Evaluación y selección de modelos

Queremos saber qué tan bueno es nuestro modelo, como lo podemos comparar con otros modelos, como lo medimos.

Medir un modelo por su accuracy es una mala idea ya que el modelo podría simplemente memorizar los casos de entrenamiento y tener un accuracy del 100%.

Medir performance sobre los datos de entrenamiento tiende a sobreestimar los resultados. Esto es por el overfitting: el modelo memoriza los datos. Si está muy overfitteado no responde bien ante un cambio, performa muy mal.

En un caso puede ser bueno usar el accuracy. El baseline es algo simple, y por lo tanto no overfittea. Como sabemos que es simple, sabemos que no memoriza.

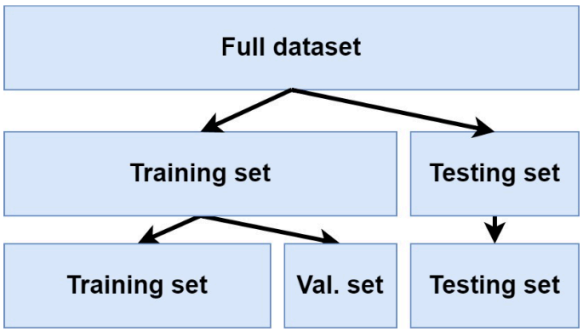
Versión mejorada de cómo evaluar un modelo

La idea es evaluar sobre datos que no se hayan usado en el entrenamiento. Para ello, nos guardamos una parte de los mismos para usarlos más adelante. (10 % a 25 %)

También se cuenta con una parte que se conoce como holdout, que es para evaluar cuando el modelo está por salir a producción.

Algo a tener en cuenta, es evitar los data leaks, esto sucede cuando parte de lo que tenes en tu parte de entrenamiento aparece en la de evaluación.

¿Cómo hacer el particionado? El particionado tiene que ser random. No debe de agarrar las primeras x instancias y listo.
Se le puede pedir cuando se está partiendo que respete la distribución de las clases. Muy útil paracuando se tienen casos muy desbalanceados (Ej. Clase positivo es el 10 % y el resto negativo)
Partición del dataset en:



Entrenamos (train), medimos (val-dev) varias veces. De forma indirecta, la información del val dev se pasa al modelo. Los hiperparametros y el procesamiento se están empezando a basar en eso.
Con el test holdout se termina de decidir si sale de producción o no.

Hay que tener cuidado cuando se dividen los datos porque puede haber data leakage de la variable target, incluso puede ocurrir cuando armamos los datos:

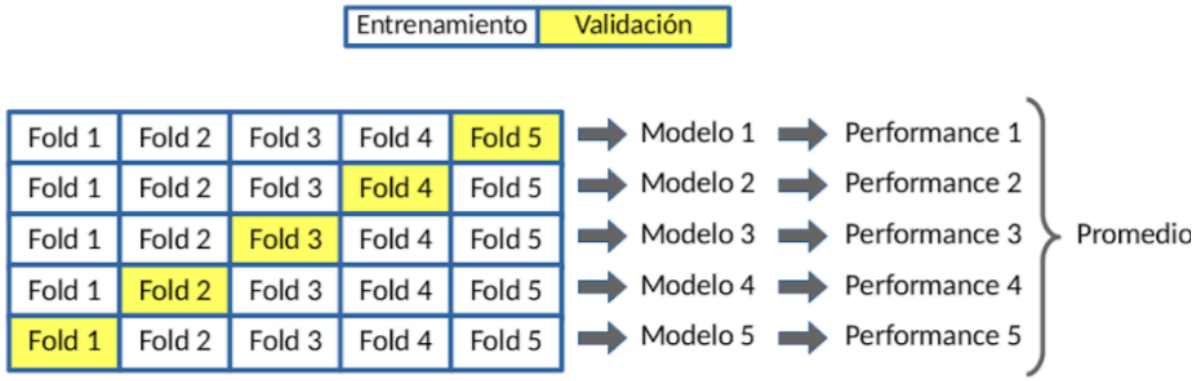
- NO quiero usar la info de val dev y test holdout para entrenar. Son como datos del futuro.

- NO tengo que usarlos para completar las filas de datos faltantes ya que genera dataleaks (se supone que el modelo no puede ver directamente nunca a val-dev o test-holdout por lo que no podemos usar dicha info para completar datos en training)

¿Qué pasa si divido mal (si el problema no te queda representativo en alguna de las dos divisiones)?

<u>Validación cruzada</u>: Se utiliza la técnica de K-fold cross validation. Disminuye el riesgo de hacer una partición mala del set de datos.

1. Se desordena el dataset.
2. Se agarran montoncitos del mismo tamaño (k montoncitos).
3. Para entrenar el modelo, voy entrenando con ciertos piloncitos, y voy rotando con cual evaluo.
4. Hago un for desde 1 hasta k, entreno sobre todos los folds menos i, y evaluo con i. Si uso k montoncitos, uso los primeros k - 1 para entrenar y el último para validar.
5. Me da k modelos entrenados con una medida de performance para cada modelo.



Tengo dos modelos comparados con K folds, ***cual es el mejor****?* Dos opciones:

Opcion 1: comparar la accuracy media (super facil).
Opción 2: método estadístico que se escapa de la materia.

¿Por qué tendríamos distintos modelos para comparar?

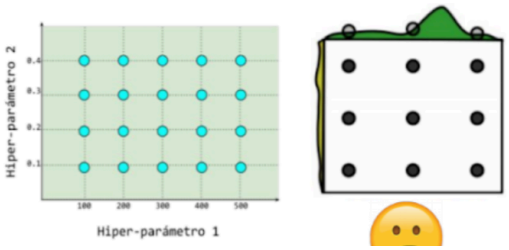
- Distintos features
- Distintos algoritmos (árboles,NB, KNN, etc)
- Distintos hiperparámetros de cada algoritmo.

Si tengo el problema del dataset desbalanceado, a la función de stratified le paso los “y” para que se mantengan las proporciones originales, de los datasets originales. Lo que queremos hacer es un stratified del fold.

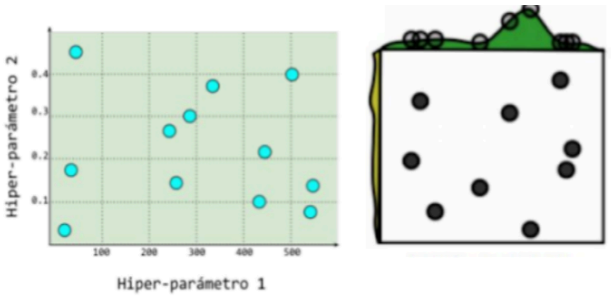
Como conclusión tenemos que explorar en el espacio de posibles combinaciones usando k folds para medir el desempeño. Finalmente con la mejor combinación entrenamos el modelo usando todos los datos.

<u>Opciones para encontrar los hiperparametros:</u>

- **Grid search:** El algoritmo consiste en ir agregando fors por cada hiper-parámetro. Escala mucho en tiempo si no se paraleliza



- **Random search**: busca de manera aleatoria. Puede mejorar o empeorar.

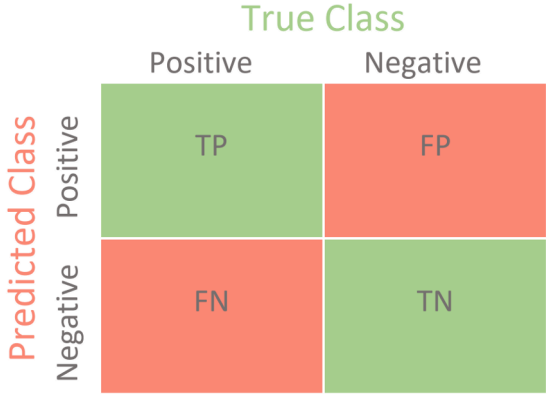


Mido los hiperparametros con el accuracy.
El accuracy puede fallar cuando la distribución de clases es desigual y cuando no me interesan todas las clases por igual.

<u>Métricas</u>

Accuracy: Evalúa directamente a cuantos le pegamos.

Matriz de confusión (binaria):

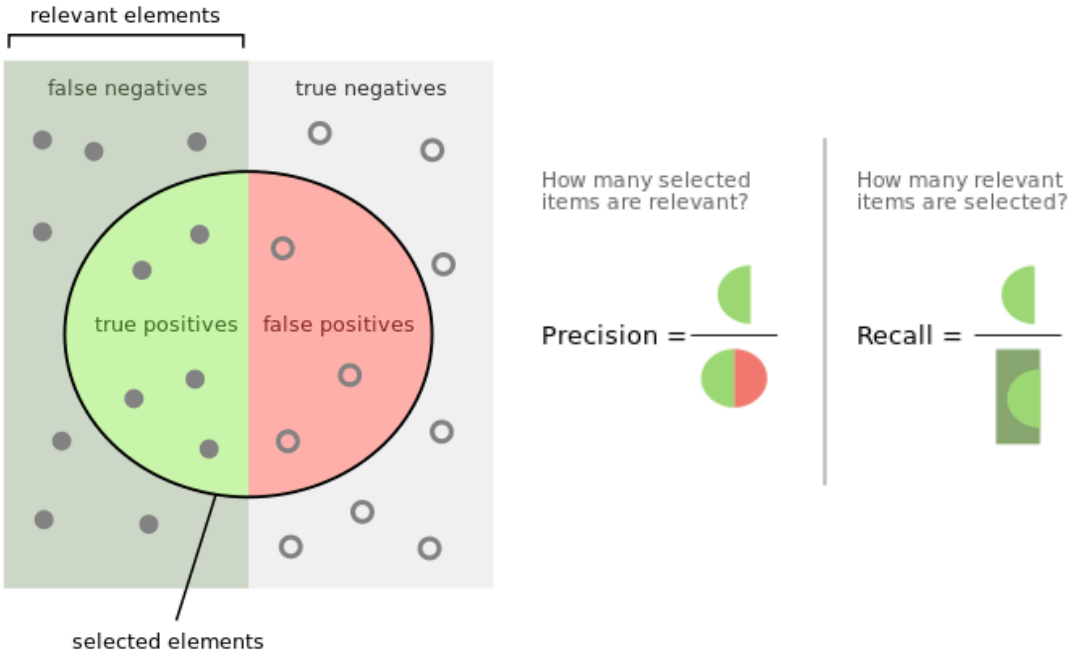


Lo de la diagonal es a lo que le pego bien.

Haciendo foco en clases que están mal clasificadas puedo mejorar mis métricas.

Precision: No le interesa si perdemos alguno, quiere saber a cuantos le pegamos de los que decimos que son. Precision = TP / (TP + FP)

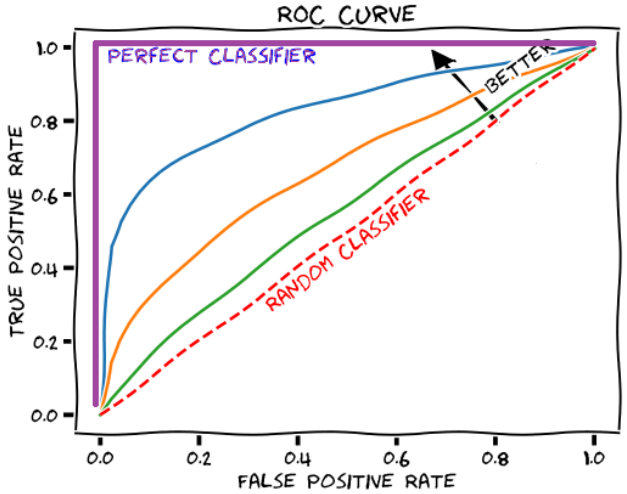
Recall: Nos importa la mayor cantidad de casos que predecimos reales. Recall = TP / (TP + FN)



F1-Score: Permite tener un único número para evaluar un modelo. Sirve para un reporte general del modelo. Se puede interpretar como una media armónica ponderada de recall y precisión.

$$F1 = \frac{2 \times (\text{precision} \times \text{recall})}{(\text{precision} + \text{recall})}$$

Curva AUC ROC: Es un gráfico que muestra que tan bueno es nuestro modelo distinguiendo clases o sea Es un gráfico que me dice de forma visual que tan buen modelo tengo



True positive rate: los correctamente clasificados como positivos dentro de todos los positivos.

$$tpr = \frac{tp}{tp + fn}$$

False positive rate: los correctamente clasificados como negativos dentro de todos los negativos.

$$fpr = \frac{fp}{fp + tn}$$

Lectura del gráfico: En el eje x tenemos el fpr y en el eje y el tpr . Se ordenan las predicciones por la probabilidad de ser positivos y se calculan los valores de cada eje. En la diagonal se muestra lo que daría un clasificador aleatorio.

AUC-ROC: Valor que surge de medir el área bajo la curva ROC.
Te da la probabilidad de que yo scoree más alto un sample positivo que un sample negativo. Me dice que tan probable es que ordene las instancias correctamente.

El valor de 1 es un clasificador perfecto porque me dice que siempre que agarre un positivo va a quedar rankeado para arriba de un negativo. Si es cero, está invirtiendo las clases.

Reporte de clasificación

macro avg: no contempla la cantidad de ejemplos de cada clase. Óptimo para los casos donde tenemos una distribución muy desigual de clases. Cuando queremos que se nos penalice mucho en caso de mala performance sin importar la cantidad de casos.

weighted avg: en el caso que me interese y pondere los ejemplos estaría bien usar esta versión. Tomo la media pesada por la ocurrencia del label. Le quiero dar más peso a la métrica que aparece más veces.

Evaluacion de calibracion
Muchas veces ademas de tener la clase de la prediccion queeremos una probabilidad de cada uno de los labels. Ahora, muchos estimadores te dan un score que no necesariamente se pueda interpretar como la probabilidad.

Un clasificador calibrado es un clasificador probabilistico para el cual la salida de predict_proba puede ser directamente interpretado como un intervalo de confianza.

Predict_proba: me tira la probabilidad de predecir bien y de predecir mal. Me dice con qué probabilidad va a predecir. Con qué “seguridad”.

¿cómo predice la clase? Se queda con la que tiene mayor probabilidad.
En vez de pasarle predict, le pasamos predict proba.