# Machine learning

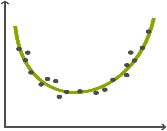
* Rama de la IA.
* Programas que aprenden desde los datos.
* Cercana a la estadística y métodos matemáticos.
* Muy en auge actualmente, muy útil.

# Técnicas de ML

## Aprendizaje no supervisado

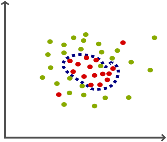
La idea principal es reconocer patrones o agrupaciones en un conjunto de datos. No interpreta los grupos, eso nos toca a nosotros.

## Aprendizaje supervisado

****La idea principal consiste en encontrar una función que explique la relación entre dos conjuntos de datos, es decir, que relacione los datos de entrada con los de salida. Es necesario tener ambos conjuntos de datos para encontrar la función.

Datos de entrada 🡪 f() ? 🡪 Datos de salida

Esta técnica se divide en dos subramas, según el tipo de dato de las salidas:

* **Regresión**: una función que pasa por todos los puntos, la cual toma números como entrada y devuelve un numero como salida.
* **Clasificación**: una función límite entre una serie de puntos que devuelve etiquetas.

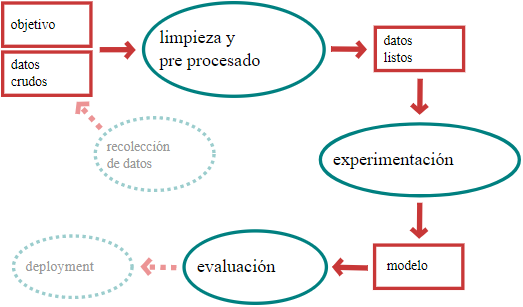
## Aprendizaje por refuerzo

La idea principal es encontrar una función que me permita accionar en el mundo para maximizar una meta. Nosotros tenemos un ambiente donde experimentar y una forma de medir los resultados.

A partir de un estado del mundo, tomo ciertas acciones para lograr ciertas metas. Se simula el ambiente y un agente realiza acciones en el mismo y aprende según sean éstas correctas o no.

Entorno 🡪 Acciones ? 🡪 Resultado

# El proceso de ML



* **Limpieza y pre procesado**: tomar los datos y convertirlos o normalizarlos de manera que el agente lo entienda y pueda utilizarlos.
* **Experimentación**: programar modelos, entrenarlos.

# Modelos paramétricos y no paramétricos

Hablando mal y pronto:

* **Modelo paramétrico**: arma una f() con algunos parámetros, y eso solo basta para hacer predicciones.
* **Modelo no paramétrico**: tiene una f() que consulta los datos. Para predecir hace falta la f() más los datos.

# Evaluación de performance: Métricas

## ¿Por qué necesitamos medir la perfomance?

* Porque necesitamos saber cuándo algo anda, y cuándo una cosa (experimento, modelo, set de datos) anda mejor o peor que otra.
* Para enfocarnos en el ambiente que queremos seguir, para mejorar el modelo, para compararlo con otro, para darle al cliente algo de calidad.
* Definir al inicio, con el cliente.
* Usar todo el tiempo.

## ¿Dónde medimos?

* Medir solo sobre los mismos datos que usamos para armar el modelo nos da información falsa. Nos **miente**.
* Hay que medir sobre **train** y **test**. Y a veces **validation**.

## Baseline

* Necesitamos superar lo que "ya se puede hacer fácil".
* Hay que establecer un piso contra el que comparar si realmente nuestro ML está haciendo algo útil.
* Utilizar el promedio para calcular el dato que necesitamos, de modo que tengamos una iniciativa para aplicar, antes de utilizar ML.

## Mean squared error y similares

* ¿Por cuánto le erramos en promedio?
* Sumar las distancias al cuadrado, para evitar que errores negativos cancelen errores positivos.
* Hay variantes. Útil especialmente en regresión.

## Accuracy

* ¿En qué porcentaje de casos acertamos la predicción?
* Útil especialmente en clasificación.
* Cantidad bien / cantidad total = accuracy.

**Problemas de Accuracy**

* Desbalances en la data
* Si erra por sí o por no tiene diferente impacto. Es mejor cuando erra algo que resulta positivo, por ejemplo, que prediga que hay cáncer cuando en realidad no lo había.

## Precision

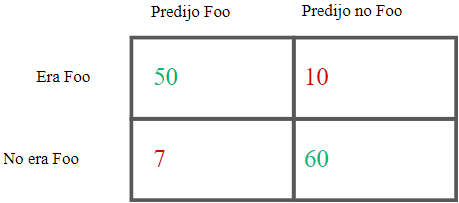
* ¿De lo que clasificamos como Foo, que porcentaje era realmente Foo?
* Útil si quiero asegurarme de solo encontrar ejemplos de Foo, no equivocarme al decir “esto es Foo”. Ejemplo: SPAM, asegurarme que lo que digo que es SPAM, realmente lo sea.
* Cantidad clasificada como Foo que era realmente Foo / cantidad clasificada como Foo = precision.

## Recall

* ¿De todos los Foo que había, cuantos encontramos?
* Útil si quiero asegurarme de que no se me escape ningún Foo.
* Cantidad clasificada como Foo que era realmente Foo / cantidad real de Foo que había = recall.

## Matriz de confusión

* Una forma de ver toda la información combinada.



## F-Score

* Combina precision y recall.
* Un promedio nos escondería si uno anda muy mal. F-Score cae mucho si alguno de los dos cae mucho.

## Auroc

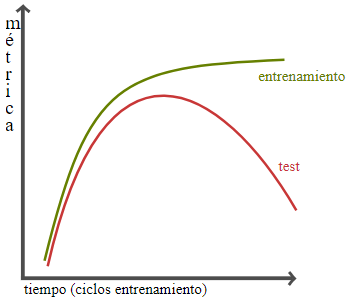
* ¿Qué tan bien diferenciamos entre verdaderos y falsos?
* Es una idea similar a F-Score: mejores valores cuanto mejor precision y recall.

## Métricas Custom

* A veces el problema es medible de forma específica al problema.
* Ejemplos: dinero ganado, productos vendidos, horas requeridas, tiempo ahorrado, etc.

## Curva de aprendizaje

* Nos ayuda a observar el progreso del aprendizaje.
* Es observar alguna métrica definida, a lo largo del loop de entrenamiento.
* Útil para ir comparando entre distintos modelos.
* Útil para saber cuándo dejamos de aprender, cuándo está overfiteando, etc.
  + A medida que va aprendiendo, precision es más preciso. Aunque llega un momento en el que la curva tiende a ser constante, y en este caso no vale la pena seguir.
* Sirve para comparar como mi modelo se comporta en el conjunto de entrenamiento y test.



# Analítica y pre-procesado de datos

## ¿Por qué? ¿Cuándo?

* Los datos no vienen usables casi nunca.
* Aunque usables, a veces los modelos requieren o andan mejor con la data en determinadas condiciones.
* Se hace al inicio, y se re-hace durante todo el proyecto.

## Limpieza de datos vs pre-procesado

* Limpieza: arreglar las cosas que no están bien en los datos.
* Pre-procesado: hacer que la data sea compatible/amigable con nuestros modelos.

## Limpieza

* **Valores nulos**:
  + Datos que faltan.
  + Podemos rellenarlos: valores a mano, o calculados (mas común, parecidos)
  + Podemos eliminarlos: siempre y cuando no sea un problema.
* **Datos redundantes**
  + Columnas repetidas, que contienen la información de otras columnas.
  + Hay que sacarlos porque muchos modelos tienen problemas graves con ese tipo de correlación. Menos datos es más simple (performance, visualización, etc.)
* **Datos duplicados**
  + Filas repetidas
  + Hay que sacarlos porque están mal. Todo depende del sentido que tenga el duplicado de esos datos.
  + Puede haber datos diferentes, pero iguales en las cosas con las que comparo y evalúo. Sexo y edad, por ejemplo.
  + Nos pueden muy fácilmente hacer llevar a predicciones y métricas re buenas, pero falsas, si el modelo se memoriza los ejemplos de train y luego ve los mismos ejemplos en test.
* **Outliers**
  + Datos aberrantes, muy diferentes al resto.
  + Hay que sacarlos porque muchas veces son datos incorrectos o falsos. Muchos modelos pueden deformar mucho sus predicciones para tratar de predecirlos bien.

## Pre-procesado

* **Escalar**
  + Si tenemos columnas numéricas, suele convenir escalarlas a rangos como (0, 1), (-1, 1), conservando su varianza.
  + Se logra que todas las columnas tengan el mismo peso y los valores tengan un valor similar.
  + Muchos modelos convergen más rápido con este tipo de valores.
  + Muchos modelos sufren mucho cuando hay valores que dominan las operaciones por estar en rangos más grandes (ej.: distancia al centro vs cantidad de habitaciones)
  + No confundir con **estandarización**: proceso en el que los ejemplos se convierten en vectores de largo 1.
* **One-hot encoder**
  + Para cuando hay categorías en una columna de entrada.
  + Convertir esa columna en tantas columnas como categorías haya, de forma que el dato indique si es o no esa categoría.
* **Count vectorizer**
  + ¿Qué hacemos cuando tenemos “documentos” en alguna columna de entrada? Documento = lista de cosas que son categorías.
  + Algo similar al one-hot encoder: columnas por “palabra” conocida, contando ocurrencias.
  + Se agregan n-gramas. Se va dividendo la frase y en base a eso se le asigna 1 o 0.
* **Composición y descomposición**
  + Convertir N features en una, o una en N.
  + Para facilitarle el trabajo al modelo. Proveerle más información, o información más procesada.
  + Para reducir necesidades de hardware.
  + Para exponerle al modelo información escondida.

# Aprendizaje supervisado

* Se utiliza cuando necesitamos que un modelo prediga o clasifique cosas, y ya tenemos ejemplos de predicciones/clasificaciones anteriores.
  + Puede que los humanos no sepan hacerlo por sí mismos (ataques de convulsiones).
  + Puede que los humanos lo sepan hacer, pero no tan bien (predecir partidos de fútbol).
  + Puede que los humanos lo hagan muy bien, pero necesitamos hacerlo en masa y automatizarlo (detectar objetos en imágenes).
* Es mágico, pero no es magia: las salidas tienen que depender de alguna forma de las entradas.

## Datos necesarios

* Ejemplos de entradas.
* Ejemplos de salidas para esas entradas (no salidas sueltas al azar).

## Regresión vs Clasificación

Depende de cómo son nuestras salidas:

* Números arbitrarios: **regresión**.
* Valores de un conjunto finito de etiquetas: **clasificación**. Puede ser una lista de etiquetas por imagen, por ejemplo.

Queremos una función que pase por los puntos o una función que divida los puntos.

## Función resultante

En ambos casos, nuestro modelo termina siendo una función que recibe datos de entrada y devuelve una salida.

* En **regresión**, esa función nos da las salidas directamente. El número que queremos.
  + Ejemplo: precio\_casa = modelo.predecir(datos\_casa)
* En **clasificación** depende:
  + La función es una **frontera de decisión**, porque dibuja una línea que separa los ejemplos de una clase y otra.
  + Hay algoritmos (modelos) que dan directamente la etiqueta como salida.
  + Otros nos dan un número que nosotros usamos para elegir la etiqueta. Lo bueno de este modelo es que (a diferencia del anterior), da un cierto porcentaje de seguridad/convencimiento en la decisión.
    - Es casi como hacer clasificación, usando regresión por dentro.
    - La mayoría de las librerías ahorran tener que hacer IFs a mano, lo hacen internamente. Sin embargo, es importante saber que es posible pedir el número también, para conocer la seguridad del modelo sobre la salida.

## Clasificación Multiclase

* Muchas veces vamos a tener que clasificar entre más de dos clases. Eso requiere de un modelo “multiclase”.
* Algunos modelos de clasificación son multiclase de forma “nativa” y otros no.
* Los que no, inicialmente solo funcionan para definir entre dos clases nomás: sí o no.
* Problema cuando se intenta clasificar más de dos clases con modelos que no son multiclase: muchos modelos a entrenar, implica utilizar muchos recursos de hardware.

# Regresión Lineal

* Uno de los modelos más simples de aprendizaje supervisado.
* Sirve para regresión. Se utiliza cuando sospechamos que las salidas dependen de las entradas de manera lineal. Si no, va a funcionar muy mal.
* Es muy rápida. Es útil para probar rápidamente una solución (por más que no sea buena para el problema).
* Es hacer una regresión, de una función con forma de recta.
* Para ello necesitamos encontrar valores para los parámetros de nuestra recta.
* Descenso por el gradiente nos encuentra los valores para esos parámetros (no es la única forma, hay otras que no vemos).

## Cómo funciona

* Sospechamos que existe esta función, y que es una recta: .
* La regresión prueba muchas rectas para ver cuál se ajusta bien, y define una recta predictora.
* La salida de nuestro modelo es la función que permite predecir, es decir, solo se utiliza la función obtenida y no los datos. Lo aprendido se guarda en los parámetros de la función.

## Entrenamiento

No se prueba al azar. Hay una forma inteligente de probar parámetros (no solo se aplica a regresión lineal): función de error.

## Función de Error

Función que evalúa el error de los parámetros establecidos para el conjunto de datos.

# Descenso por el Gradiente (Gradient Descent)

* Consiste en descender (reducir el error) usando la derivada de la función de error para saber para qué lado mover P y hacer que el error disminuya su valor.
* Al finalizar, se encuentra un valor de P que es lo suficientemente bueno, según las métricas definidas: cantidad de iteraciones, porcentaje de error, etc.
* Esta misma técnica se puede utilizar para aproximar cualquier función que se desee. Solo hay que:
  1. Armar el molde de la función (tipo de función).
  2. Elegir parámetros.
  3. Hacer descenso por el gradiente de esos parámetros.
* Preguntas frecuentes:
  + Si no se conoce la estructura, hay que probar, pero no se podrá usar el algoritmo hasta saberlo.
  + Si se tienen muchos parámetros, se soluciona con derivadas parciales. No es tan lento en la actualidad (lo es, pero se puede vectorizar, usar GPUs, etc.).
  + El problema definirá hasta dónde/cuándo seguir.
  + El problema definirá qué tanto error se permite.
  + Si la función no es derivable, hay soluciones complejas (que no veremos).
* El descenso por el gradiente tiene variantes: Batch, Nesterov momentum, Adam.

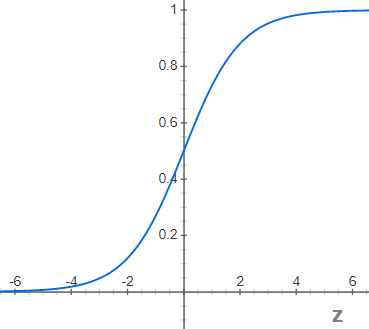
Learning Rate

Para mejorar el descenso, se multiplica el valor de la derivada por un “**learning rate**”, que se encarga de regular el tamaño de los saltos. No hay una regla objetiva para establecer el valor, pero normalmente se prueba y evalúa cómo funciona.

Si el learning rate es:

* Muy pequeño: se necesitarán demasiadas iteraciones para llegar a una solución aceptable (pero llegará en algún momento). Mejora muy lentamente.
* Muy grande: es posible que comience a saltar de un lado a otro y nunca logre encontrar la solución. Empeora.

# Regresión Logística

Queremos: (valores entre 0 y 1).

Función sigmoide o logística

* Siempre tiene valores entre 0 y 1.
* Los valores cercanos al medio (0,5) son bastante simétricos.

¿Cómo hacer que los datos se ajusten y queden comprendidos en esta función?

Antes:

La ecuación representa los valores predictores o parámetros que se tengan multiplicados por los datos (x). Theta es el vector con los “pesos” de cada x (feature/propiedad de entrada).

Ahora:

Que es igual a:

Esto hace que ahora obtengamos valores entre 0 y 1, sin importar cuál sea nuestra función inicial.

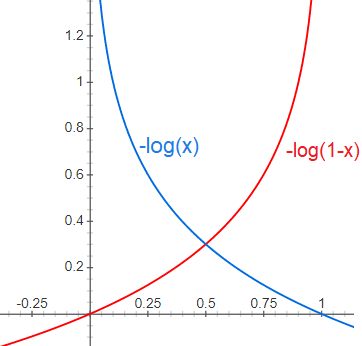
Es decir, el resultado podrá ser tomado como la probabilidad de que con el conjunto de datos y parámetros dados.

Para algunos problemas, no nos da lo mismo equivocarnos al decir verdadero o falso (caso de enfermedades, por ejemplo), por lo que quizá queramos correr el umbral o threshold. Se llaman problemas cost-sensitive.

Frontera de decisión

Si definimos valores para , podemos establecer la frontera de decisión, despejando la ecuación.

Sirve para discernir qué casos pertenecen a un grupo u otro.

¿Cómo aprendemos?

Optimizar .

El error cuadrático es una función poco conveniente para nuestro objetivo: no es convexa, por lo que presenta muchos mínimos locales.

Por eso, se utiliza otra función de costo para la regresión logística:

Esta función de costo se puede simplificar y expresar como una única función J derivable:

# Vecinos cercanos (k-Nearest Neighbors o k-NN)

* Se lo conoce también como aprendizaje o razonamiento basado en casos, o “Lazy learning” (no hay cálculo alguno hasta la fase de clasificación).
* El modelo no existe explícitamente.
* El único parámetro es el k (número de vecinos a utilizar). No es paramétrico porque k no almacena conocimiento obtenido, es solo un parámetro de funcionamiento.
* Es un método fácil de entender y de implementar, pero puede presentar algunos problemas de performance. En términos generales, funciona bien con muchos datos y pocas dimensiones.

## Forma de medir la distancia

Distancia de Minkowsky, también conocida como distancia Euclidea (p=2) o de Manhattan (p=1). Consisten en medir la distancia en cada parámetro o dimensión que se tenga.

Se deben normalizar los datos para evitar que la suma de las distancias no se vea inclinada por alguno de los parámetros.

## Algunas alternativas

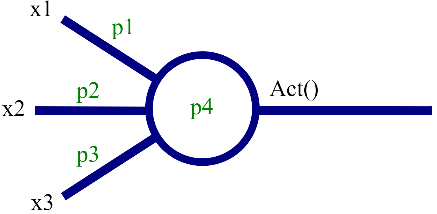
* K-NN con rechazos:
  + Demanda algunas garantías para predecir, por ejemplo, que exista una mayoría de 75% sobre la clase predominante.
  + Si no se consigue, se deja sin clasificar hasta que se consigan más datos o se utilice otro método.
* K-NN de distancias medias:
  + Se calculan las distancias medias de todas las clases y se asigna la clase que tiene en total una distancia menor.

# Redes Neuronales

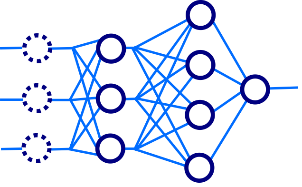
* Uno de los modelos más complejos y poderosos.
* Idea vieja, pero resucitada por disponibilidad de datos y hardware.
* Para aprendizaje supervisado, no supervisado y por refuerzo. Nos centramos en supervisado.
* No suele ser buena idea usarlas como primer intento para resolver un problema.
* Son multiclase de forma nativa, porque permiten especificar muchas salidas.

Las usamos cuando:

* Modelos más simples no funcionan.
* Tenemos hardware y datos. Sobre todo, datos, porque las RN tienden a hacer overfitting.
* Trabajamos con imágenes.
* Alguien ya entrenó y publicó una red que hace lo que queremos, o muy parecido.

Neurona

* Vagamente inspirada en neuronas reales.
* N entradas, 1 salida.
* La salida resulta de hacer una operación con las entradas y pesos internos 🡪 es una función.
* A cada entrada la multiplica por su peso. Suma los resultados y suma el peso global. Pasa dicho valor por una función de activación y devuelve un valor.
* Si 🡪 regresión logística.
* Limitación: una única neurona solo sabe separar por una recta. Muchas neuronas juntas podrían hacer algo mejor.

Red Neuronal

* Este es un tipo de red neuronal específico: Multi-Layer Perceptron, o MLP.
* Es una red “feed forward”: las operaciones se hacen en una sola dirección, hacia adelante.
* Es una red “completamente conectada”: todas las salidas de una capa están conectadas a las neuronas de la siguiente capa.
* Podemos agregar muchas capas, con muchas neuronas en cada capa. Consecuencia: entrenar será mucho más complejo, tendrá más parámetros, tendrá más operaciones, etc. Será más pesado calcular salidas y derivadas. Además, cuanto más grande sea una red, mayor capacidad de overfitting tendrá.
* Hay una **capa de entrada** y una **capa de salida**. Las demás son **capas ocultas**.
* Se suele dibujar a las entradas como neuronas también, aunque no lo sean realmente.
* Es demostrable que una sola capa oculta es suficiente para poder lograr cualquier función. El tema es cuántas neuronas, y qué tan fácil aprende eso.
* Podemos tener muchas salidas si ponemos varias neuronas en la capa de salida.

Dónde está el conocimiento

* El conocimiento se almacena en los **pesos** de los distintos parámetros de las neuronas.
* Es posible guardar eso en disco para volver a usarlo luego (escribirlo a mano sería difícil).

Cómo encontramos los pesos

* Mediante descenso por el gradiente. Hay otras opciones, como algoritmos genéticos.
* Las derivadas de estas funciones gigantes son complejas de calcular. Afortunadamente, las herramientas las resuelven internamente.
* Problema: si tocamos un peso de una neurona de atrás, eso también afectará a todas las siguientes neuronas, alterando todos los cálculos y haciendo que no funcione el descenso por el gradiente. Para resolver esto se usa **backpropagation**.

Backpropagation

* Algoritmo que calcula el cambio que hay que hacer en cada peso, teniendo en cuenta su relación con los pesos siguientes y el resultado. Un peso del final se modifica sin problemas. Sin embargo, un peso del inicio debe modificarse teniendo en cuenta cómo influye lo que hay después que él.

Arquitectura de la Red

Decidir:

* **Cantidad de capas.**
* **Cantidad de neuronas por capa.**
  + Pocas neuronas: podemos no lograr buen resultado.
  + Muchas neuronas: más lento, puede sobreentrenar.
* **Conexiones entre capas.**
* **Función de activación.**
  + **Relu** (plana y en un punto se pone lineal), **Tanh** (tangente hiperbólica, varía entre [-1,1]) o **Sigmoid** en capas intermedias.
  + **Sigmoid** en capa de salida de redes con una sola salida entre 0 y 1.
  + **Softmax** (parecida a Sigmoid, pero garantiza que las salidas de todas las neuronas sumen 1) en capa de salida de redes con N salidas entre 0 y 1.
* **Variante de descenso por el gradiente.**
  + Normalmente “Adam” anda bien, en casos generales.
* **Variante de función de error.**
  + Binary\_crossentropy para redes con una salida entre 0 y 1 (clasificación binaria).
  + Categorical\_crossentropy para redes con N salidas entre 0 y 1.
  + A veces puede hacer falta configurar “pesos” para los errores positivos y negativos.

Consejos

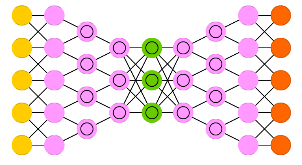
* Normalizar las entradas.
* Probar cambios de a una cosa a la vez.
* Entrenar y testear con sets separados. Las redes neuronales pueden sobreentrenar muy fácil.

Técnica relacionada útil: dropout

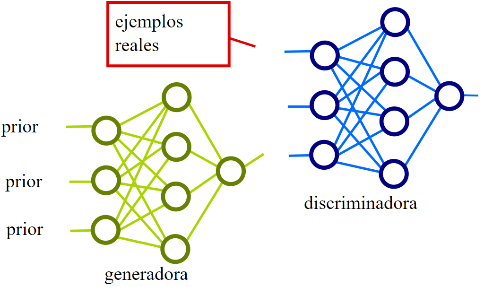
* Si el sobreentrenamiento se produce rápidamente, la forma más simple de solucionarlo es quitar neuronas (siempre que sea posible o útil).
* Dropout permite armar una red neuronal con muchas neuronas a la que le cueste sobreentrenarse, a costa de tener mayor tiempo de entrenamiento.
* Elige conexiones al azar en cada ciclo de entrenamiento y las “corta”.
* Sirve para evitar que cada capa se “memorice” resultados que tiene que dar para determinados inputs (overfit interno y de la red en general).

**Zoológico de redes neuronales**

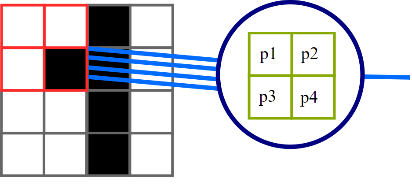
* MLP es el tipo más común, pero hay muchas otras formas de redes neuronales.
* Principalmente, lo que cambia es la arquitectura de la red.

Autoencoders

* Entrenamos usando las entradas también como salidas. Esperamos que la salida sea lo mismo que entró (o lo más parecido posible).
* Dado que la red se achica en el medio, debe comprimir la información de entrada de alguna forma para que sea expresada en menos números.
* Desde esos pocos números tiene que volver a generar las salidas originales, por lo que debe ser capaz de descomprimir la información comprimida.
* Termina aprendiendo dos redes juntas: una que comprime información, y otra que la descomprime.
* También se puede usar para reducir dimensionalidad de un set de datos (para visualizarlo, dárselo a modelos más simples, etc.).
* Desventaja: puede perder información. Ventaja: al ser específico para el problema atacado, es posible que permita comprimir de una manera mucho más eficiente.

Generative Adversarial Networks (GANs)

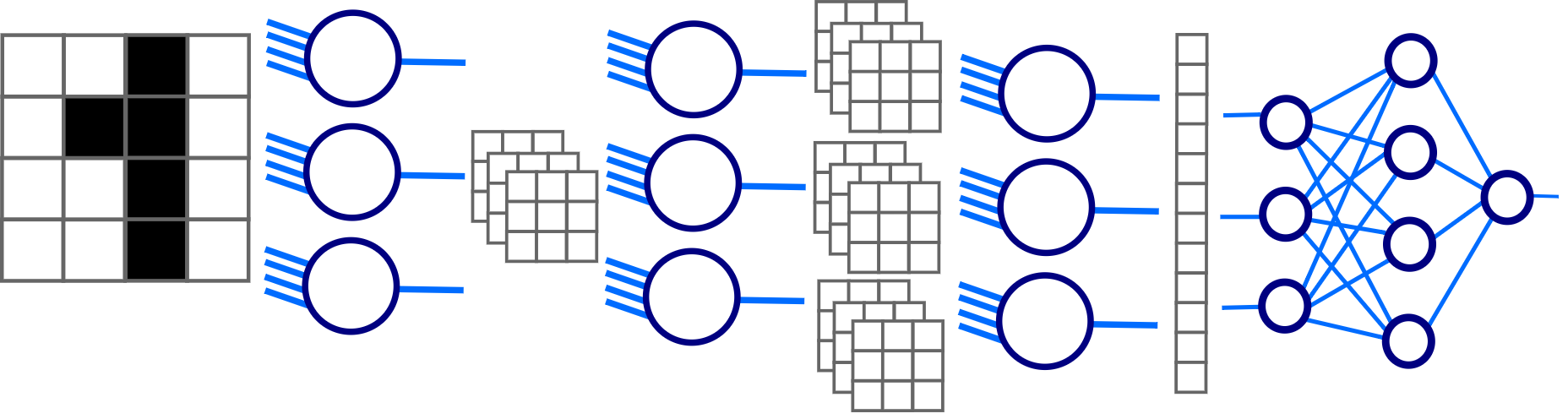
* La **red discriminadora** intenta predecir correctamente si sus entradas son un ejemplo real o un ejemplo generado.
* La **red generadora** trata de generar ejemplos que confundan a la red discriminadora, a partir de entradas (random, o algún dato relacionado, como un dibujo a mano).
* Primero se entrena la red discriminadora hasta que sepa reconocer bien la cosa que queremos generar después.
* Luego se comienza a generar ejemplos con la generadora, y se los pasa a la discriminadora. Cuanto mejor prediga la discriminadora, más error le devolvemos a la generadora.
* Seguimos entrenando ambas redes a la vez, que a partir de ahora compiten (una por detectar bien perros reales, la otra por generar perros que parezcan reales).

Convolucionales

* Son muy efectivas para clasificación de imágenes. El estándar hoy en día.
* Son muy pesadas de calcular.
* Implica determinar cosas nuevas: qué tamaño tienen los kernels, cuántas convoluciones, etc.
* Un tipo de convolución especial útil: Maxpolling ("el número más alto visto en una región")

Funcionamiento

* Si se intenta suministrar información de una imagen en forma de vector a una red neuronal, se perderá información espacial. Si bien la información está, es más difícil de detectar para la red. La solución es trabajar con matrices en vez de vectores, y permitir que las neuronas vean partes de la imagen en lugar de ver todas las entradas juntas.
* Convolución: recibe como entrada una matriz que corresponde a una porción de la imagen. La misma neurona se usará en la misma imagen múltiples veces para distintas “entradas” (regiones de la imagen). Con estos resultados se arma una nueva matriz más chica.
* Podemos tener muchas de estas neuronas al mismo tiempo, lo que generará una salida de muchas matrices a partir de la misma imagen. Asimismo, podemos concatenar convoluciones.
* Finalmente, las salidas se convertirán en un vector que podrá ser procesado por una red neuronal tradicional (MLP).
* Las convoluciones pueden terminar aprendiendo "features" de la imagen de a poco, desde conceptos más simples a más complejos.



Deep Learning

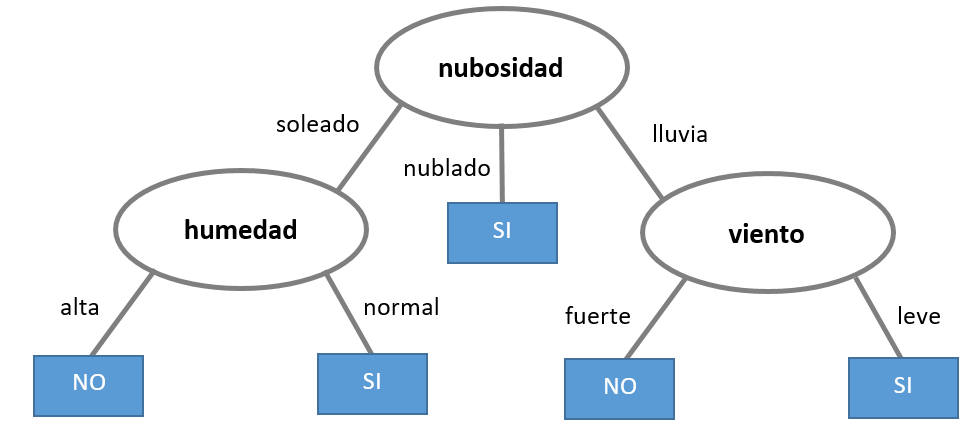
Modelos con capas que aprenden features y esto sirve a las capas siguientes para aprender nuevas cosas.

# Árboles de decisión

* Uno de los métodos más utilizados.
* Fáciles y rápidos de construir.
* Modelo no-paramétrico: posee diversos hiper-parámetros, pero el modelo en sí no tiene pesos.
* White-box model: permiten inferir conocimiento de qué es lo que está pasando.
* El proceso de construcción del árbol realiza un filtrado de variables (estilo embedded).
* Pueden ser expresados como conjuntos de reglas "if-then".
* Podemos utilizarlos para clasificación y regresión.
* Un camino del árbol es una **conjunción** de pruebas sobre distintos atributos. El árbol entero es una **disyunción** de esas conjunciones.
* Classification And Regression Trees (CART) permite trabajar con problemas de regresión obteniendo el promedio de las instancias que quedan en cada hoja. Es uno de los algoritmos más utilizados.

Se utilizan cuando

* Interesa dar explicaciones de las salidas del modelo.
* Se necesita prototipo rápido: no se necesita demasiado pre-procesado.
* Tratan dominios complejos donde no existe una clara separación lineal en los datos.
* Se debe predecir rápido ().
* Se tienen variables categóricas y numéricas.
* Se tienen más de dos clases, o múltiples etiquetas.

Representación del árbol

* Cada nodo especifica una prueba contra algún atributo (variable de entrada).
* Cada rama descendente está relacionada con algún valor posible de dicho atributo.
* Cada nodo hoja tiene una etiqueta o clase determinada.

Algoritmos de aprendizaje

Existen muchos algoritmos distintos: CLS, ID3, C4.5, ID4, ID5, C4.8, C5.0.

Algoritmo básico: ID3 (Iterative Dicotomiser)

* Búsqueda voraz (greedy, busca la opción óptima en cada paso) por el espacio de todos los árboles posibles.
* Enfoque top-down: el árbol se construye desde la raíz hacia las hojas.
* Proceso:
  + En cada nodo se pregunta qué atributo debería ser utilizado en ese momento.
  + Crea una rama por cada valor posible de ese atributo.
  + Se repite el proceso sacando del conjunto de atributos los que ya se utilizaron anteriormente en esa rama.
  + Termina cuando todas las instancias pertenecen a la misma clase o todos los atributos fueron utilizados.
  + Cada nodo hoja se etiqueta con la categoría que tenga mayor cantidad de instancias.

¿Cómo elegir cuál es el mejor atributo en cada caso?

Se desea elegir el atributo que haga que los datos se separen lo mejor posible. Para ello, se utilizan los conceptos de:

* Información Mutua: mide la dependencia mutua de dos variables, es decir, mide la reducción de la incertidumbre (entropía) de una variable aleatoria X, debido al conocimiento del valor de otra variable aleatoria Y.
* Entropía: mide la variabilidad de una variable o fuente de información.

Observaciones

* El algoritmo no asegura encontrar el óptimo global.
* La complejidad se incrementa de forma lineal con el número de instancias de entrenamiento y de forma exponencial en relación al número de atributos.
* Variables con mayor cantidad de valores son favorecidas en la selección.

Mejoras posibles

* Establecer una profundidad máxima para el árbol.
* Manejo de variables continuas.
* Permitir valores nulos en los datos de entrada.
* Definir pesos a las clases o etiquetas para problemas cost-sensitive.
* Post-pruning (poda del árbol).

Desventajas

* Facilidad de sobreentrenamiento.
* Inestabilidad. Pequeños cambios en los datos generan que el árbol resultante cambie notablemente.
* Encontrar el árbol óptimo es un problema NP-completo.
* Hay que tener cuidado si el dataset no está balanceado.

Consejos prácticos

* Utilizar el menor número de variables posible para evitar sobreentrenamiento. Usar técnicas de reducción de dimensionalidad y max\_depth.
* Analizar los árboles resultantes: dan mucha información de qué es lo que está pasando.
* Balancear los datos.
* Pueden ser utilizados para hacer análisis post-mortem en otros problemas.

# Ensembles

También conocidos como metaclasificadores.

Motivación

* Distintos algoritmos funcionan correctamente en distintas situaciones. A priori, no es factible saber qué algoritmo funcionará mejor.
* Por más que ajuste muy bien en el conjunto de test, aún puede estar fallando en algunos casos. Podría existir otro algoritmo que ajuste mejor dichos casos.
* A la hora de predecir, combinar distintas opiniones puede ser una excelente idea. Debe existir variabilidad en las combinaciones.

Formas de caracterización

* Según el conjunto de datos de entrada:
  + Utilizando distintos espacios de predictoras: a cada método se le suministra una porción de los datos.
  + Utilizando el mismo espacio de predictoras: a cada método se le suministra la totalidad de los datos.
* Según la forma de los clasificadores bases:
  + Homogéneos: conjunto de clasificadores de un mismo tipo.
  + Heterogéneos: conjunto de clasificadores de distintos tipos.
* Según la estructura del ensemble:
  + Paralelos: todos los clasificadores bases realizan una predicción y se combinan de alguna manera para obtener una única salida.
  + Seriales: se consultan los clasificadores bases de forma serial, donde cada uno recibe los datos de entrada y la salida del clasificador previo.
  + Jerárquicos: se establece una jerarquía y las salidas de los clasificadores base constituyen las entradas del metaclasificador.

Métodos básicos

* Fusión de etiquetas
  + Voto por mayoría.
  + Mayoría simple (la clase debe tener, al menos, la mitad de los votos).
  + Voto por mayoría con umbral (la clase debe superar una cantidad mínima de votos).
  + Voto por mayoría ponderado.
  + Usando probabilidades: media aritmética, mínimo, máximo, mediana, media ponderada.

Métodos avanzados

* **Bagging: Bootstrap AGGregatING**
  + Metodología para construir el ensemble (no un algoritmo en sí).
  + Se crean L conjuntos de datos a partir del conjunto inicial, para entrenar L clasificadores.
  + Para predecir se combinan las salidas de los L clasificadores utilizando alguna técnica (voto por mayoría).
  + Cada conjunto de datos se genera utilizando "bootstrap" (muestreo con reemplazamiento): algunas instancias van a quedar repetidas y otras se eliminarán.
  + Tiene sentido con clasificadores base que sean inestables.
  + Si se tienen muchos datos, los conjuntos probablemente serán similares y el método perderá eficacia.
  + Procedimiento
    - Fase de entrenamiento:
      * Para L clasificadores base, generar L conjuntos de datos con muestras bootstrap a partir del conjunto de datos original.
      * Entrenar los L clasificadores, cada uno con uno de los conjuntos.
    - Fase de predicción:
      * Clasificar la instancia con los L clasificadores.
      * Combinar las salidas y devolver un único resultado.
* ***Random Forest*** 
  + Es un algoritmo que utiliza Bagging.
  + Ensemble homogéneo y paralelo: árboles como clasificadores bases y creados de forma independiente.
  + Introduce aleatoriedad extra: cada nodo del árbol se genera seleccionando un subconjunto aleatorio dentro de los atributos disponibles en cada momento.
  + Hiper-parámetros para definir:
    - Cantidad de clasificadores bases.
    - Porcentaje de variables a utilizar en cada split.
    - Todos los propios de un árbol (profundidad máxima, cantidad de instancias mínimas en cada nodo hoja, etc.).
  + Variantes
    - Pasting: conjuntos de datos más pequeños y generados sin reemplazamiento.
    - Random Subspaces: conjuntos de datos generados sobre un subconjunto de variables de entrada.
    - Random Patches: conjuntos de datos generados con las dos condiciones previas.
* **Boosting**
  + Método para construir el ensemble.
  + Los clasificadores bases se construyen uno después del otro de forma incremental.
  + Idea: que cada modelo se concentre más en las instancias donde el anterior falló.
  + Se inicia entrenando un clasificador base con una muestra bootstrap. A partir del segundo conjunto de datos, la probabilidad de seleccionar a cada instancia está condicionada al error del clasificador previo.
* ***AdaBoost (ADAptive BOOSTing)***
  + Algoritmo que utiliza Boosting.
  + Ensemble homogéneo y en serie. No está condicionado a un tipo en particular.
  + Los clasificadores bases deben ser lo más simples posibles.
  + Procedimiento
    - Fase de entrenamiento:
      * Inicializar los pesos de cada instancia en (siendo la cantidad de instancias).
      * Generar una muestra bootstrap utilizando los pesos de las instancias.
      * Entrenar un estimador con la muestra generada.
      * Calcular el error y actualizar los pesos de las instancias de acuerdo a ese error (a mayor error, mayor peso).
      * Repetir desde el paso 2 hasta que el error del modelo estimado esté por debajo del umbral definido.
    - Fase de predicción:
      * Utilizar todos los clasificadores bases entrenados en la fase de entrenamiento.
      * Combinar las salidas utilizando la técnica de voto por mayoría ponderado, donde el peso de cada estimador está relacionado con el error obtenido en la fase de entrenamiento.
* ***Gradient Boosting***
  + Algoritmo que utiliza Boosting.
  + Homogéneo y en serie
  + Se basa en ir ajustando clasificadores bases utilizando las salidas del modelo previo como variable a predecir.
  + Generalmente se utilizan árboles como clasificadores bases.
  + Suele dar muy buenos resultados sin tener que preprocesar demasiado los datos.
  + *xgboost* es una de las librerías más utilizadas.

Bagging vs. Boosting

Similitudes

* Combinan clasificadores bases de un mismo tipo.
* Se busca perder un poco de interpretabilidad para ganar en performance.
* Las salidas se combinan utilizando votos por mayoría (o alguna variante).
* Todos los clasificadores base se generan a partir del mismo conjunto de entrenamiento.

Diferencias:

* Bagging permite paralelizar la construcción del ensemble mientras que Boosting no.
* Bagging aísla cada modelo construido, mientras que en Boosting el conocimiento se "comparte".
* Boosting otorga pesos a los clasificadores base mientras que en Bagging todos cuentan por igual.

# Feature Engineering

Los datos pueden ser modificados para mejorar la capacidad de expresar el problema y ayudar al modelo a aprender.

Puede verse como parte de la fase de limpieza y pre procesado. También es borroso el límite con técnicas de selección de features. Diferencia:

* Preprocesado y limpieza: deja los datos “listos”.
* Feature engineering: deja datos “más expresivos”.

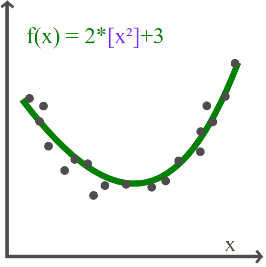
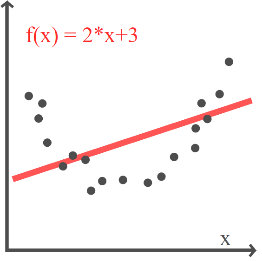
Características

* No es una ciencia exacta.
* Depende mucho del problema y de nuestra habilidad para descubrir cómo expresar mejor la información.
* En algunos modelos y problemas no es necesario (Ejemplo: redes neuronales con imágenes).
* Es un proceso iterativo: idear features, probar modelos con las features, seleccionar, volver a empezar.

Técnica 1: Features no lineales para modelos lineales

Para poder seguir usando un modelo lineal simple (Linear Regression, Logistic Regression), con datos que no son lineales.

* Se pueden agregar features que operen entre variables (Ejemplo: ).
* Super simple.
* Permite seguir usando modelos que son rápidos y simples en casos que antes no lo permitían.
* Hay que probar (qué features, a qué exponentes, etc.).



Técnica 2: Crear features extrayendo información de otras features

Encontrar columnas que contienen información escondida, y separarla en nuevas columnas.

Usos típicos:

* Momento del día.
* Solo fecha/solo hora.
* Medidas o coordenadas por separado.
* Región geográfica (extraer país, continentes, zona, etc.).
* Clase de producto/cliente/etc.

Evita que el modelo tenga que aprender a extraer esa información. Aprende más rápido.

Técnica 3: Binning/redondeo de números

Evita que el modelo sobreentrene y ayuda a facilitar la visualización de puntos que quiebre.

* Se quita ruido y se deja información.
* Puede evitar que el modelo se sobreentrene en decimales/valores aleatorios que no definen al problema.
* Evita que el modelo tenga que aprender los rangos significativos por sí solo. Aprende más rápido.
* Depende de nuestra habilidad para determinar el nivel de precisión significativo para cada feature.

Técnica 4: Features a partir de datos pasados

Aplica cuando se tienen datos ubicados en el tiempo (ejemplo: ventas con fechas). Puede darse a cada ejemplo un “resumen” de datos pasados relacionados a ese ejemplo (ejemplo: promedio de días de demora de las ventas anteriores de ese mismo vendedor).

* Cuidado: no incluir información futura. Funcionaría mucho mejor al entrenarlo, pero mucho peor en la realidad.
* Puede aportar mucha información nueva al modelo, que de otra forma sería extremadamente difícil de aprender.
* Depende de nuestra habilidad para encontrar esas relaciones entre ejemplos, no siempre es obvio.

# Computer Vision

Se ocupa de crear computadoras que tengan la capacidad de “ver”. No se trata solo de mirar, sino de obtener información a partir de imágenes.

Resulta difícil. Los mismos objetos pueden ser terriblemente distintos, pueden existir luces, sombras y particularidades en la imagen, pueden tener geometría variable, pueden deformarse.

Algunas aplicaciones reales

* Reconocimiento óptico de caracteres (OCR).
* Biometría.
* Reconstrucción 3D (tenis).
* Cine (animación gráfica, realidad aumentada).
* Fotografía.
* Robótica.
* Marketing y publicidad.
* Análisis de comportamiento.

Subcampos dentro de CV:

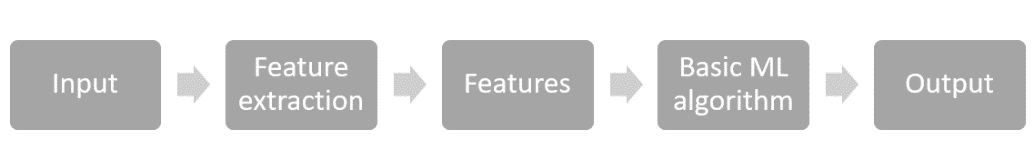
* Computer Vision: no solo con obtener imágenes sino poder realizar acciones por intermedio de su entendimiento. Relacionado con deducciones o pensamientos que obtenemos en nuestro cerebro.
* Machine Vision: cuestiones relacionadas con la adquisición de imágenes. Los "ojos" de la máquina.
* Procesamiento de imágenes: operaciones que se pueden aplicar a una imagen una vez constituida.

En muchos problemas de CV podemos hacer que una máquina aprenda de los datos:

* Reconocimiento de caras.
* Detección de objetos.
* Segmentación de imágenes.
* Reconocimiento de expresiones/emociones.
* Detección de estilos.

Tenemos dos formas principales de hacerlo para construir las entradas:

* Obtener conocimiento de la imagen usando técnicas de CV.



* Obtener una representación digital de la imagen (Matriz).



Conclusión

* CV ataca distintos problemas, y solo algunos de ellos pueden resolverse utilizando ML.
* Por distintos aspectos, hacer que una máquina tenga la capacidad de ver no es trivial.
* Si bien en las últimas décadas hubo grandes avances, queda mucho camino por recorrer.

# Procesamiento de Lenguaje Natural

Lenguaje natural: lengua o idioma hablado o escrito por humanos para propósitos generales de comunicación.

NLP: área de las ciencias de la computación y la inteligencia artificial que se ocupa de las interacciones entre máquinas y lenguajes naturales humanos. Dichas interacciones pueden ser en ambos sentidos.

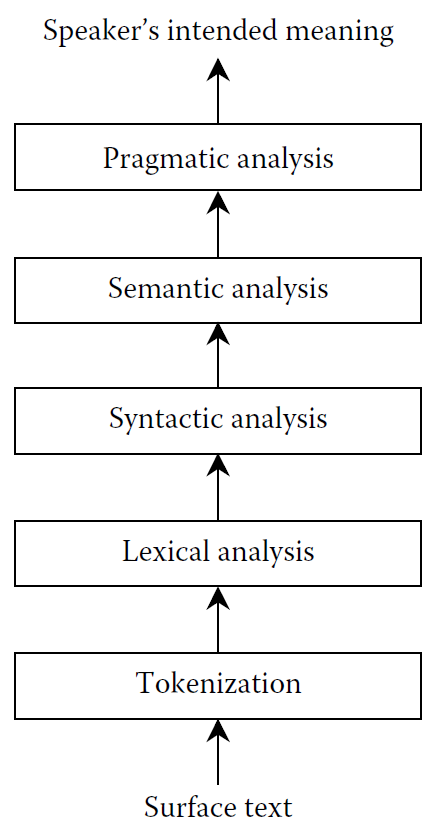
Motivación: Existe información relevante e interesa poder acceder a ella.

Fuentes estructuradas vs. fuentes desestructuradas

* Estructuradas: datos almacenados con una estructura que permite entenderlo de alguna manera. Los comúnmente utilizados: archivos de texto, etc.
* Desestructuradas: no poseen una estructura que nos permita entender los datos y, por ende, obtener información de ellos. Ejemplos: video, audio, texto.

Historia

* 1950: Alan Turing publica “Computing Machinery and Intelligence”. Test de Turing.
* 1954: Experimento Georgetown – IBM. Intento de traducción automatizada.
* 1966: Nace ELIZA. Intentaba imitar a un psicólogo, tratando de contestar como lo haría tal profesional.
* 1970s: Surgen las primeras ontologías y “chatterbots”. Primeros intentos por hacer que las máquinas entiendan lo que se le dice.
* 1980s: Se empiezan a aplicar técnicas estadísticas a la resolución de problemas de NLP.
* 2011: Watson gana el concurso de televisión Jeopardy! Sistema/producto desarrollado por IBM que tiene hardware y software propio y que es capaz de responder preguntas formuladas en lenguaje natural.
* 2011: Desafío de los esquemas de Winograd. Ejemplo: “The trophy would not fit in the Brown suitcase because it was too big”. What was too big? The trophy / the suitcase.

Problemas de los que se ocupa NLP

* Topic classification: a qué categoría pertenece un determinado texto.
* Recuperación de información: determinar información relevante de un texto.
* Question answering.
* Análisis de opiniones/sentimientos.
* Text summarization.
* Word-sense desambiguation. Uno de los problemas abiertos dentro de NLP.
* Reconocimiento de entidades nombradas (NER).
* Machine translation (interlinguas). Interlingua se pensó como un lenguaje nuevo a partir del cual se pudieran obtener fácilmente traducciones a los idiomas existentes.
* Natural Language Generation (NLG). Generar texto descriptivo a partir de una imagen, por ejemplo.
* Natural Language Understanding (NLU). Otro de los problemas abiertos de NLP. Darle a la máquina capacidad de entender y razonar.
* Speech recognition (o “voz a texto”).
* Text to speech.

Técnicas de preprocesado para datos no estructurados (lenguaje natural)

* Tokenización: separar en palabras/signos de puntuación.
* Stemming: normalización para lenguaje natural, extraer raíces de palabras.
* Part-of-Speech tagging (POS). Etiquetado de partes del discurso, determinar tipos de palabras dentro del texto.
* Lemmatization. Otra técnica de normalización, similar a stemming pero intentando hacer un análisis más completo (y no solo estadístico). Permite introducir el resultado de un etiquetado POS para realizar la operación.
* Eliminación de stop-words. Unidades que se pueden encontrar en la entrada, pero no aportan información: artículos, pronombres, preposiciones, etc. Generalmente deben ser eliminadas.

Feature Engineering en NLP

* Bag of words: columnas por cada posible palabra a encontrar. Luego, en las filas, se completa con la cantidad de veces que aparece cada palabra en cada instancia.
* Term Frequency - Inverse Document Frequency (tf-idf)
  + Intenta reflejar estadísticamente la importancia de cada término incluido dentro de una serie de documentos.
  + Term-Frequency: número de veces que un término aparece en un documento. Existen algunas variantes: tf, bool, tf log, etc.
  + Inverse Document Frequency: indica cuánta información aporta el término expresando qué tan frecuente es en todos los documentos.
* Word embedding: intenta entender el contexto de la oración, el orden o correlación de las palabras.

Problemas resolubles utilizando Machine Learning

* Topic classification
* Question answering (chatbots)
* Sentiment analysis
* Word-sense desambiguation
* Machine translation (interlingua)
* NLG
* Speech recognition

Problemas abiertos en NLP

* Recuperación de información
* Relationship extraction
* Text summarization
* NER
* NLU
* Text-to-speech

**Aprendizaje No Supervisado**

Necesitamos que un modelo encuentre estructura en datos que no están etiquetados (no tienen columnas de salida).

La estructura puede ser:

* Clustering: agrupamiento en los datos.
* Datos anómalos.
* Reducción de dimensionalidad: nuevas formas de expresar la información en menos features.

Por lo general, hablamos de problemas donde los humanos no pueden hacerlo por sí mismos (ejemplo: demasiada información).

El modelo no da una explicación de la salida, sino que debe ser interpretada por un humano.

Salida

* Clustering: datos agrupados. Opcionalmente, una función que permita seguir agrupando.
* Detección de anomalías: anomalías detectadas. Opcionalmente, una función que permita seguir detectando anomalías en nueva data.
* Reducción de dimensionalidad: data reducida más una función que permite reducir nuevos ejemplos.

**Clustering**

Se tienen muchos puntos y se desea saber cómo se agrupan.

K-means

Algoritmo sencillo que permite agrupar los datos en K grupos.

Lo logra buscando centroides que minimicen la distancia entre ellos y los ejemplos de entrenamiento.

Valores iniciales:

* Aleatorio entre rangos de los valores de las features.
* Aleatorio entre los ejemplos de los datos.

Problemas:

* Qué valor de K usar.
  + Un K pequeño puede agrupar varios grupos en uno solo. Un K grande va a diferenciar pequeños grupos dentro de un mismo grupo.
  + Para intentar determinar si existe algún problema con K, se puede evaluar la distancia promedio de los puntos a su centroide y de los centroides entre sí.
  + En algunos casos, el problema mismo puede ayudar a decidir el valor para K.
  + Para encontrar el mejor valor para K, se puede iterar. El mejor valor de K será aquel cuyos centroides tengan menor distancia promedio a los puntos de su grupo.
* Máximos locales.
  + Solución, correrlo varias veces con el mismo K (de forma que los puntos iniciales varíen) y se elige el mejor.

**Reducción de dimensionalidad**

Se tienen un montón de puntos y se busca una forma de representar esos puntos usando menos columnas.

Objetivo:

* Acelerar algoritmos de ML que usen esos datos, reduciendo sus requerimientos de hardware.
* Poder visualizar información que tiene demasiadas columnas como para ser visualizada.

Normalmente se crean nuevas dimensiones que condensan la información presente en varias columnas. Estas nuevas dimensiones no son ninguna de las columnas originales, tienen un nuevo significado.

Se pierde información. Nosotros debemos evaluar el trade-off. Algunos algoritmos permiten decidir cuánta información perder vs cuántas dimensiones reducir.

El algoritmo más utilizado es “Principal Component Analysis” (PCA).

**Detección de anomalías**

Una anomalía es un ejemplo u observación que no sigue un patrón esperado de acuerdo al resto de casos disponibles. Estadísticamente se los denomina outliers.

Aplicaciones reales

* Detección de fraude.
* Detección de intrusos.
* Mecanismos de control en procesos de manufactura.
* Monitoreo de comportamiento en data centers.
* Detección de “cheaters” en juegos.
* Detección de publicaciones de artículos prohibidos o inválidos en marketplaces.

Outliers vs. Novelty detection

* Outlier detection: se dispone de casos anómalos dentro del conjunto de datos de entrenamiento.
* Novelty detection: en los datos existentes, no existe ningún valor anómalo, pero interesa poder detectarlos.

**Outlier detection**

El objetivo es entrenar un modelo que sea capaz de discernir casos normales (inliers) de outliers.

No se plantea como problema de clasificación por el desbalanceo y porque es probable que de las pocas instancias de ejemplo no se pueda aprender demasiado.

Métodos para detectar Outliers

* **Asumiendo normalidad**. Enfoque simple y conveniente, pero tiene el problema de asumir que todas las variables son independientes entre sí y que cada una tiene un comportamiento normal.
* **Isolation Forest**: se utilizan árboles para intentar encontrar outliers dentro del conjunto de datos disponible. Se crean árboles de forma aleatoria. Luego, la probabilidad de que un determinado caso sea un outlier estará dada por la profundidad promedio en la que ese caso fue aislado en cada árbol.
* **Local Outlier Factor**: se utiliza k-NN para estimar el local outlier factor de cada instancia y así poder obtener cuáles son outliers. Por cada instancia se computan dos valores: densidad promedio de sus k vecinos (distancia promedio de los vecinos de sus vecinos, a sus vecinos) y la densidad local (distancia promedio de sus vecinos). A partir de estos valores se obtiene una ratio que indica el grado de similaridad que existe entre la densidad de muestras que tiene la instancia y la densidad que tiene cada vecino de esta. Es esperable que dicha ratio sea cercana a 1 para inliers y cercano a 0 para outliers.

**Novelty detection**

Se pueden aplicar las técnicas vistas anteriormente, pero no es factible medir la performance del modelo dado que no existen casos anómalos aún en el conjunto de datos.

Detección de anomalías como parte del preprocesado y limpieza de datos

Muchas veces nos encontramos con anomalías dentro de nuestras variables de entrada. Existen numerosos experimentos donde se demuestra que quitar estos ejemplos mejora los resultados finales.

En esos casos existen distintas acciones que se pueden llevar a cabo:

* Realizar una sustitución por valores “normales” (ejemplo: media, mediana).
* Eliminar estos casos.
* Acotar valores máximos y mínimos, centrar los datos.

# Sistemas de Recomendación

* Sistema capaz de seleccionar el *i* más apropiado dentro de un conjunto *S* de elementos posibles.
* Una de las aplicaciones más importantes de machine learning. Muy populares en los últimos años.
* Se le da mucha importancia en la industria, pero no tanto como ámbito de conocimiento.

Aplicaciones reales: Amazon, Netflix, Spotify, Facebook, Youtube, Google, Banca/seguros.

Ejemplo:

* = cantidad de usuarios/personas
* = cantidad de películas
* = 1 si el usuario i dejó su opinión sobre la película j
* = rating del usuario i sobre la película j

Feedback: ¿explícito o implícito?

* Muchas veces tenemos el feedback directamente: stars, +1, "me gusta”. Pero muchas veces no.
* Generalmente se dispone de algún tipo de feedback, explícito o implícito: si el usuario vio un video completo o no, cantidad de veces que puso pausa o volvió hacia atrás, número de veces que vio o accedió a un video, etc.

**Recomendaciones basadas en contenido**

Se basa en la idea de que dos elementos parecidos entre sí deberían recibir opiniones similares.

Por cada usuario voy a intentar aprender los parámetros de la recta que intentan replicar mejor los valores que tengo.

¿Cómo encontramos ? ¿Empezamos a probar al azar?

Función de error:

Una vez definida la función de error, ¿cómo la usaríamos para que el modelo "aprenda" por sí solo?

Descenso por el gradiente

¿Cuántas veces? ¿1 o 5?

Podemos redefinir la función de error e integrar las salidas para todos los usuarios:

**Recomendaciones basadas en filtros colaborativos**

Se basa en la idea de que dos personas que tuvieron opiniones parecidas en el pasado también las tendrán en el futuro.

Pedro: "me gustan mucho las películas de terror y poco las románticas". Dicho de otra forma, Pedro nos estaría dando su :. Si tenemos todos los vectores de parámetros:

Problemas comunes

* Cold start: ¡Utilizarlos requiere tener data histórica disponible!
* Escalabilidad: En muchos casos podemos tener cientos de miles o millones de cosas para recomendar, lo cual hace que obtener recomendaciones de forma automática y al instante sea todo un desafío.
* Datos "sparse": Generalmente tenemos data de un número mucho menor al total disponible de elementos.

Actualidad: enfoque híbridos

Existe evidencia científica de que combinar ambos modelos puede ser una buena alternativa para obtener resultados mejores que si se utilizaran de forma aislada.

Existen principalmente dos formas de combinarlos:

* Combinando las salidas de ambos.
* Creando un único modelo que combine ambos enfoques.

Un caso de ejemplo es Netflix, que realiza recomendaciones de películas a partir del historial de búsqueda y visitas de otros usuarios (filtros colaborativos) y de películas que tienen características similares (basados en contenido).

Métricas

Definir medidas de performance varía en relación a lo que se hace para un problema de clasificación común y corriente.

¿Que hagamos una recomendación y que el usuario no la haya seleccionado, significa que estaba mal?

Surgen otras métricas:

* Diversidad: mide cuánto se parecen las recomendaciones entre sí (la idea subyacente es que, a mayor diversidad, más probabilidades de acertar).
* Persistencia: mide el grado en el que, ante situaciones semejantes, el criterio de selección se mantiene consistente.
* Privacidad: generalmente existen restricciones legales sobre la privacidad de los datos de los usuarios, lo cual limita o condiciona la utilidad de un determinado sistema.
* Serendipia: Mide el grado en que una recomendación es predecible o "esperable". Cuanto menos predecible sea, más probable es que el usuario se vea interesado y más información aporta, mientras que, si es predecible, quizá no aporta demasiado.

# Aprendizaje por Refuerzo

Se necesita que un modelo aprenda a maximizar un objetivo realizando acciones en un ambiente, que pueden tener consecuencias positivas o negativas para el objetivo.

Se aplica cuando se necesita aprender a partir de los resultados de acciones que realizamos en el ambiente.

* No conocemos del todo el ambiente, necesitamos explorar para aprender qué cosas dan buenos resultados.
* O lo conocemos, pero resulta impráctico/imposible analizar todos los posibles estados.

El aprendizaje se hace “en vivo”, mientras el agente va realizando acciones en un ambiente (real o simulado).

Puede que esto suceda en “episodios” (ej: N partidas de Super Mario, independientes), o de forma continua (ej: acciones en la bolsa).

Política y resultados

Política es lo que determina qué acción tomar en un momento dado. Es, por lo general, el objetivo del aprendizaje (salvo en pasivo).

Existen diversas formas de modelarla.

* Agente basado en utilidad

Aprende una función que le dice qué tan útil es un estado. . Se debe poder predecir cada estado.

* Agente Q-Learning

Aprende una función Q que le dice qué tan útil es una acción dada en un estado dado. No necesariamente necesita saber a qué estado llegará.

* Agente reflejo

Aprende una “tabla” que mapea estados a acciones. . Se limita a problemas donde la cantidad de estados posibles es finito y no tan grande.

Nosotros nos vamos a centrar en Q-Learning.

Pasivo vs. Activo

Dependiendo de qué interesa:

* Aprendizaje por refuerzo **pasivo**: se busca estimar la utilidad (qué tan buena es) de una política.
* Aprendizaje por refuerzo **activo**: se busca crear una política que sea útil.

En abos casos, el aprendizaje se hace principalmente a partir de las recompensas que el ambiente nos otorga por nuestras acciones. Las ……..

Recompensas en el tiempo

El agente reciba las recompensas luego de cada acción realizada.

El agente actualiza su política en base a las recompensas recibidas:

* Método de Monte Carlo: al final de cada episodio, se actualizan los valores de toda la política entera, en base a la sumatoria de recompensas recibidas en cada acción. Solo es aplicable cuando tenemos episodios, no cuando es un problema continuo.
* Métodos de aprendizaje por diferencia temporal (TD). Al final de cada acción ……

¿Vale lo mismo una recompensa inmediata que una recompensa demorada?

Dos enfoques:

* Recompensas aditivas: la recompensa en un mometo dado es la suma de todas las recompensas hasta ese momento.
* Recompensas depreciativas: la recompensa en un momento dado, es la suma de todas las recompensas hasta ese momento, pero depreciando las recompensas a medida que son más tardías. El parámetro gama varía entre 0 y 1. Cuando es 1, las recompensas no pierden valor. Cuanto más cercano a 1, menos valor pierde.

Selección de acción

Sabemos qué se hará con la recompensa después de tomar la acción (se utiliza para evaluar y/o mejorar la política. ¿Pero qué acción se toma?

Si siempre se elige la que se cree mejor en base a la política, se puede estancar en máximos locales.

Tampoco es bueno elegir siempre al azar porque deja de tener sentido y se demorará demasiado en encontrar una buena política.

Exploración vs Explotación

Se necesita explorar más al inicio (aprender conocimiento), y explotar más al final (usar conocimiento aprendido).

Esto lo determina la función de exploración: función que, dado un estado y acciones disponibles en un momento dado, determina cuál se elige.

Normalmente se buscan funciones con transición suave.

…………

Selección de acción vs modelado de la política

………….

Ejemplo: Q-Learning

* Mejor con episodios.
* Para amar una política nueva.
* Política: función de valores para los pares [estado, acción a realizar].
* Actualizar la política al final de cada acción (TD).
* Recompensas depreciativas.
* La función de exploración no está definida (se puede elegir la que se quiera).

Redes Neuronales – Deep Reinforcement Learning

Muy similar a Q-Learning. En lugar de armar a mano la función Q(estado, acción), lo que hacemos es que una red neuronal sea esa función, y que se entrene a partir de las recompensas que va recibiendo. La red neuronal es la política.