguiente

manera:

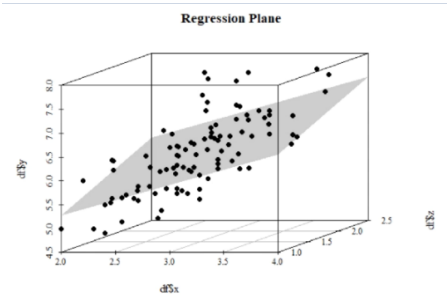


$$\hat{y} = mx + b$$

Siendo ‘m’ y ‘b’ los parámetros que deberemos encontrar como científicos de datos.

Si al mismo ejemplo le agregamos un grado de dificultad más, es decir, le agregamos más dimensiones al problema, nos encontraremos con ahora en vez de estar ajustando una curva en 2 dimensiones, lo haremos en n dimensiones, ej: agregando una característica de entrada adicional, como lo sería el nivel de educación (bachelors, masters, PhD, etc.).

Resultando un hiperplano.



Otra manera adicional de agregar más dificultad al problema es, en vez de querer ajustar una recta, podemos ajustar a algo no lineal, la vida en si tiene muchos problemas que son no lineales por lo que sería una representación más fiel a la realidad.

Ejemplo:

Cuantas push-ups puedo hacer antes de hacer daño a mi musculatura (esto plantea que infinitos push-ups ≠ mejor para mi musculatura)

otro ejemplo:

Clasificación

Al igual que la regresión, la clasificación es otro problema de aprendizaje supervisado.

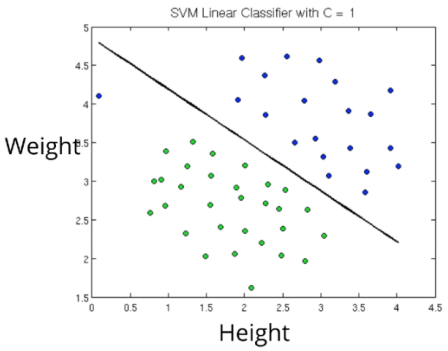
Cuando la regresión quiere predecir un número en un dominio continuo, la clasificación quiere determinar una clase/asignar un número discreto en un dominio determinado/etiqueta. Un ejemplo práctico de un ejercicio de clasificación es la clasificación de imágenes (por ej, un modelo que me pueda decir si la imágen que le estoy pasando corresponde a un gato o a un perro).

Ej: Supon que quiero clasificar el riesgo de accidente cardiovascular de pacientes dado su peso y altura.

Nuevamente, debo colectar una base de datos y guardar los datos de forma estructurada:

Subject ID	Height	Weight	At Risk?
1	6'	300 lb	1
2	5'5"	150 lb	0
3	5'2"	175 lb	1
4	5'10"	185 lb	0

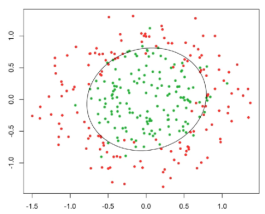
Una vez colectados mis datos, puedo graficarlos en un plano y codificarlos por color:

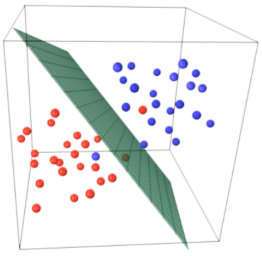


Nuevamente, el modelo más básico podría ser ajustar una recta que separe mis clases. Pero esta linea es diferente a la línea de regresión, que era una recta que se ajustara de la mejor manera a mis datos. Ahora trataremos de ajustar una recta que mejor pueda separar estas 2 clases, nuevamente, un problema de geometría.

De igual forma, si quisieramos clasificar el riesgo de que un cliente de un banco haga fraude o si vale la pena darle un crédito a alguien, la forma del approach al problema no cambia, solo cambian los datos en si.

Ejemplos de clasificación más difícil:





Nuevamente, independiente de lo complejo que se vaya convirtiendo el problema de clasificación, sigue pudiendo ser descrito como un problema de geometría.

Resumen de la lectura:

- Se deja atras la descripción “mágica” del machine learning,
- Se describe y se argumenta dando ejemplos del porque el ML es un problema de geometría,
- Se define la regresión: Ajustar una recta/curva lo más cercana a mis datos,
- Se define clasificación: Ajustar una linea/curva que separe de mejor forma mis clases de los datos.
- Independiente el espíritu de la tarea, los algoritmos aplican de igual transversalmente.

Preparación para programar

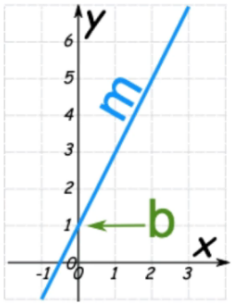
Las mayores diferencias que se darán en este curso a diferencia de uno clásico con scikit-learn son:

- Arquitectura del modelo: Se irá desde la ecuación hasta el input y prediction.
- Definición y programación de como el modelo es entrenado
  - Función de costo / pérdida / error,
  - Decenso del gradiente para minimizar el costo.

Arquitectura del modelo

Pensando en geometría, lo que tenemos es datapoints en un grid con clases a ser separadas por una recta o un hiperplano, pensemos por ahora en una recta. Si los dos ejes son llamados X1 y X2, podemos formular la ecuación de nuestra recta como:

$$w_1 x_1 + w_2 x_2 + b = 0$$

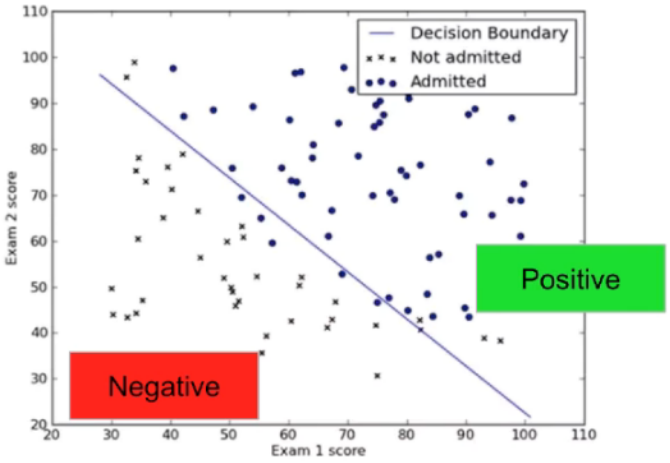


Esto no se ve como una ecuación clásica de una recta, la respuesta es dado a que mantener todo con una variable (x) es más útil, pronto se verá por que.

- **¿Cómo ocupamos la línea para clasificar?**

Afortunadamente dado las reglas de la geometría, si evaluamos un punto X el cual no está en la linea entonces tendremos un número mayor a 0 o menos a 0. De hecho, un data point cualquiera en uno de los lados de la recta siempre nos dará un número mayor que 0, y un datapoint cualquiera la otro lado de la recta nos dará un número menor a 0. Podemos convertir lo anterior fácilmente en un modelo de predicción. Todo lo que tenemos que hacer es tomar cualquier datapoint, pasarlo por la expresión de la línea y luego checkear su valor.

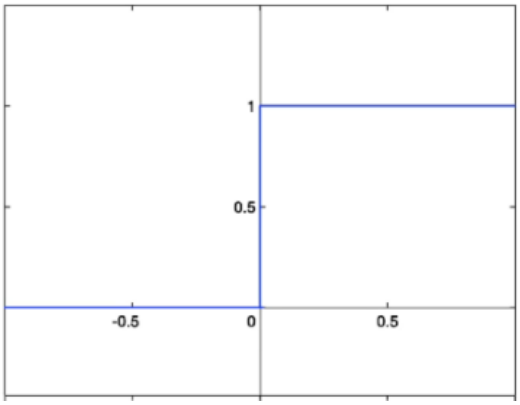
$$a = w_1 x_1 + w_2 x_2 + b$$
$$\text{if a} \geq 0 \rightarrow \text{predict } 1$$
$$\text{if a} \leq 0 \rightarrow \text{predict } 0$$



- Regla de decisión

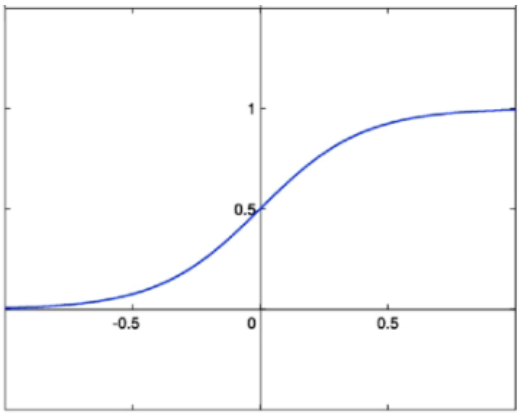
Matemáticamente se puede compactar en una función de escalón.

$$\hat{y} = u(a), a = w_1 x_1 + w_2 x_2 + b$$



En deep learning, se prefiere ocupar funcionas más “suaves”, que puedan ser diferenciables, como por ejemplo ‘sigmoide’.

$$\hat{y} = o(a), a = w_1 x_1 + w_2 x_2 + b$$



- Interpretación probabilística

Usualmente se interpretan estos outputs como: “La probabilidad de que $y = 1$ dado x”.

Para hacer la predicción, nosotros redondeamos a, por ejemplo: $\text{if } p(y = 1|x) \geq 50\% \rightarrow \text{predict } 1, \text{else } 0$

La función de forma de ‘S’ es llamada función **sigmoide.**

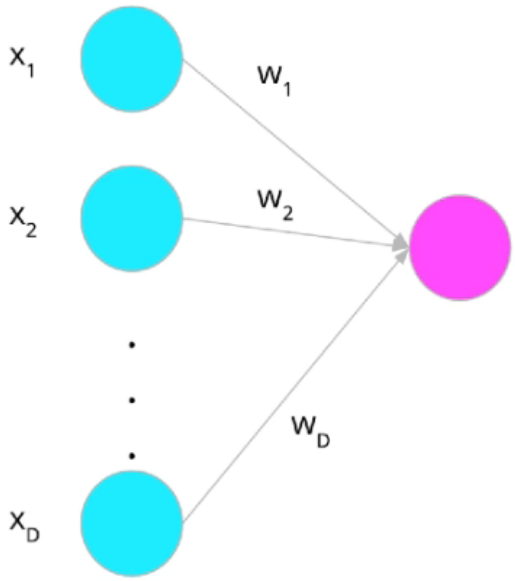
Este modelo es llamado **logistic regression.**

Sigmoid = Función logística.

Regresión logística con más de 3 inputs

Para poder computar una regresión lógistica con n número de variables de entrada, usamos matrices:

$$p(y = 1|x) = o(w^T x + b) = o(\sum_{d=1}^{D} w_d x_d + b)$$



Entrenamiento / Ajuste

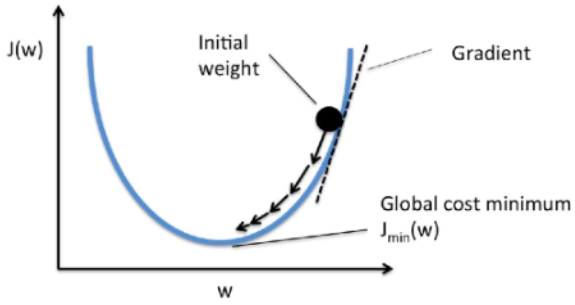
- **Función de costo**

Una función de costo, también conocida como función de pérdida o de error, es una medida que cuantifica la diferencia entre las predicciones realizadas por un modelo y los valores reales o deseados. El objetivo del aprendizaje automático es minimizar esta función de costo, es decir, ajustar los parámetros del modelo de tal manera que las predicciones se acerquen lo más posible a los valores reales.

Un ejemplo común de función de costo en problemas de clasificación binaria es la "binary_crossentropy" o entropía cruzada binaria. La entropía cruzada es una medida de la similitud entre dos distribuciones de probabilidad y, en el caso de la clasificación binaria, compara las probabilidades predichas por el modelo con las etiquetas verdaderas (0 o 1) del conjunto de datos. Al minimizar la entropía cruzada binaria durante el proceso de entrenamiento, el modelo aprende a hacer predicciones más precisas en términos de la probabilidad de pertenecer a una de las dos clases posibles.

- **Decenso del gradiente**

El descenso del gradiente es un algoritmo de optimización ampliamente utilizado en el aprendizaje automático y el aprendizaje profundo para minimizar la función de costo. Este algoritmo funciona de manera iterativa, actualizando los parámetros del modelo en la dirección opuesta al gradiente de la función de costo con respecto a esos parámetros. La idea es que, siguiendo el gradiente hacia abajo, el modelo encontrará el conjunto de parámetros que minimizan la función de costo y, por lo tanto, mejorarán su rendimiento en la tarea que se está resolviendo.



El optimizador "Adam" (Adaptive Moment Estimation) es una variante popular del descenso del gradiente que combina las ventajas de dos otros algoritmos de optimización: RMSprop y Momentum. Adam ajusta de manera adaptativa la tasa de aprendizaje para cada parámetro del modelo utilizando estimaciones del primer y segundo momento de los gradientes. Esto permite que Adam tenga un mejor rendimiento en comparación con el descenso del gradiente estocástico estándar y otros optimizadores en muchos problemas de aprendizaje automático y aprendizaje profundo.

- **Accuracy** (u otra métrica)

La métrica es un valor que permite evaluar el rendimiento de un modelo de aprendizaje automático en una tarea específica. A diferencia de la función de costo, que se utiliza durante el entrenamiento para ajustar los parámetros del modelo, las métricas se emplean principalmente para medir la efectividad del modelo y comparar diferentes modelos en términos de su desempeño en la tarea.

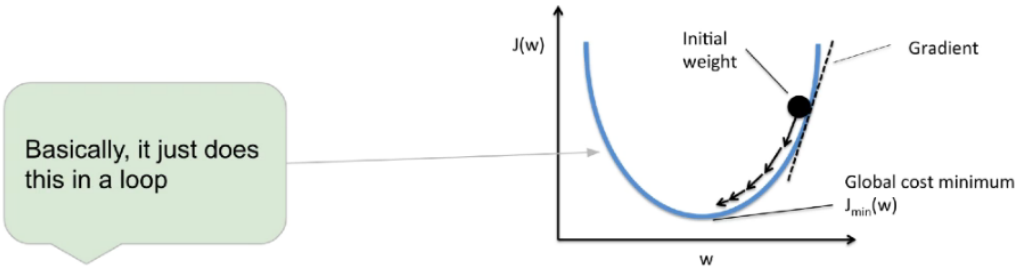
En el caso de los modelos de clasificación, una métrica comúnmente utilizada es la "accuracy" o precisión. La precisión mide la proporción de predicciones correctas realizadas por el modelo con respecto al total de predicciones. Es decir, indica qué porcentaje de las etiquetas predichas coinciden con las etiquetas verdaderas del conjunto de datos de evaluación. La precisión es una métrica intuitiva y fácil de interpretar, pero en ciertos casos, como en problemas con clases desequilibradas, puede no ser la mejor medida del rendimiento del modelo. En esos casos, otras métricas como la sensibilidad, especificidad, F1-score o área bajo la curva ROC pueden ser más apropiadas para evaluar el desempeño del modelo en la tarea de clasificación.

$$\text{accuracy} = \frac{\#correct}{\#total}$$

- **Entrenamiento / Ajuste**

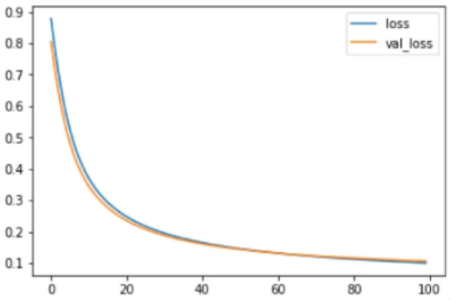
La función ‘fit’ toma 2 argumentos obligatorios, el set de datos para entrenar y el set de datos para validar su entrenamiento. También podemos ingresar el parámetro para el número de iteraciones, esto en DL es llamado ‘epoch’, el número de epoch necesarios no se pueden saber a modo determinista, es algo que debe ser trabajado modelo a modelo. Otro argumento que puede pasarse son los datos de validación, que son para ver como se comporta el modelo en datos que no fueron vistos en el set de entrenamiento.

```
r = model.fit(X_train, y_train,
              validation_data=(X_test, y_test),
              epochs=100)
```



Una vez el modelo está ajustado/entrenado, se entrega un objeto que tiene la historia del proceso de entrenamiento, particularmente, por cada iteraicón evaluará la función de pérdida. Lo que al final se termina es con un gráfico de pérdida por iteración y acccuracy por iteración, podremos ocupar esta información para tener una intuición de número de epochs, si necesito tunear más hiperparámetros, etc.

```
r = model.fit(X_train, y_train,
              validation_data=(X_test, y_test),
              epochs=100)
plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='val_loss')
```



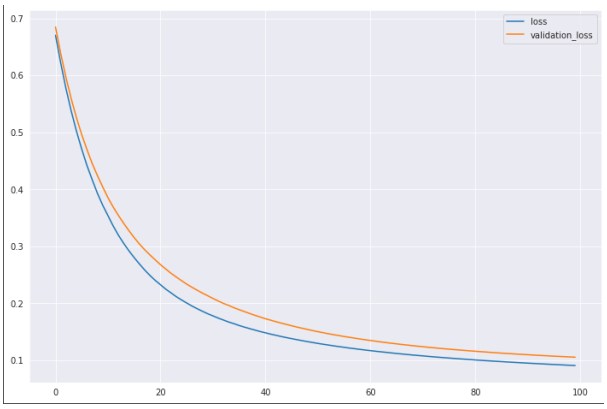
Code

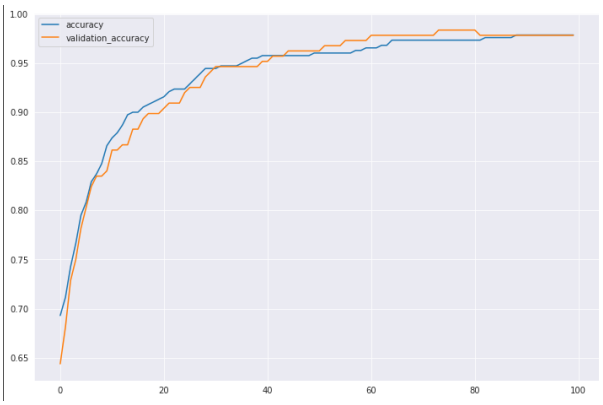
```
import tensorflow as tf
print(tf.__version__)

"""## Data pre processing"""

# load int the data
from sklearn.datasets import load_breast_cancer

data = load_breast_cancer()
data

type(data)

data.keys()

data.data.shape # (m, d) m: número de muestras y d: número de features/características

data.target # Problema de clasificación binaria

data.target_names

data.target.shape # m debe ser igual que en data

data.data

# Es importante saber qué significan las features, por lo que debemos ver qué
  # significa cada elemento

data.feature_names

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
                                                    data.data, data.target,
                                                    test_size=0.33, random_state=31
                                                   )

M, D = X_train.shape

print(M, D)

# Debemos escalar la data
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train) # Ajustados (fit) solo en el entrenamiento,
                                        # dado que los parámetros del modelo deben ser ajustado a partir de este set de datos.
X_test = scaler.transform(X_test)



"""## Model building"""

model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(D,)),
    tf.keras.layers.Dense(1, activation='sigmoid') # 1 porque solo queremos 1 output,
])

model.compile(
    optimizer='adam',
    loss='binary_crossentropy', # este loss va con el sigmoid y con la clasificación binaria
    metrics=['accuracy']
) #

"""## Model training"""

r = model.fit(X_train, y_train,
              validation_data=(X_test, y_test),
              epochs=100
              )

"""## Model results"""

print("Training score:", model.evaluate(X_train, y_train))
print("Test score:", model.evaluate(X_test, y_test))

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('darkgrid')

# Ajusta el tamaño del gráfico (ancho, alto)
plt.figure(figsize=(12, 8))

# Grafica las curvas de pérdida y validación
plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='validation_loss')

# Muestra la leyenda
plt.legend()

# Muestra el gráfico
plt.show()

# Ajusta el tamaño del gráfico (ancho, alto)
plt.figure(figsize=(12, 8))

plt.plot(r.history['accuracy'], label='accuracy')
plt.plot(r.history['val_accuracy'], label='validation_accuracy')

plt.legend();

plt.show()
```

Resultados

```
Training score: [0.09058354794979095, 0.9790025949478149]
6/6 [==============================] - 0s 4ms/step - loss: 0.1055 - accuracy: 0.9787
Test score: [0.10545062273740768, 0.978723406791687]
```


loss
validation_loss
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
20
40
60
80
100


accuracy
validation_accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0
20
40
60
80
100