Laboratorio de Datos
Entrenamiento y testeo

Primer Cuatrimestre 2024
Turnos tarde y noche

Facultad de Ciencias Exactas y Naturales, UBA

¿Cómo elegir entre distintos modelos?

Toda tarea de «aprendizaje automático», «machine learning» o «inteligencia artificial», consiste en:

1. Tomar un problema relevante del mundo material.
2. Elegir un modelo matemático que lo represente. El modelo en general va a depender de ciertos parámetros $\beta_1, \ldots, \beta_s$. Por ejemplo en un modelo de regresión lineal, estos parámatros son los coeficientes de cada variable.
3. Definir una forma de medir qué tan bueno es un modelo, en relación a la realidad (vía los datos disponibles). Esto suele hacerse mediante una *función de pérdida*.
4. ...

¿Cómo elegir entre distintos modelos?

Función de pérdida: Si fijamos los parámetros de nuestro modelo $\beta_1, \ldots, \beta_s$ y aplicamos el modelo resultante a un conjunto de datos X, la función de pérdida mide cuánto "perdemos" utilizando el modelo en lugar de los datos reales. En regresión lineal, la función de pérdida más común es el *error cuadrático medio (MSE)*.

¿Cómo elegir entre distintos modelos?

Toda tarea de «aprendizaje automático», «machine learning» o «inteligencia artificial», consiste en:

1. Tomar un problema relevante del mundo material.
2. Elegir un modelo matemático que lo represente. El modelo en general va a depender de ciertos parámetros $\beta_1, \ldots, \beta_s$. Por ejemplo en un modelo de regresión lineal, estos parámatros son los coeficientes de cada variable.
3. Definir una una *función de pérdida*.
4. "Aprender" los coeficientes $\beta_1, \ldots, \beta_s$. Es decir, encontrar valores $\beta_1, \ldots, \beta_s$ que minimicen la pérdida.

Nivel 0: Entrenar y evaluar sobre todo el conjunto de datos

Como primer acercamiento, entrenamos nuestros modelos con un conjunto de datos X y evaluamos la perfomance sobre *los mismos datos* X.

En este contexto, si un modelo M_1 es *más complejo* que un modelo M_0 (es decir tiene más variables o parámetros), entonces necesariamente el valor de la función de pérdida será menor o igual.

Ejemplo:

- $M_0 =$ gastos mensuales $b_0 + b_1$sueldo $+ b_2$cantidad de hijos
- $M_1 =$ gastos mensuales $b_0 + b_1$sueldo $+ b_2$cantidad de hijos $+ b_3$altura

El modelo M_1 es más complejo que M_0. Podemos ver a M_0 como un caso particular de M_1 tomando $b_3 = 0$.

El valor mínimo de pérdida de M_1 es menor o igual que el valor mínimo de pérdida de M_0.

Nivel 1: Entrenar y evaluar sobre todo el conjunto de datos

Pero esto no nos asegura que el modelo M_1 es mejor que M_0 (tiene mejor capacidad predictiva).

Que un modelo alcance un error muy pequeño durante el entrenamiento, puede ser tanto por mérito propio del modelo como un síntoma de una excesiva parametrización, que le permite "interpolar" o "memorizar" los datos.

Si tenemos n observaciones y tomamos un polinomio de grado $n-1$ o un modelo con n variables (independientes), vamos a poder obtener un modelo exacto, la pérdida será 0.

Nivel 2: Separar en conjuntos de entrenamiento («train») y prueba («test»)

Para evitar el *sobreajuste* (overfitting) de los modelos, es habitual dividir el conjunto de datos en dos partes mutuamente excluyentes (y conjuntamente exhaustivas, ¡nada se tira!).

Entrenaremos cada modelo con los mismos datos de train para obtener los β óptimos, pero seleccionaremos como "mejor" a aquel modelo que minimice la pérdida sobre el conjunto de test.

Nivel 3: Split entrenamiento - validación - prueba

Como antes argumentamos que el modelo que minimiza el error de entrenamiento puede estar sobreajustándose a los datos, es igualmente posible que aquél que minimiza el error de prueba esté sobreajustándose a los datos de test: al fin y al cabo, así fue como definimos nuestra regla de selección (minimizar el error de prueba).

Para evitar este problema, se suele dividir el conjunto de datos en tres partes: entrenamiento, validación y test:

1. Entrenamos los modelos minimizando L en Xtrain (los datos de entrenamiento)
2. Seleccionamos el mejor minimizando L en Xval y
3. Evaluamos su performance "en el mundo real" con L

Nivel 4: Validación Cruzada en k Pliegos ("K-fold CV")

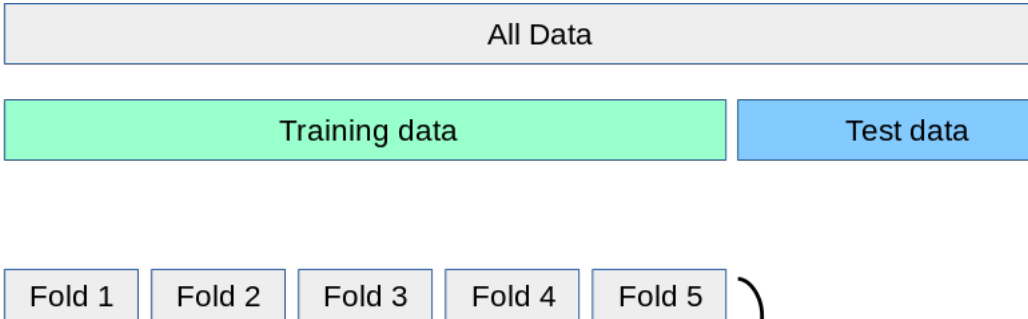
En un esquema tripartito «train-val-test», el error de test sólo sirve para reporte, y achica el tamaño efectivo de la muestra. Más aún, como hay un único conjunto de test, y todo el proceso está atravesado por ruido estocástico, la selección de modelos sigue teniendo un fuerte componente de azar.

Si queremos estimar la distribución del error de prueba Ltest de un modelo, necesitaremos de varias repeticiones del experimento. ¿Pero de dónde sacamos los datos para ello?

¡Pues los reutilizamos!

Sobreajuste

En validación cruzada de k pliegos ("k-fold cross-validation"), dividimos primero el conjunto de datos sólo en train y test. Luego, partimos train en en k partes iguales, que se rotarán el papel de validacion: entrenamos y evaluamos el modelo k veces, cada vez dejando uno distinto de los k pliegos como val y el resto para train.



Selección de modelos

¿Qué estrategias se les ocurren para ver cuál modelo es mejor?

Selección de modelos

¿Qué estrategias se les ocurren para ver cuál modelo es mejor?

1. Verificar la fórmula en otros días.

Selección de modelos

¿Qué estrategias se les ocurren para ver cuál modelo es mejor?

1. Verificar la fórmula en otros días.
2. Utilizar más días al plantear el sistema de ecuaciones.

Selección de modelos

¿Qué estrategias se les ocurren para ver cuál modelo es mejor?

1. Verificar la fórmula en otros días.
2. Utilizar más días al plantear el sistema de ecuaciones.

Las dos estretegias son ideas centrales en la construcción de modelos:

1. Probar el modelo en datos distintos a los que usamos para construir el modelo.
2. Utilizar la mayor cantidad posible de datos para construir el modelo.

1. Conjuntos de entrenamiento y testeo

Separamos nuestro conjunto de datos en dos subconjuntos:

- **Conjunto de entrenamiento.** Lo utilizamos para construir el modelo. En un modelo lineal, lo usamos para calcular los coeficientes (c_1, c_2, c_3).
- **Conjunto de testeo.** Lo utilizamos para verificar si el modelo construido ajusta bien a los datos en este conjunto.

2. Más ecuaciones que variables - Ejemplo de juguete

Si consideramos el sistema original, tenemos 5 ecuaciones y 3 variables.

$$
\begin{array}{lcccc}
 & & YPF & Santander & Nvidia \\
 & & \downarrow & \downarrow & \downarrow \\
\text{Día 1} \rightarrow & 170262.00 & = 20935c_1 & + 20100c_2 & + 37100.0c_3 \\
\text{Día 2} \rightarrow & 169929.50 & = 21030c_1 & + 20500c_2 & + 36255.0c_3 \\
\text{Día 3} \rightarrow & 171064.00 & = 20770c_1 & + 21700c_2 & + 36000.0c_3 \\
\text{Día 4} \rightarrow & 169637.35 & = 20950c_1 & + 21000c_2 & + 35645.5c_3 \\
\text{Día 5} \rightarrow & 164625.45 & = 20750c_1 & + 20316c_2 & + 33878.5c_3
\end{array}
$$

En este ejemplo (de juguete) si utilizamos los datos de Santander, el sistema tiene solución. Si usamos los datos de Galicia el sistema no tiene solución.

2. Más ecuaciones que variables - La vida real

Cuando consideramos un sistema con más ecuaciones que variables, en general **NO** tiene solución.

Aunque teóricamente exista solución, en la práctica siempre aparecen errores numéricos y no podemos determinar si un sistema tiene solución (numéricamente es MUY difícil saber si un número es igual a 0 o no).

Solución: en vez de buscar una solución exacta del sistema de ecuaciones

$$Xc = y,$$

buscamos un vector c que minimice el error, es decir, que haga pequeñas las coordenadas del vector de errores

$$Xc - y.$$

El milagro de los mínimos cuadrados

Llegamos así al método de mínimos cuadrados. El vector c que minimiza la suma de los errores al cuadrado del sistema

$$Xc = y,$$

es solución del sistema lineal de ecuaciones

$$X^T X c = X^T y.$$

Es un sistema cuadrado y en general tiene solución única.

El problema DIFICIL de minimizar los errores se transforma en el problema FÁCIL de resolver un sistema lineal de ecuaciones. **Este es el milagro de los mínimos cuadrados.**

Resumen

Tenemos dos modelos posibles y queremos elegir el más apropiado:

$$A) total = c_1 \cdot YPF + c_2 \cdot Santander + c_3 \cdot Nvidia$$
$$B) total = c_1 \cdot YPF + c_2 \cdot Galicia + c_3 \cdot Nvidia$$

Seguimos los siguientes pasos:

1. Buscamos datos de la mayor cantidad posible de días.
2. Separamos el conjunto en dos: conjunto de entrenamiento ($80\,\%$ de los días) y conjunto de testeo ($20\,\%$ restante)
3. Calculamos c_1, c_2, c_3 para cada uno de los dos modelos utilizando mínimos cuadrados en el conjunto de entrenamiento.
4. Calculamos el error cuadrático medio de lás fórmulas resultantes aplicadas al conjunto de testeo.

Modelo lineal multivariado

El caso recién visto fue un ejemplo de juguete, donde existe una relación lineal entre las variables que tenemos que estimar.

En las aplicaciones reales, esa formula puede no existir (por ejemplo, si queremos estimar los gastos en tarjeta de crédito de una persona en función del sueldo y la cantidad de hijos), pero el modelo lineal puede darnos una buena estimación.

En el modelo lineal suponemos que existe una relación del tipo

$$Xc = y.$$

Como esa relación en general no existe, buscamos un vector c que haga que Xc se parezca lo más posible a y.

Ejercicio: modelos lineales

Dadas una variable a predecir y y variables explicativas $x_1, x_2, \ldots$, ¿cuáles de los siguientes modelos son lineales? ¿Cuál es la matriz X en cada caso?

1. $y = c_0 + c_1 x_1 + c_2 x_2$
2. $y = c_0 + c_1 x_1 + c_2 x_1^2$
3. $y = c_0 + c_1 x_1 + x_1^{c_2}$
4. $y = c_0 + c_1 x_1 + c_2 x_2 + c_3 x_1 x_2$
5. $y = c_0 + c_1 \sin(x_1) + c_2 \sin(x_2)$
6. $y = c_0 + c_1 \sin(c_2 + x_1)$
7. $y = c_0 + c_1 e^{x_1}$
8. $y = c_0 \cdot c_1^{x_1}$

Ejercicio: modelos lineales

Dadas una variable a predecir y y variables explicativas $x_1, x_2, \ldots$, ¿cuáles de los siguientes modelos son lineales? ¿Cuál es la matriz X en cada caso?

1. $y = c_0 + c_1 x_1 + c_2 x_2$
2. $y = c_0 + c_1 x_1 + c_2 x_1^2$
3. $y = c_0 + c_1 x_1 + x_1^{c_2}$
4. $y = c_0 + c_1 x_1 + c_2 x_2 + c_3 x_1 x_2$
5. $y = c_0 + c_1 \sin(x_1) + c_2 \sin(x_2)$
6. $y = c_0 + c_1 \sin(c_2 + x_1)$
7. $y = c_0 + c_1 e^{x_1}$
8. $y = c_0 \cdot c_1^{x_1}$

Algunos de estos modelos se pueden linearizar, pero eso ya es otra historia...

Ejemplo real: cálculo de calorías

Vamos a repetir estos pasos en un caso real: calcular las calorías de un alimento en función de sus componentes, utilizando las herramientas de Python para la construcción de modelos.

1. Construimos las matrices X e y utilizando `Formulaic`.
2. Separamos las matrices en entrenamiento y testeo utilizando `train_test_split`
3. Ajustamos el modelo utilizando `linear_model.fit()`.
4. Calculamos el error en los datos de testeo utilizando `linear_model.predict()`.