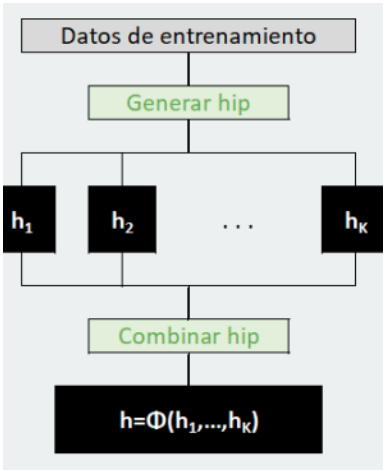
Resumen parcial ML

Ensemble



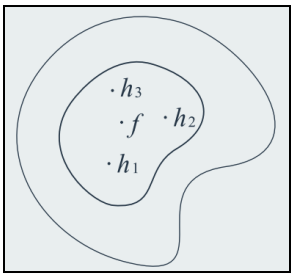
Técnica basada en:

1. Generar k hipótesis $h1,\ h2,\ \ldots,\ hk$
2. Fusionarlas utilizando una función $h(x) = \Phi(h1(x),\ h2(x),\ \ldots,\ hk(x))$

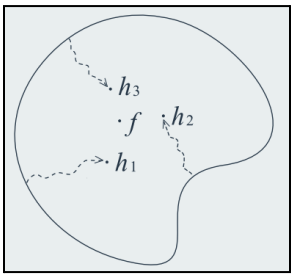
La función Φ define cómo combinar las hipótesis y puede ser fija o aprendida durante el entrenamiento.

- Promedio (regresión)
- Suma pesada (regresión)
- Voto mayoritario (clasificación)
- Voto pesado (clasificación)
- Soft-voting (clasificación)

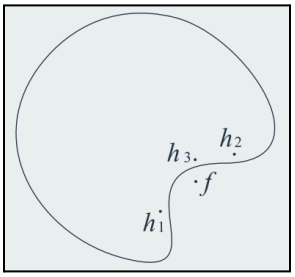
Beneficios de usar ensemble:



1. **Varianza Estadística (Statistical):** Este diagrama muestra que diferentes muestras de entrenamiento (diferentes conjuntos de datos "S") pueden llevar a la creación de hipótesis o modelos distintos (h1, h2, h3, etc.). En el contexto de ensemble, la idea es reducir la varianza estadística al promediar los resultados de múltiples modelos, lo que puede suavizar el efecto de anomalías en una muestra de entrenamiento particular.



2. **Varianza Computacional (Computational):** Ilustra el concepto de que al usar solo un modelo o hipótesis (h1), se podría llegar a una solución óptima local, pero no necesariamente a la mejor solución global. Los modelos de ensemble, al combinar múltiples modelos, pueden superar esta limitación y encontrar una solución mejor y más robusta.



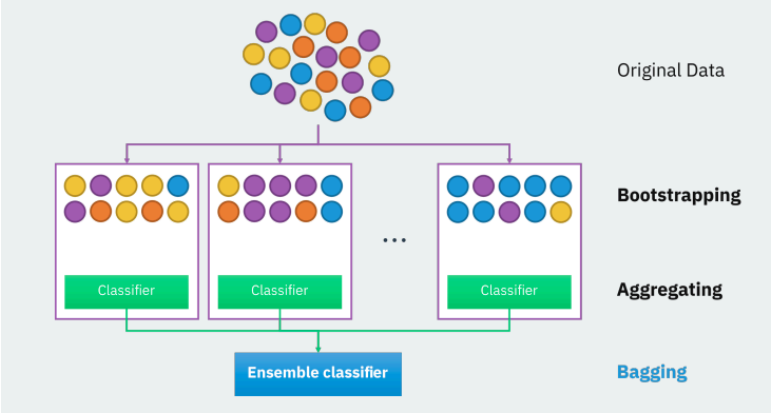
3. **Sesgo Inductivo (Representational):** Muestra cómo un único modelo (h1) puede tener un sesgo inductivo que le impide representar la función verdadera "f". Al combinar modelos con sesgos inductivos diferentes, los métodos de

ensemble pueden ofrecer una representación más completa y flexible de los datos, que potencialmente se acerca más a la función real "f".

Bagging

Para un set de datos con N elementos, consiste en generar B conjuntos (o bags) de M <N elementos, utilizando selección con repetición (Bootstrapping), clasificarlos y generar un único output en función de todos los B outputs de todos los conjuntos (Aggregating).

Bootstrapping + Aggregating = Bagging



Para cada “bag” (que puede tener datos repetidos con otros), entrenar la nueva hipótesis con esos datos y después ensamblarlos con las demás hipótesis

Es el mismo algoritmo de aprendizaje, pero con diferentes subconjuntos de datos. Mejora el desempeño de algoritmos inestables (alta varianza y sobre ajuste)

Se puede hacer algún caso para promediar también todos los datos que quedaron Out-of-bag.

Random Forest

Es un caso de Bagging. Consiste en tomar B conjuntos de M <N elementos cada uno (usualmente, $M = \sqrt{N}$) y construir un árbol de decisión para cada uno de ellos (bootstraping). Luego, para evaluar un atributo, se evalúa para cada conjunto y se utiliza agregación para concluir un único resultado. Formas comunes de agregación pueden ser el promedio o una votación.

Beneficios de RF sobre árboles de decisión

1. **Mayor Exactitud:** Los árboles de decisión individuales pueden sufrir de sobreajuste, especialmente si el árbol es muy profundo. Esto significa que pueden ser muy precisos en los datos de entrenamiento, pero no generalizan bien a datos nuevos. Random Forest crea un conjunto de árboles de decisión (un "bosque") y luego toma el voto mayoritario de las predicciones de todos los árboles. Esta técnica reduce el sobreajuste y, por lo tanto, mejora la exactitud en datos no vistos.

2. **Baja Varianza:** Un árbol de decisión puede cambiar drásticamente con pequeñas variaciones en los datos de entrenamiento, lo que significa que tiene una alta varianza. Random Forest combate esto al construir cada árbol a partir de una muestra aleatoria de los datos con reemplazo utilizando bagging. Además, cuando se divide un nodo durante la construcción del árbol, Random Forest selecciona la mejor división de un subconjunto aleatorio de las características en lugar de todas las características disponibles. Estos procesos aseguran que los árboles sean menos correlacionados entre sí y que el modelo final sea más estable y menos sensible a las variaciones en los datos de entrenamiento.

Al combinar las predicciones de múltiples árboles que han sido entrenados de esta manera, Random Forest suele producir un modelo más fuerte y confiable, superando así las principales desventajas de un único árbol de decisión.

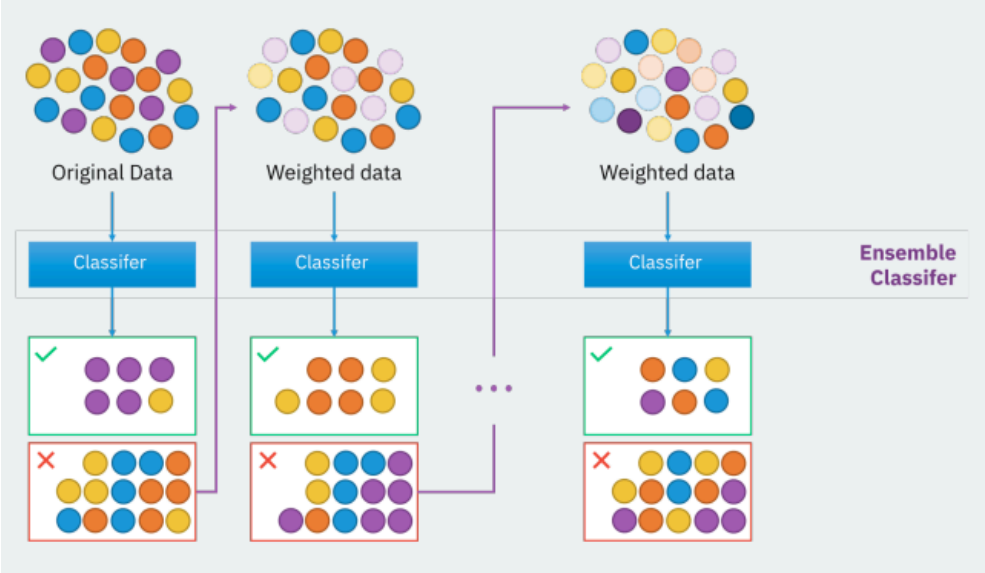
En random forest promediamos la disminución de impurezas.

Boosting

Es una técnica de ensemble que apunta a reducir el sesgo en modelos con alto sesgo. Consiste en:

1. Se construye una secuencia h1, h2, ..., hk de predictores débiles (generalmente árboles de 1 nivel).
2. Se le asocian pesos wi a cada observación (xi, yi).
3. Se asocia una confianza ci a cada hi, para luego combinar los predictores de manera ponderada

A alto nivel, el boosting genera secuencialmente predictores, corrigiendo errores del modelo anterior en cada iteración. La predicción final proviene de un promedio ponderado o voto ponderado.



Adaptive Boosting

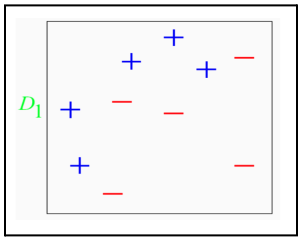
Se construye una secuencia de k clasificadores binarios débiles: A1(x), A2(x), ..., Ak(x), en donde las predicciones individuales **no** son tratadas por igual. La clasificación del modelo depende de cada clasificación individual de la siguiente manera:

$$A_{\text{boost}}(\boldsymbol{x}) = \text{Signo}\left\{\sum_{k=1}^{K} \alpha_k A_k(\boldsymbol{x})\right\}.$$

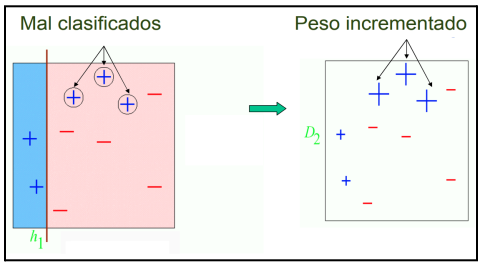
Los clasificadores se procesan **secuencialmente** y cada clasificador es dependiente de los resultados de los clasificadores que se ejecutaron antes.

Se usa una clasificación binaria con etiquetas {-1, 1}

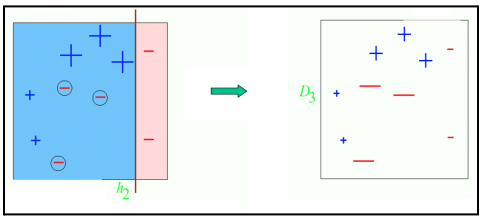
Ejemplo de un algoritmo de AdaBoost con Datasets:



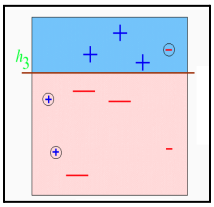
Dataset inicial (D1): Se muestra un conjunto de datos simple con observaciones de dos clases (positivas y negativas). Inicialmente, todos tienen el mismo peso.



Primera ronda (D2): Se genera una hipótesis $h1$ muy básica en base a un clasificador al azar para clasificar el dataset. Las observaciones que se clasifican incorrectamente (indicadas por un círculo) reciben un mayor peso. La segunda ronda, recibirá “puntos extra” por clasificar correctamente aquellos elementos que quedaron por fuera



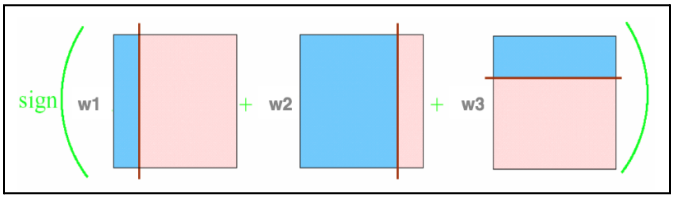
Segunda ronda (D3): Se entrena un segundo clasificador a través de otra hipótesis básica $h2$, enfocado en las observaciones que ahora tienen más peso. Nuevamente, las mal clasificadas reciben un peso aún mayor para la siguiente ronda.



Tercera ronda: Este proceso continúa para cada ronda, cada vez con clasificadores que se enfocan en las observaciones que los

clasificadores anteriores encontraron difíciles. En el ejemplo, se genera una última hipótesis $h3$

Modelo final: Al final del proceso, se combinan todos los clasificadores débiles para formar el modelo final. Cada clasificador débil vota para determinar la clase de una nueva observación, y el modelo AdaBoost toma en cuenta el peso de cada clasificador en la votación para llegar a una decisión final.



En resumen, AdaBoost trabaja en secuencia, adaptando los pesos de las observaciones y la importancia de cada clasificador basándose en su rendimiento anterior, con el objetivo de crear un modelo fuerte a partir de una combinación de modelos más débiles.

Gradient Boosting

Es similar al Adaptive Boosting, pero se enfoca en ir mejorando el modelo de tal manera que cada nuevo árbol se concentra en aquellas áreas donde el modelo anterior tuvo un rendimiento deficiente. Para hacer esto, se utiliza el gradiente de la función de pérdida. La continuación del modelo debe apuntar hacia donde la gradiente de la función de pérdida **disminuya.**

Redes Neuronales

Un modelo de redes neuronales consiste en construir una red de neuronas (funciones parametrizadas) que reciben y emiten estímulos (inputs/outputs).

- Modelo paramétrico:

$$\widehat{y} = f_{\theta}(x)$$

donde la función f está parametrizada por θ

- Regresión lineal - Modelo paramétrico básico de regresión

$$\widehat{y} = w_1 x^{(1)} + w_2 x^{(2)} + \cdots + w_D x^{(D)} + b$$

- Regresión logística - Modelo paramétrico básico de clasificación

$$\widehat{y} = \text{Sigmoidea}\left(w_1 x^{(1)} + w_2 x^{(2)} + \cdots + w_D x^{(D)} + b\right)$$

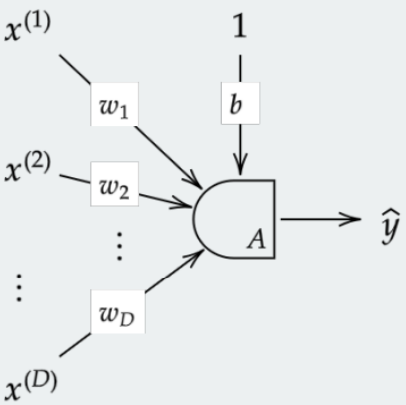
- En ambos modelos los parámetros son $\theta = (w, b)$

W es el vector de pesos y b es el término constante. Se pueden expresar como notación matricial: $\widehat{y} = A\left(x^{\top} w + b\right)$

- **A** es la función de activación. Se encarga de mapear los números reales a otros números reales, lo cual es crucial para introducir no linealidades en el modelo y permitir que el perceptrón realice clasificaciones no lineales.

Perceptron: modelo de una neurona:

Es una neurona o nodo que recibe n inputs distintos, les aplica una función θ y expone un output procesado.



Usando la notación matricial:

$$\widehat{y} = A\left(x^{\top} w + b\right)$$

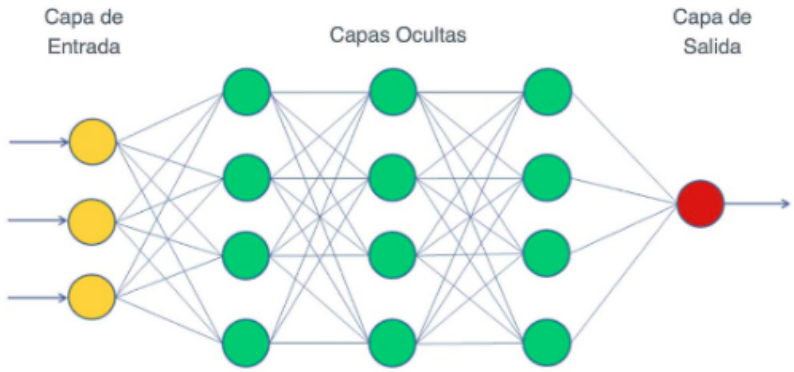
en donde

$$w = \begin{bmatrix} w_1 \\ \vdots \\ w_D \end{bmatrix} \quad x = \begin{bmatrix} x^{(1)} \\ \vdots \\ x^{(D)} \end{bmatrix}$$

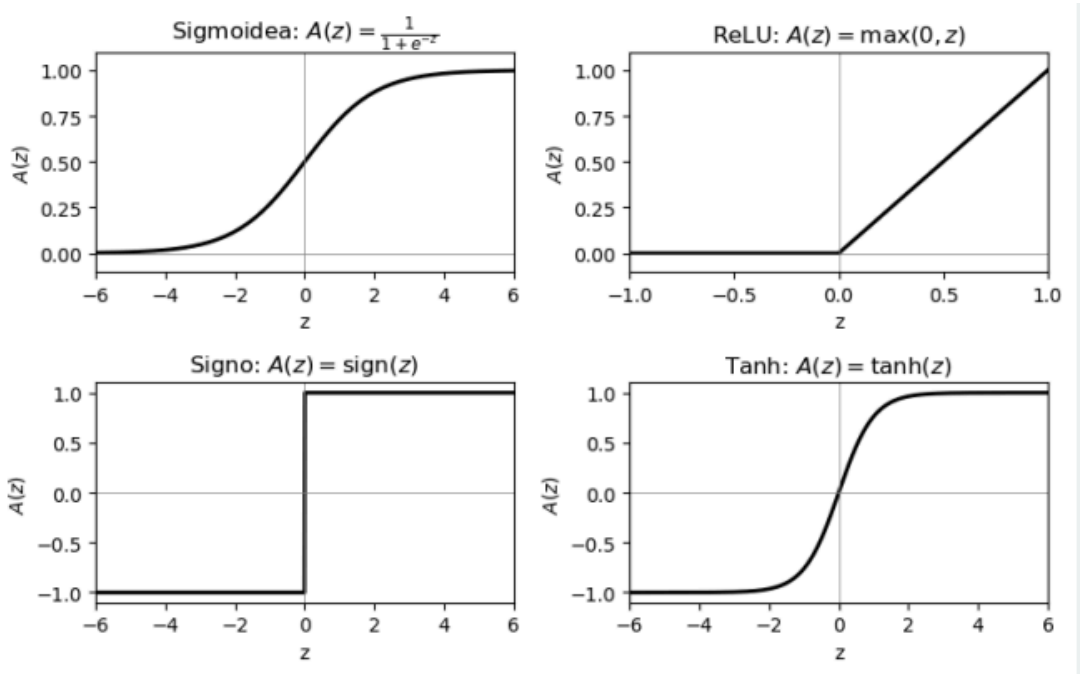
$A : \mathbb{R} \to \mathbb{R}$ activación

Perceptron multicapa:

Permite analizar inputs y exponer outputs mucho más complejos. El perceptrón multicapa es lo que llamamos red neuronal.
En resumen:
El perceptrón toma entradas, las asigna pesos, suma estos productos y luego aplica una función de activación para producir una salida, que puede ser utilizada para tareas de regresión o clasificación.
El entrenamiento de un perceptrón implica ajustar los pesos y el sesgo para minimizar el error entre las salidas predichas y las salidas reales en los datos de entrenamiento.



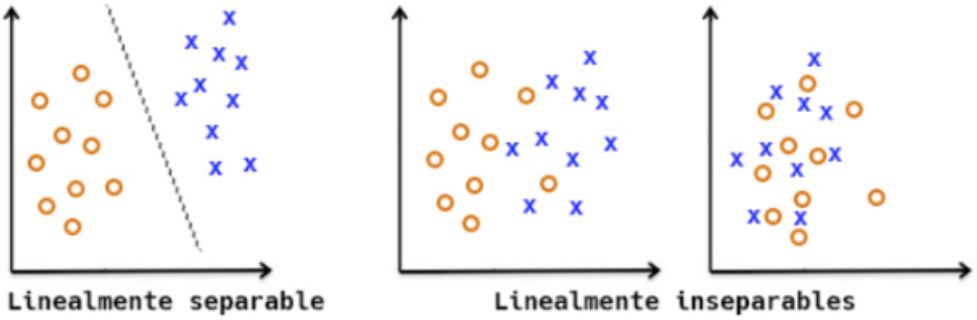
Funciones de activación que se encuentran dentro de cada neurona. Estas son las que se usan generalmente:



Separabilidad lineal

Un perceptrón simple se basa en la linealidad, lo que significa que **solo puede manejar problemas que son linealmente separables**, es decir, aquellos en los que se puede trazar una línea recta (o un hiperplano en dimensiones superiores) para separar las clases.

Para dos dimensiones, un problema es linealmente separable si se puede trazar una línea que separe los valores según etiqueta, de la siguiente manera:

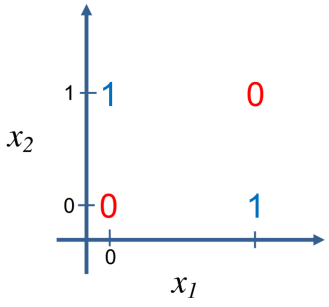


Para 3 dimensiones, es linealmente separable si se puede trazar un plano, y así sucesivamente.

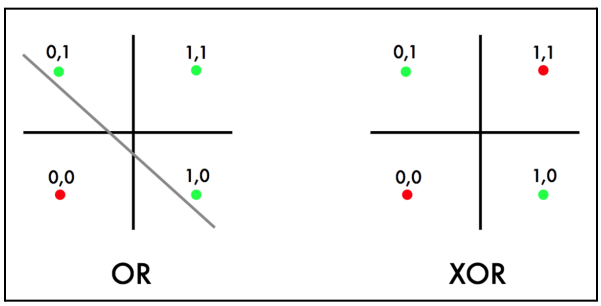
Dada la naturaleza de un perceptrón y su funcionalidad básica, sólo se pueden modelar problemas linealmente separables. Por supuesto que con un perceptrón multicapa este problema se soluciona, para cualquier dimensión, siempre que le podamos continuar agregando capas o más neuronas por nivel.

Ejemplo:

Hay un ejemplo muy conocido para ejemplificar este problema, y es a través del XOR. Un XOR da 1 si el número de entradas con valor 1 es impar. Gráficamente, un XOR con dos entradas $x1$ y $x2$, se ve de la siguiente manera:



Es natural notar como no es posible trazar una línea recta que separe los valores según su etiqueta (0, 1), por lo tanto, esto no es traducible a un perceptrón. Para un OR, si es posible:



¿Por qué los perceptrones sólo pueden enseñar funciones linealmente separables?

Un perceptrón simple, en su forma más básica, es un clasificador lineal. Esto significa que toma una decisión de clasificación basándose en una combinación lineal de los pesos y las entradas. Matemáticamente, esto se representa como un producto punto entre los pesos (w) y las entradas (x), más un sesgo (b). Como se vio anteriormente:

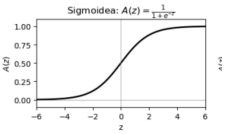
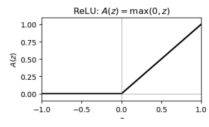
$$\widehat{y} = A(x^T w + b)$$

La decisión se toma pasando este resultado a través de una función de activación, que en el caso del perceptrón es una función de umbral o escalón.

La limitación clave aquí es la "combinación lineal". En dos dimensiones, una combinación lineal de entradas se puede visualizar como una línea recta en el plano de las entradas. En tres dimensiones, sería un plano, y en dimensiones más altas, un hiperplano. La función de activación de umbral del perceptrón (A) divide el espacio de entrada en dos partes: una en la que la salida es positiva (1) y otra en la que es negativa (0).

Regularizadores para entrenar un MLP (Multi Layer Perceptron)

- Ridge (norma l2)
- Lasso (norma l1)
- Dropout: Aleatoriamente apagar algunas neuronas.
- Batch normalization.
- Early stopping: usar un conjunto de validación “V” y parar de entrenar cuando el error en V no decrece por un cierto tiempo.
- Data augmentation

Tipos de redes	ACTIVACION	FUNC. DE PERDIDA	Salida	Explicacion	
Clasificación Multiclase	Soft-Max	CCE	N	Agarra todas las clases, las transforma en números entre 0 y 1, que entre todos suman 1. La clase resultante es la que tenga valor más grande	
Clasificación Binaria	Sigmoidea	BCE	1	El resultado da valores entre 0 y 1. Estamos hablando de una Regresión Logística.	
Regresion	ReLu/ID	MSE	1	Me tiene que dar un número.	

NLP - Natural Language Processing

El Natural Language Processing se basa en procesar un input en formato de lenguaje natural y procesar un output. El input puede ser texto, audio, video, etc. Un enfoque muy común es utilizar el siguiente proceso:

1. **Documentos:** Este es el punto de partida donde se recopilan los datos. Los documentos pueden ser cualquier forma de texto, como artículos, libros, publicaciones en redes sociales, etc.

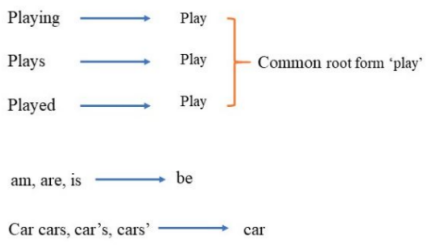
2. **Preprocesamiento:** Es la primera etapa de procesamiento de los documentos y puede incluir una variedad de tareas como la limpieza de datos (eliminar formato, números innecesarios, etiquetas html, etc.), la conversión a minúsculas para mantener la consistencia, y la eliminación de información irrelevante o ruido.

3. **Tokenización:** Consiste en dividir el texto en unidades más pequeñas llamadas tokens, que suelen ser palabras o frases. Esto permite al sistema tratar elementos lingüísticos individuales de manera más efectiva.

   Hola, este es un ejemplo.
   [“Hola”, “,”, “este”, “es”, “un”, “ejemplo”, “.”]
   [“Hola”, “este”, “ejemplo”]

4. **Eliminación de palabras vacías (Stop words removal):** Las palabras vacías son palabras comunes que no agregan mucho significado a una frase (como "y", "la", "es") y suelen eliminarse para mejorar la eficiencia del procesamiento y centrarse en las palabras significativas.

5. **Reducir a una forma raíz:** Esta etapa busca reducir las palabras a su forma base o raíz. Por ejemplo, "corriendo" se reduciría a "correr". Esto se puede hacer mediante algoritmos de stemming (que cortan los finales de las palabras) o lematización (que utiliza conocimientos lingüísticos para obtener la forma base correcta).

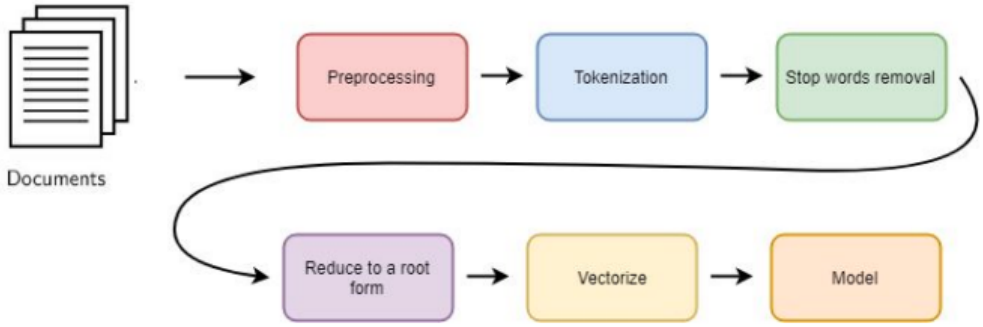


6. **Vectorizar:** Es la conversión de texto a una forma que las máquinas pueden procesar, generalmente vectores numéricos. Esto se puede hacer a través de técnicas como la del TF-IDF.

x_1: The quick brown fox jumped over the lazy dog.
x_2: The dog hunts a fox.

	the	quick	brown	fox	jumped	over	lazy	dog	hunts	a
x_1	2	1	1	1	1	1	1	1	0	0
x_2	1	0	0	1	0	0	0	1	1	1

7. **Modelo:** Finalmente, el texto preprocesado y vectorizado se alimenta a un modelo de NLP. Este modelo podría ser para clasificación de texto, análisis de sentimientos, traducción automática, etc. Los modelos pueden ser desde simples clasificadores estadísticos hasta redes neuronales complejas.



Normalización del vector (TFIDF)

$$tf(t,d) = \frac{f(t,d)}{\sum_{t'=d} f(t',d)}$$

$$idf(t,D) = log\left(\frac{N}{d \in D/t \in d}\right)$$

$$tfidf(t,d,D) = tf(t,d).idf(t,D)$$

f(t,d) = occurrences of t in c
t = term
d = document
D = corpus (all documents)

Token frequency: tf(t, d1) = (cant. ocurrencias del token “t” en el documento “d1”) / (Sumatoria de todos los tokens en “d1”)

Inverse Token Frequency: idf(t, D) = (1 / (cant. Documentos en los que aparece “t” / cant de documentos))