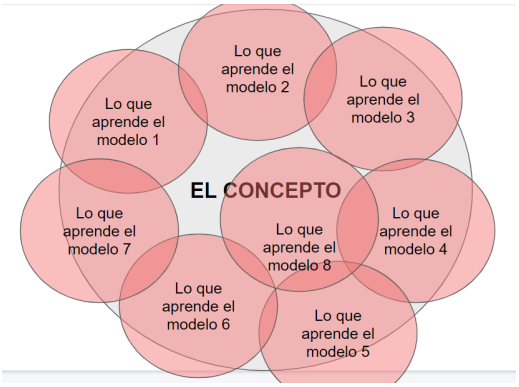
Ensambles de modelos

Cada modelo aprende algo diferente. Al tener todos un error y una parte parcial de la verdad, cada uno overfitteado de forma distinta, que vieron distintos datos. Juntos se complementan de manera diferente. Cada modelo *ajusta distinto*. Cada uno tiene una visión distinta.



Ensamble:
¿Qué hace? Entrenar varios modelos, cada uno sobre distintos datos.
¿Por qué sirve?

- Cada modelo overfitea de distinta manera (tiene bajo sesgo y alta varianza)
- Cada modelo aprende una cosa distinta del concepto.Tenes muchos modelos con alta varianza esta se anula y al juntarlos se anula el error y se tiende a aproximar mejor el concepto, queda algo más estable.

Es importante que los modelos esten poco correlacionados.

De esta manera el modelo se ajusta de manera más homogénea a todos los sets de datos. Se puede usar la probabilidad que devuelve el modelo para usar como el promedio ponderado. Esa probabilidad es la seguridad del modelo con respecto al valor que devuelve.

<u>Técnicas:</u>
Modelos homogéneos (usan el mismo algoritmo detrás)

Bagging
Consiste en construir nuevos conjuntos de entrenamiento usando bootstrap (muestras aleatorias con reemplazo, tipo extracciones con reposición) para entrenar distintos modelos, y luego combinarlos.
Todas las muestras tienen la misma cantidad de individuos. Hay que tener en cuenta que puede ser que un mismo individuo esté dos veces en el set de entrenamiento que armamos.

En datasets con alta varianza funciona mejor (datos más dispersos). Puede reducir el overfitting y también reduce el ruido. Puede mejorar el voto ponderado pero no demasiado.

1. Dividimos el set de entrenamiento en distintos subsets, obteniendo como resultado diferentes muestras aleatorias.
    - Las muestras son uniformes (misma cantidad de individuos)
    - Son muestras con reemplazo (los individuos pueden repetirse en el mismo set de datos)

2. Entrenamos un modelo con cada set

3. Construimos un único modelo predictivo a partir de los anteriores

Características:
- Disminuímos la varianza en nuestro modelo final
- Muy efectivo en datasets con alta varianza
- Puede reducir el overfitting
- Puede reducir el ruido de los outliers (porque no aparecen en todos los datasets)
- Puede mejorar levemente con el voto ponderado
- En algunos casos patológicos, el error puede aumentar

Problema con los árboles en bagging:
Si tengo atributos fuertes y yo entreno distintos árboles con distintas partes del dataset pero estas partes tienen más o menos las mismas propiedades y a veces el dataset no tiene alta varianza, va a ocurrir que **todos los árboles se van a parecer demasiado porque el arbol de decision elige siempre los mejores atributos**. Vamos a tener los mismos nodos más o menos. Esto se soluciona con random forest.

Random forest
Igual que bagging pero en cada nodo considera sólo un **subconjunto** de m atributos elegidos al azar.
Se vuelve bueno para la selección de features porque como el árbol de decisión va a elegir los atributos, ya se queda cuales son los features que bajaron la impureza en cada paso. Entonces es como bagging pero con la aleatoriedad de los atributos.

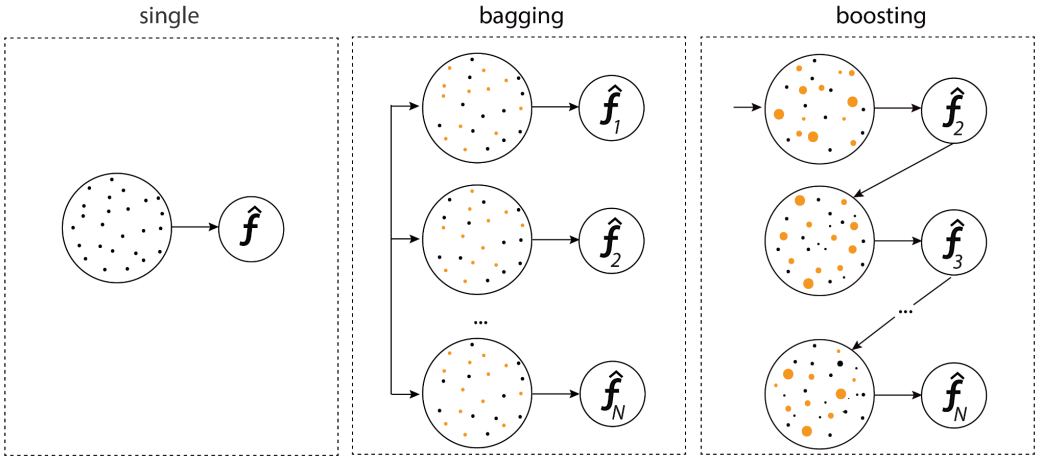
Boosting
Funciona de forma secuencial.
Comenzar con un modelo (simple) entrenado sobre todos los datos: h_0. En cada iteración i, entrenar h_i dando mayor importancia a los datos mal clasificados por las iteraciones anteriores.

A las que falla les da un peso más fuerte y entrena otro modelo. En este nuevo modelo pondera más las instancias mal clasificadas. Esto lo hace hasta que no se quiera seguir mejorando la precisión o hasta cierta cantidad de iteraciones.

Luego la votación es ponderada ya que por ejemplo el primer modelo entrenado es el más importante. Con el sesgo sucede que baja. Baja la varianza, porque estamos hablando de

un ensamble y baja el sesgo porque cada modelo mira más los datos y mira más las instancias que clasificaron mal los demás modelos, te estás contraponiendo al sesgo del modelo anterior.

Cada predicción también se pondera dependiendo de en qué punto de la cadena se encuentre.



Adaboost
Utiliza árboles de decisión. Todas las instancias arrancan con el mismo peso. A las instancias mal clasificadas se les asigna un mayor peso luego.

Gradient boosting
También utiliza árboles como base. Se empieza con un algoritmo base y se van mejorando los parámetros. Se van generando modelos en base a los errores del modelo anterior.

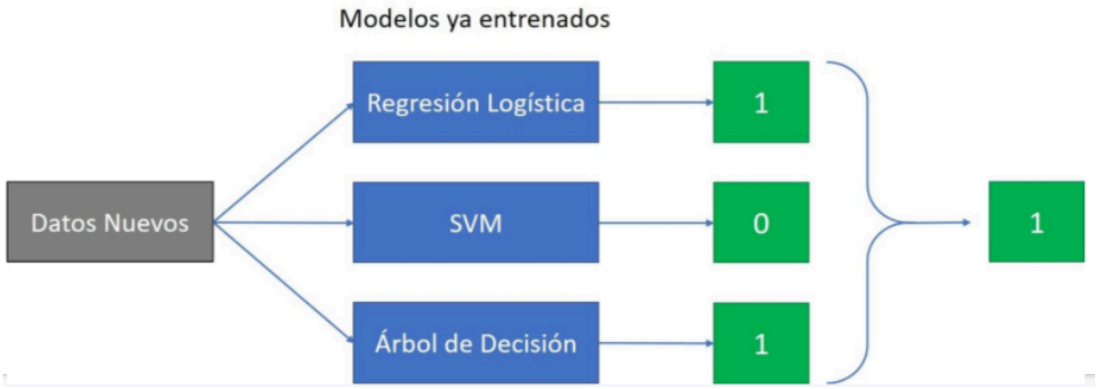
Calcula el error de predicción de cada árbol mediante una función de pérdida.

En base a la función de pérdida optimiza los parámetros de los nuevos árboles mediante descenso del gradiente.

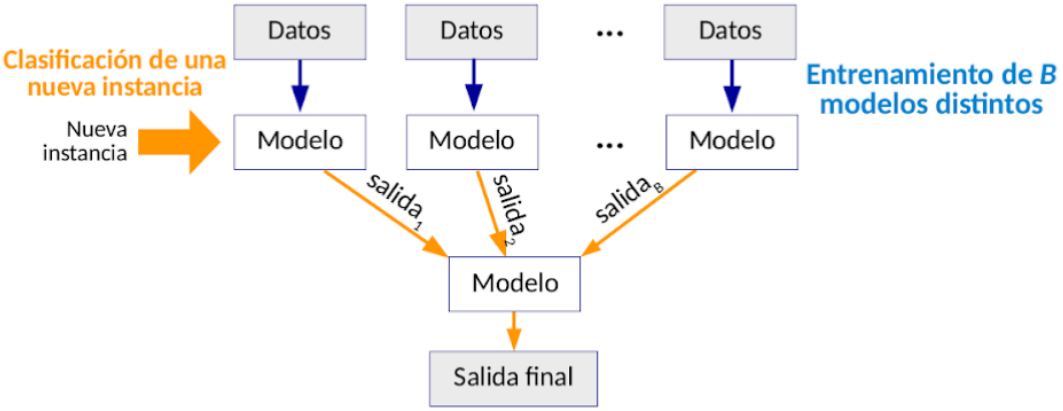
XG boost
Es una re mejora y es derivado de gradient boosting. Tiene un buen uso de GPU. Es uno de los que mejor rendimiento tiene actualmente. Es un gradient boosting mejorado. El accuracy es mejor porque porque se baja el overfitting con regularizaciones

<u>HÍBRIDOS</u>: combinan clasificadores de distinto tipo

Voting
Construimos N modelos utilizando los mismos datos y luego tomamos la decisión mayoritaria.



Stacking
Similar a voting. Tiene un modelo más que decide dada una instancia nueva, que modelo usar y te ayuda a ver a quién mira más al momento de votar.



<u>Características</u>

- Los modelos para el meta-aprendizaje suelen ser: árboles, NB, SVM o Perceptrón (lo vemos en Redes Neuronales)
- Menos popular que boosting, bagging.
- Desventajas: Dificultad de análisis teórico: caja negra, Múltiples variantes.
- Se puede interpretar como una mejora (generalización) del método de votación (Voting)
- Si los clasificadores base pueden generar medidas de certeza, suele funcionar mejor.

<u>Stacking y Voting</u>

Voting: es el más simple. Ve cual es el que mayor tiene.
Stacking: más complejo. Usa un modelo más para predecir, para ver que voto al final hacer.

Voting va a tomar una decisión trivial. Hay dos formas: toma la mayoría o puedo usar la probabilidad (que tan seguro están de la predicción tomada). Tienen que estar calibrados los modelos para esto último. No vemos una ganancia significativa de hacer voting con respecto a stacking.

Cascading

Busca minimizar el error de predicción. Quiere tener alta precisión, lo usamos cuando queremos una alta certeza.
Enfoque en el que se pasan sucesivamente los datos de un modelo a otro. Se pasan las predicciones que no se tiene tanta certeza al nuevo modelo y este se enfoca en las partes que el modelo no sabe predecir bien.
A diferencia de Stacking cada “capa” tiene un sólo modelo.

El input de cada modelo son las instancias predichas **con poca certeza** por el modelo anterior.

Overfitting: Cascadas muy profundas. En los últimos niveles de la cascada van a quedar pocos datos y los modelos tienen menos datos para aprender. En los últimos niveles están los modelos más complejos.
Construimos entonces una secuencia (o cascada) de modelos de ML.

Ventaja de ensambles:

- Mejores predicciones que un solo modelo
- Buen trade-off sesgo varianza.

Desventajas :

- Se pierde interpretabilidad
- Tiene una complejidad computacional mayor.

<u>Cascading vs boosting</u>

Se diferencia de boosting (ambos secuenciales) ya que boosting usa el mismo modelo y cascading no, y además boosting entrena cada modelo y le da más importancia a los ejemplos que la pifio el modelo anterior, en cambio con cascading se pasan al siguiente modelo las predicciones en las que no se tienen tanta confianza. Cascading tiene en cuenta la confianza que tiene cada modelo al predecir, esto se decide con un threshold que definimos nosotros. Si la predicción para una cierta clase no supera el threshold, vamos a poner como que no está seguro. El threshold es la confianza.