Modelo Lineal Multivariado: Una Introducción Práctica

Fórmulas de Wilkinson y convalidación cruzada en Python

Laboratorio de Datos - Primer Cuatrimestre 2024 - FCEyN, UBA

Tabla de Contenidos

Modelo Lineal Multivariado (MLM)

MLM: M de ~~Motomami~~ Modelo

Modelar (en CD, no en la pasarela) una variable respuesta («dependiente») $y \in \mathbb{R}$ en función de ciertas variables explicativas («independientes») $(x_1, ..., x_d) \in \mathbb{R}^d$, consiste en imponerle restricciones «útiles» a la relación (función) $y = f(x_1, ..., x_d)$.

MLM: M de ~~Motomami~~ Modelo

Modelar (en CD, no en la pasarela) una variable respuesta («dependiente») $y \in \mathbb{R}$ en función de ciertas variables explicativas («independientes») $(x_1, ..., x_d) \in \mathbb{R}^d$, consiste en imponerle restricciones «útiles» a la relación (función) $y = f(x_1, ..., x_d)$.

Hemos visto modelos univariados (d=1)

$y = f(x) = \beta_0 + \beta_1 x,$

MLM: M de ~~Motomami~~ Modelo

Modelar (en CD, no en la pasarela) una variable respuesta («dependiente») $y \in \mathbb{R}$ en función de ciertas variables explicativas («independientes») $(x_1, ..., x_d) \in \mathbb{R}^d$, consiste en imponerle restricciones «útiles» a la relación (función) $y = f(x_1, ..., x_d)$.

Hemos visto modelos univariados (d=1)

$y = f(x) = \beta_0 + \beta_1 x,$

y polinomiales en una variable

$y = \text{Poli}_k(x) = \Sigma_{i=0}^{k} \beta_i x^i = \beta_0 + \beta_1 x + \beta_2 x^2 + ... + \beta_k x^k$

MLM: M de ~~Motomami~~ Modelo

Modelar (en CD, no en la pasarela) una variable respuesta («dependiente») $y \in \mathbb{R}$ en función de ciertas variables explicativas («independientes») $(x_1, ..., x_d) \in \mathbb{R}^d$, consiste en imponerle restricciones «útiles» a la relación (función) $y = f(x_1, ..., x_d)$.

Hemos visto modelos univariados (d=1)

$y = f(x) = \beta_0 + \beta_1 x,$

y polinomiales en una variable

$y = \text{Poli}_k(x) = \Sigma_{i=0}^{k} \beta_i x^i = \beta_0 + \beta_1 x + \beta_2 x^2 + ... + \beta_k x^k$

Los dos son *lineales* en x.

MLM: L de Lineal

Un modelo $y = m(x_1, ..., x_n)$ se dice *lineal* con *coeficientes* $\beta \in \mathbb{R}^n$ si puede ser expresado como una combinación linea entre unos *coeficientes* β y las x

$$
\begin{aligned}
y &= m(x_1, ..., x_n) \\
&= \sum_{i=1}^{n} \beta_i \cdot x_i \\
&= \beta_0 \cdot 1 + \beta_1 \cdot x_1 + ... + \beta_n \cdot x_n
\end{aligned}
$$

MLM: L de Lineal

Un modelo $y = m(x_1, ..., x_n)$ se dice *lineal* con *coeficientes* $\beta \in \mathbb{R}^n$ si puede ser expresado como una combinación linea entre unos *coeficientes* β y las x

$$\begin{aligned} y &= m(x_1, ..., x_n) \\ &= \sum_{i=1}^{n} \beta_i \cdot x_i \\ &= \beta_0 \cdot 1 + \beta_1 \cdot x_1 + ... + \beta_n \cdot x_n \end{aligned}$$

Esta limitación, aparentemente brutal, es más flexible de lo que parece.

M de Multivariado

En el modelo univariado $y = a + bx$, $d = 1$ (por definición), pero aún así, $n = 2$: $\beta = (a, b)$.

M de Multivariado

En el modelo univariado $y = a + bx$, $d = 1$ (por definición), pero aún así, $n = 2$: $\beta = (a, b)$.

En el polinomial, $d = 1$ y $n = k + 1$ (una cuadrática tiene 3 coeficientes)

M de Multivariado

En el modelo univariado $y = a + bx$, $d = 1$ (por definición), pero aún así, $n = 2$: $\beta = (a, b)$.

En el polinomial, $d = 1$ y $n = k + 1$ (una cuadrática tiene 3 coeficientes)

En el modelo lineal multivariado, tenemos d arbitrariamente grande, y $n \geq d$. Será de sumo interés tener una notación concisa cuando la relación entre d y n sea compleja.

Descripción de Modelos: Fórmulas de Wilkinson-Rodgers

Fórmulas de Wilkinson

Wilkinson y Rogers, dos señores ocupados en hacer cantidades de estudios de análisis de la varianza (ANOVA), deciden sentarse a elegir una *descripción simbólica* de los modelos:

Wilkinson, G. N., & Rogers, C. E. (1973). Symbolic Description of Factorial Models for Analysis of Variance. Applied Statistics, 22(3), 392 (esto no es un link a scihub)

Fórmulas de Wilkinson

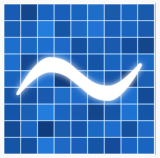
Wilkinson y Rogers, dos señores ocupados en hacer cantidades de estudios de análisis de la varianza (ANOVA), deciden sentarse a elegir una *descripción simbólica* de los modelos:

Wilkinson, G. N., & Rogers, C. E. (1973). Symbolic Description of Factorial Models for Analysis of Variance. Applied Statistics, 22(3), 392 (esto no es un link a scihub)

Se hicieron populares con S (el antecesor de R) y a hoy, todo *software* de modelo lineal implementa su propia (ligera e insidiosamente diferente) *gramática de fórmulas*.

Formulaic: Wilkinson en Python, bien rápido

En R las fórmulas son objetos de primer nivel. En python, la librería `patsy` las implementa hace tiempo ya, y en `statsmodels` se usan ampliamente. Hoy, consideraremos una implementación más reciente: `formulaic`.



Formulaic

Documentación

Gramática

Formulaic en acción

```
import pandas as pd
from formulaic import model_matrix

df = pd.DataFrame({
    'y': [0, 1, 2],
    'a': ['A', 'B', 'C'],
    'b': [0.3, 0.1, 0.2],
})
y, X = model_matrix("y ~ a + b + a:b", df)
# Esta es la versión taquigráfica (shorthand) de
# y, X = formulaic.Formula('y ~ a + b + a:b').get_model_matrix(df)
pd.concat([y, X], axis=1)
```

Formulaic en acción

```python
import pandas as pd
from formulaic import model_matrix

df = pd.DataFrame({
    'y': [0, 1, 2],
    'a': ['A', 'B', 'C'],
    'b': [0.3, 0.1, 0.2],
})
y, X = model_matrix("y ~ a + b + a:b", df)
# Esta es la versión taquigráfica (shorthand) de
# y, X = formulaic.Formula('y ~ a + b + a:b').get_model_matrix(df)
pd.concat([y, X], axis=1)
```

	y	Intercept	a[T.B]	a[T.C]	b	a[T.B]:b	a[T.C]:b
0	0	1.0	0	0	0.3	0.0	0.0
1	1	1.0	1	0	0.1	0.1	0.0
2	2	1.0	0	1	0.2	0.0	0.2

¿Gramática?

Claro! Hete aquí una breve descripción de los operadores más usuales, comunes a (imagino) todas las implementaciones de Wilkinson-Rodgers. Todos están soportados por `formulaic`:

Operador	Ejemplo	Función
~	$y \sim x$	Separa la variable (y) respuesta a la izquierda, de el/los predictor/es a la derecha (x).
+	$y \sim x + z$	**Adiciona** (suma) términos al modelo.
:	$y \sim x : z$	**Interacción** entre términos. y es lineal en $x \cdot z$.
*	$y \sim x * z$	Combina **adición e interacción**. entre términos. $y \sim x * z$ es equivalente a $y \sim x + z + x : z$

Gramática (cont.)

Existen muchos más operadores, incluyendo «–» para la **negación** de términos (para quitarlos del modelo) y «^» ó «**» para las interacciones de términos de orden *hasta* $\mathbf{n}$.

Gramática (cont.)

Existen muchos más operadores, incluyendo «–» para la **negación** de términos (para quitarlos del modelo) y «^» ó «**» para las interacciones de términos de orden *hasta* $\mathbf{n}$.

`formulaic` tiene implementadas *transformaciones* que se pueden aplicar a las variables predictoras:

- funciones arbitrarias de Python $\mathbf{y \sim mi\_fun(x)}$
- todo el módulo `numpy` disponible como `np` $\mathbf{y \sim np.log(x)}$
- algunas transformaciones propias de `formulaic`, como `center` para darle media 0 a una variable.

Gramática (cont.)

Existen muchos más operadores, incluyendo «–» para la **negación** de términos (para quitarlos del modelo) y «^» ó «**» para las interacciones de términos de orden *hasta* n.

`formulaic` tiene implementadas *transformaciones* que se pueden aplicar a las variables predictoras:

- funciones arbitrarias de Python $\mathrm{y} \sim \mathrm{mi\_fun}(\mathrm{x})$
- todo el módulo `numpy` disponible como `np` $\mathrm{y} \sim \mathrm{np.log}(\mathrm{x})$
- algunas transformaciones propias de `formulaic`, como `center` para darle media 0 a una variable.

Además, se pueden usar paréntesis $\mathrm{y} \sim (\mathrm{a} + \mathrm{b}) : \mathrm{c}$ para «agrupar términos». Se aplican las reglas de la propiedad distributiva.

¿Y el intercept?

Aunque no es *obligatorio* incluir un coeficiente constante en el modelo, es habitual hacerlo salvo contadas excepciones. En `formulaic` (y en `patsy`, y en `R`...) el intercept existe por omisión, pero se puede hacer explícito con el operador `+ 1`.

Para *quitarlo*, naturalmente, se usa «–1»: $\mathrm{y} \sim \mathrm{x} - 1$ resulta en el modelo $\mathrm{y} = \beta \cdot \mathrm{x}$.

¿Y el intercept?

Aunque no es *obligatorio* incluir un coeficiente constante en el modelo, es habitual hacerlo salvo contadas excepciones. En `formulaic` (y en `patsy`, y en `R`...) el intercept existe por omisión, pero se puede hacer explícito con el operador `+ 1`.

Para *quitarlo*, naturalmente, se usa «-1»: $\mathrm{y} \sim \mathrm{x} - 1$ resulta en el modelo $\mathrm{y} = \beta \cdot \mathrm{x}$.

En código de `R`, pueden ver la expresión «$+0$» para remover la ordenada. Es un lenguaje curioso.

Un ejemplo con pingüinos

Supongamos que queremos estudiar la relación entre la masa corporal `masa` de los pingüinos registrados en el dataset `penguins`, su `sexo`, la `isla` de nacimiento y el largo de las aletas (`aleta_mm`).

Un ejemplo con pingüinos

Supongamos que queremos estudiar la relación entre la masa corporal `masa` de los pingüinos registrados en el dataset `penguins`, su `sexo`, la `isla` de nacimiento y el largo de las aletas (`aleta_mm`).

Asumiendo que la masa es proporcional al *volumen* del pingüino, puedo imaginar una relación `y ~ poly(aleta_mm, 3)`.

Un ejemplo con pingüinos

Supongamos que queremos estudiar la relación entre la masa corporal `masa` de los pingüinos registrados en el dataset `penguins`, su `sexo`, la `isla` de nacimiento y el largo de las aletas (`aleta_mm`).

Asumiendo que la masa es proporcional al *volumen* del pingüino, puedo imaginar una relación `y ~ poly(aleta_mm, 3)`.

Si creo que además el *sexo* del pingüino influye en la relación, puedo agregarlo como `y ~ poly(aleta_mm, 3) + sexo`. ¿Pero cómo cambiaría la `masa` en relación a una variable categórica?

Todo será un(os) número(s): OHE, o *one-hot encoding*

Una manera inmediata de transformar una variable categórica con $\mathbf{k}$ categorías, es «embeberla», en un espacio de dimensión $\mathbf{k}$, e identificar cada categoría con uno de los vectores canónicos de la base.

E.g.: $\mathbf{sexo} \in \{\mathbf{macho}, \mathbf{hembra}\}$ se peude codificar en $\mathbb{R}^2$ como $(1, 0)$ y $(0, 1)$ respectivamente.

Este procedimiento se denomina *one-hot encoding*, ya que el índice de la categoría «activa/caliente» se identifica con un «1» (y los demás con «0»).

Todo será un(os) número(s): OHE, o *one-hot encoding*

Una manera inmediata de transformar una variable categórica con $\mathbf{k}$ categorías, es «embeberla», en un espacio de dimensión $\mathbf{k}$, e identificar cada categoría con uno de los vectores canónicos de la base.

E.g.: $\mathbf{sexo} \in \{\mathbf{macho}, \mathbf{hembra}\}$ se peude codificar en $\mathbb{R}^2$ como $(1, 0)$ y $(0, 1)$ respectivamente.

Este procedimiento se denomina *one-hot encoding*, ya que el índice de la categoría «activa/caliente» se identifica con un «1» (y los demás con «0»).

¿Qué riesgos conlleva OHE? ¿Son equivalentes todas las codificaciones posibles?

OHE: ¿Cómo se hace en Python?

```python
def ohe_pandas(serie):
    return pd.get_dummies(serie, prefix=serie.name).astype(int)


def ohe_gonza(serie):
    niveles = sorted(set(serie))
    ohe = pd.DataFrame({serie.name + "_" + n: (serie == n) for n in niveles})
    return ohe.astype(int)


assert all(ohe_pandas(df.a) == ohe_gonza(df.a))
ohe_gonza(df.a)
```

OHE: ¿Cómo se hace en Python?

```python
def ohe_pandas(serie):
    return pd.get_dummies(serie, prefix=serie.name).astype(int)


def ohe_gonza(serie):
    niveles = sorted(set(serie))
    ohe = pd.DataFrame({serie.name + "_" + n: (serie == n) for n in niveles})
    return ohe.astype(int)


assert all(ohe_pandas(df.a) == ohe_gonza(df.a))
ohe_gonza(df.a)
```

	a_A	a_B	a_C
0	1	0	0
1	0	1	0
2	0	0	1

Un ejemplo con pingüinos (cont.)

Por defecto, `formulaic` aplica OHE a las variables categóricas. Asumamos que el *contraste* elegido es con respecto a la categoría «hembra», así que `sexo` sera `0` si el pingüino es hembra y `1` si es macho. Usando la notación $1(\mathrm{x})$ para la *función indicadora*

$$1(\mathrm{x}) = \begin{cases} 1 \text{ si x es Verdadero} \\ 0 \text{ si x es Falso} \end{cases}$$

¿Qué representa `y ~ poly(aleta_mm, 3) + sexo`?

Un ejemplo con pingüinos (cont.)

Por defecto, `formulaic` aplica OHE a las variables categóricas. Asumamos que el *contraste* elegido es con respecto a la categoría «hembra», así que `sexo` sera `0` si el pingüino es hembra y `1` si es macho. Usando la notación $1(\mathrm{x})$ para la *función indicadora*

$$1(\mathrm{x}) = \begin{cases} 1 \text{ si x es Verdadero} \\ 0 \text{ si x es Falso} \end{cases}$$

¿Qué representa `y ~ poly(aleta_mm, 3) + sexo`?

En ese modelo, la relación entre el largo de las aletas y la masa corporal es la misma para ambos sexos, *pero* con una diferencia constante es entre sexos. ¿Por qué? ¿No sería más interesante *otro* tipo de modelo?

Un ejemplo con pingüinos (cont.)

Por defecto, `formulaic` aplica OHE a las variables categóricas. Asumamos que el *contraste* elegido es con respecto a la categoría «hembra», así que `sexo` sera `0` si el pingüino es hembra y `1` si es macho. Usando la notación $1(\mathrm{x})$ para la *función indicadora*

$$1(\mathrm{x}) = \begin{cases} 1 \text{ si x es Verdadero} \\ 0 \text{ si x es Falso} \end{cases}$$

¿Qué representa `y ~ poly(aleta_mm, 3) + sexo`?

En ese modelo, la relación entre el largo de las aletas y la masa corporal es la misma para ambos sexos, *pero* con una diferencia constante es entre sexos. ¿Por qué? ¿No sería más interesante *otro* tipo de modelo?

¿Por qué no usé dos indicadoras, una para macho y otra para hembra?

Un ejemplo con pingüinos (cont.)

¿Qué representa `y ~ poly(aleta_mm, 3) * sexo`?

Un ejemplo con pingüinos (cont.)

¿Qué representa `y ~ poly(aleta_mm, 3) * sexo`?

En este modelo, «se esconden» dos modelos con la misma *forma* funcional (un polinomio de tercer grado), pero coeficientes (potencialmente) distintos término a término.

Un ejemplo con pingüinos (cont.)

¿Qué representa `y ~ poly(aleta_mm, 3) * sexo`?

En este modelo, «se esconden» dos modelos con la misma *forma* funcional (un polinomio de tercer grado), pero coeficientes (potencialmente) distintos término a término.

¿Y si escribiese `y ~ poly(aleta_mm, 3) * sexo * isla`? ¿Cuántos términos tendría?

Un ejemplo con pingüinos (cont.)

¿Qué representa `y ~ poly(aleta_mm, 3) * sexo`?

En este modelo, «se esconden» dos modelos con la misma *forma* funcional (un polinomio de tercer grado), pero coeficientes (potencialmente) distintos término a término.

¿Y si escribiese `y ~ poly(aleta_mm, 3) * sexo * isla`? ¿Cuántos términos tendría?

¿Y cómo elijo entre todos estos modelos?

Selección de Modelos: Convalidación Cruzada

Modelos alternativos

En las slides anteriores, mencionamos al menos los siguientes modelos:

- `y ~ poly(aleta_mm, 3)`

Modelos alternativos

En las slides anteriores, mencionamos al menos los siguientes modelos:

- `y ~ poly(aleta_mm, 3)`
- `y ~ poly(aleta_mm, 3) + sexo`

Modelos alternativos

En las slides anteriores, mencionamos al menos los siguientes modelos:

- `y ~ poly(aleta_mm, 3)`
- `y ~ poly(aleta_mm, 3) + sexo`
- `y ~ poly(aleta_mm, 3) * sexo`

Modelos alternativos

En las slides anteriores, mencionamos al menos los siguientes modelos:

- `y ~ poly(aleta_mm, 3)`
- `y ~ poly(aleta_mm, 3) + sexo`
- `y ~ poly(aleta_mm, 3) * sexo`
- `y ~ poly(aleta_mm, 3) * sexo * isla`

Modelos alternativos

En las slides anteriores, mencionamos al menos los siguientes modelos:

- `y ~ poly(aleta_mm, 3)`
- `y ~ poly(aleta_mm, 3) + sexo`
- `y ~ poly(aleta_mm, 3) * sexo`
- `y ~ poly(aleta_mm, 3) * sexo * isla`

¿Considerarían algún otro? ¿Más sencillo o más complejo? Y sobre todo,

Modelos alternativos

En las slides anteriores, mencionamos al menos los siguientes modelos:

- `y ~ poly(aleta_mm, 3)`
- `y ~ poly(aleta_mm, 3) + sexo`
- `y ~ poly(aleta_mm, 3) * sexo`
- `y ~ poly(aleta_mm, 3) * sexo * isla`

¿Considerarían algún otro? ¿Más sencillo o más complejo? Y sobre todo,

¿Cómo elegir entre ellos?

Función de pérdida

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

1. Tomar un problema relevante del mundo material

Función de pérdida

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

Función de pérdida

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
3. Definir una *función de pérdida* $\mathrm{L}\left(\beta \mid \mathrm{X}\right)$ que mida de alguna manera cuán *bueno* es el modelo (a través de β) en relación a la realidad (vía los datos X).
   - En regresión, la pérdida más común es el *error cuadrático medio* (MSE)

Función de pérdida

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
3. Definir una *función de pérdida* $\mathrm{L}\left(\beta \mid \mathrm{X}\right)$ que mida de alguna manera cuán *bueno* es el modelo (a través de β) en relación a la realidad (vía los datos X).
   - En regresión, la pérdida más común es el *error cuadrático medio* (MSE)
4. «Aprender» los coeficientes β, es decir, encontrar β^* que minimiza L.

Función de pérdida

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
3. Definir una *función de pérdida* $\mathrm{L}\left(\beta \mid \mathrm{X}\right)$ que mida de alguna manera cuán *bueno* es el modelo (a través de β) en relación a la realidad (vía los datos X).
   - En regresión, la pérdida más común es el *error cuadrático medio* (MSE)
4. «Aprender» los coeficientes β, es decir, encontrar β^* que minimiza L.

$$\beta^* = \operatorname{argmin}_\beta \mathrm{L}\left(\beta \mid \mathrm{X}\right)$$

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

Hasta ahora, hemos entrenado nuestros modelos con un conjunto de datos X, y evaluado la performance *sobre los mismos datos* de entrenamiento. En este contexto, si un modelo M_1 con coeficientes β_1 es tan o más complejo[0] que otro M_0 (notemos $\mathrm{M}_0 \subseteq \mathrm{M}_1$) entonces *necesariamente* $\mathrm{L}\left(\beta_1 | \, \mathrm{X}\right) \leq \mathrm{L}\left(\beta_0 \mid \mathrm{X}\right)$. ¿Por qué?

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

Hasta ahora, hemos entrenado nuestros modelos con un conjunto de datos X, y evaluado la performance *sobre los mismos datos* de entrenamiento. En este contexto, si un modelo M_1 con coeficientes β_1 es tan o más complejo[0] que otro M_0 (notemos $\mathrm{M}_0 \subseteq \mathrm{M}_1$) entonces *necesariamente* $\mathrm{L}\left(\beta_1 | \, \mathrm{X}\right) \leq \mathrm{L}\left(\beta_0 \mid \mathrm{X}\right)$. ¿Por qué?

[0]: Hay muchas maneras de definir «complejidad»: por ahora, la identificaremos con el número de coeficientes del modelo (la *dimensión* de su vector β).

Nivel 1: Separar en conjuntos de entrenamiento («train») y prueba («test»)

Que un modelo alcance un error muy pequeño durante el entrenamiento, puede ser tanto por mérito propio del modelo, o síntoma de una excesiva parametrización, que le permite «interpolar» o «memorizar» los datos (e.g.: si tenemos $\mathbf{n}$ observaciones $\left(\mathrm{y_i}, \mathrm{x_i}\right)_{\mathrm{i=1}}^{\mathbf{n}}$, existe un polinomio P de grado $\mathbf{n}$, tal que $\mathrm{L}\left(\mathrm{P|X}\right) = \mathrm{ECM}(\mathrm{P}(\mathrm{X}), \mathrm{X}) = 0$.

Nivel 1: Separar en conjuntos de entrenamiento («train») y prueba («test»)

Que un modelo alcance un error muy pequeño durante el entrenamiento, puede ser tanto por mérito propio del modelo, o síntoma de una excesiva parametrización, que le permite «interpolar» o «memorizar» los datos (e.g.: si tenemos $\mathbf{n}$ observaciones $\left(\mathrm{y_i}, \mathrm{x_i}\right)_{\mathrm{i=1}}^{\mathrm{n}}$, existe un polinomio P de grado $\mathbf{n}$, tal que $\mathrm{L}\Big(\mathrm{P|X}\Big) = \mathrm{ECM}(\mathrm{P(X)}, \mathrm{X}) = 0$.

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 aquél modelo que minimice L sobre el conjunto de *test*.

Nivel 1: Separar en conjuntos de entrenamiento («train») y prueba («test»)

Que un modelo alcance un error muy pequeño durante el entrenamiento, puede ser tanto por mérito propio del modelo, o síntoma de una excesiva parametrización, que le permite «interpolar» o «memorizar» los datos (e.g.: si tenemos $\mathbf{n}$ observaciones $\left(\mathrm{y_i}, \mathrm{x_i}\right)_{\mathrm{i=1}}^{\mathrm{n}}$, existe un polinomio P de grado $\mathbf{n}$, tal que $\mathrm{L}\Big(\mathrm{P|X}) = \mathrm{ECM(P(X), X)} = 0$.

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 aquél modelo que minimice L sobre el conjunto de *test*.

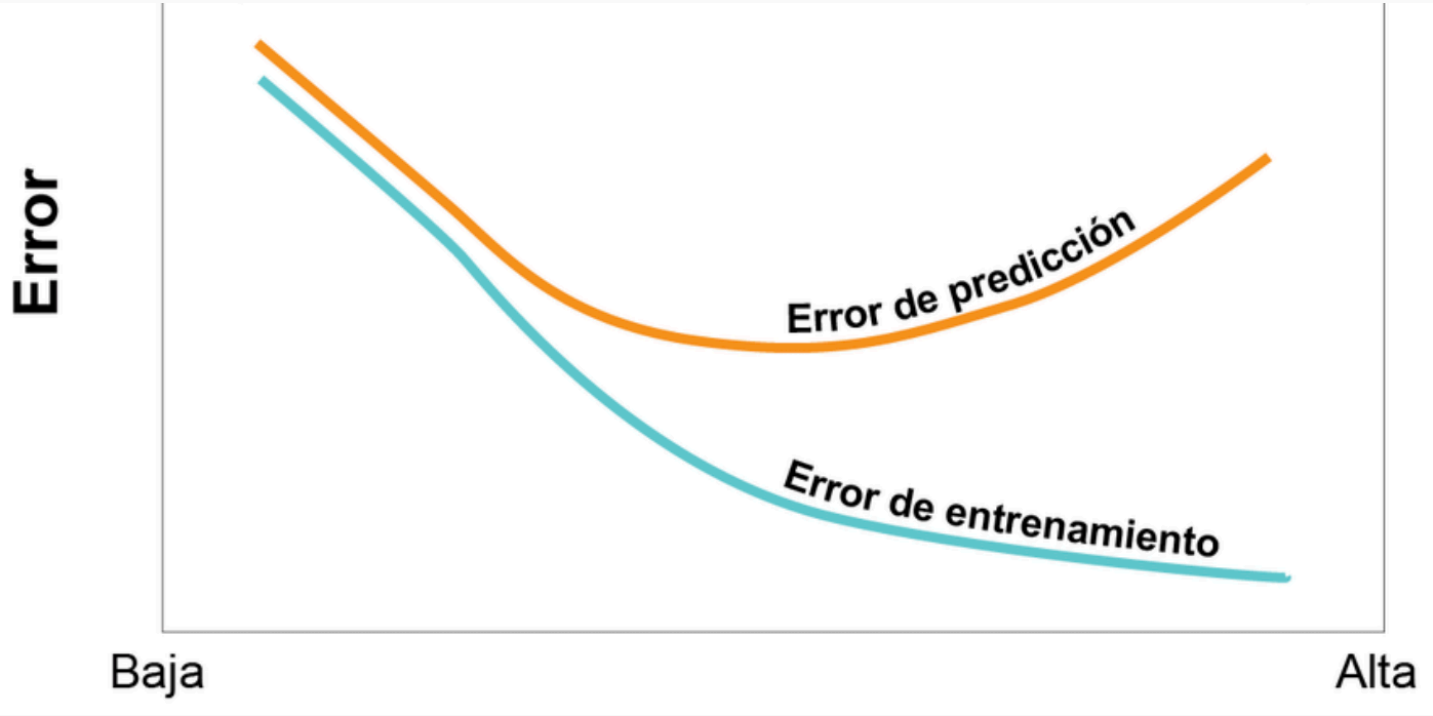
La partición habitual (el «train-test split») suele ser de 80% train, 20% test (o 70/30), pero todo dependerá del contexto y el cantidad de observaciones $\mathbf{n}$ disponible.

Relación entre error de entrenamiento y prueba

¿Cómo esperan que se relacione el error de entrenamiento con el de prueba?

Relación entre error de entrenamiento y prueba

¿Cómo esperan que se relacione el error de entrenamiento con el de prueba?



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).

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*:

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*:

- Entrenamos los modelos minimizando L en $\mathrm{X}_{\text{train}}$ (los datos de entrenamiento)
- seleccionamos el mejor minimizando L en X_{val} y
- *evaluamos* su performance «en el mundo real» con $\mathrm{L}\left(\beta \mid \mathrm{X}_{\text{test}}\right)$.

¿Qué ventajas y desventajas empíricas tiene este enfoque?

Nivel ~~GOD~~ 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.

Nivel ~~GOD~~ 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 $\mathrm{L}_{\mathrm{test}}$ de un modelo, necesitaremos de varias *repeticiones* del experimento. ¿Pero de dónde sacamos los datos para ello?

Nivel ~~GOD~~ 4: Validación Cruzada en k Pliegos («K-fold CV»)

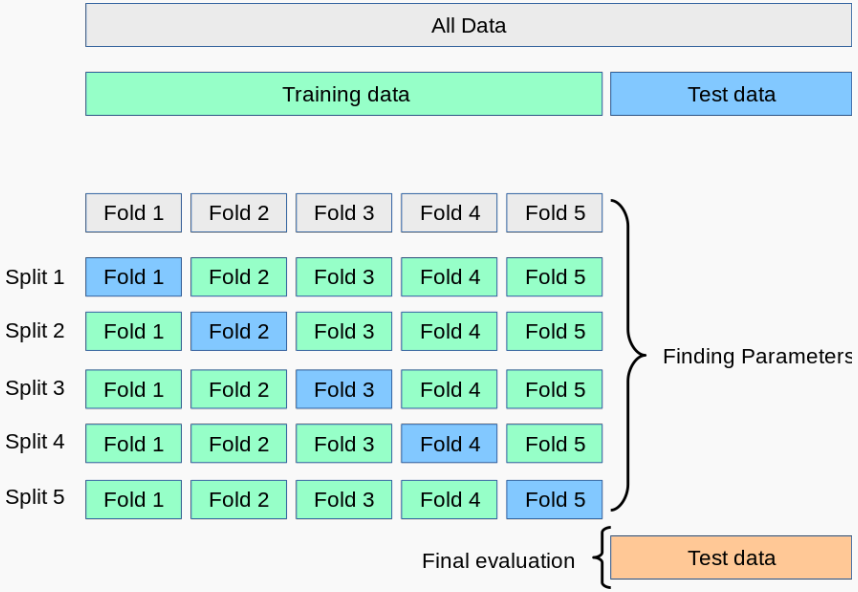
En un esquema tripartito «train-val-test», el error de `test` sólo sirve para reporte, y achica el tamaño efectivo de la muestra. Más aún, como hay un único conjunto de `test`, y todo el proceso está atravesado por ruido estocástico, la selección de modelos sigue teniendo un fuerte componente de azar.

Si queremos *estimar la distribución* del error de prueba $\mathrm{L}_{\mathrm{test}}$ de un modelo, necesitaremos de varias *repeticiones* del experimento. ¿Pero de dónde sacamos los datos para ello?

¡Pues los reutilizamos!

K-fold CV, una representación gráfica

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



Mas allá: Otros tipos de CV, y aplicación en Python

- K-fold CV no es la única manera de hacer CV. Existen variantes como *leave-one-out* (LOO), *stratified* y métodos «progresivos» para series de tiempo, entre otros.

Mas allá: Otros tipos de CV, y aplicación en Python

- K-fold CV no es la única manera de hacer CV. Existen variantes como *leave-one-out* (LOO), *stratified* y métodos «progresivos» para series de tiempo, entre otros.
- Como cuando aprendimos a multiplicar, mejor empezar haciéndolo «a mano», sin pasar por implementaciones preexistentes. Luego, scikit-learn tiene implementaciones de CV para todos los gustos.

¡Gracias!