Por: Meyers-Angulo, JE., PhD

Preprocesamiento y Validación en R

Guía para evitar data leakage en modelos predictivos

1. El Concepto Fundamental

Cuando construimos modelos predictivos, simulamos un escenario real: entrenar con datos históricos y predecir sobre datos futuros que aún no hemos visto. Esta separación es fundamental para obtener estimaciones honestas del rendimiento del modelo.

> **Regla de oro:** Los conjuntos de validación y test no pueden usarse para "aprender" ningún parámetro. Ni medias para imputar, ni escalas, ni cargas de PCA, ni coeficientes del modelo. Todo se aprende exclusivamente con train.

Sin embargo, cuando evaluamos en validación o test, los datos sí deben pasar por las mismas transformaciones que aplicamos a train: mismo escalado, mismo PCA, misma imputación. La diferencia crucial es que *aplicamos transformaciones usando parámetros ya calculados*, sin recalcular nada.

2. Los Tres Conjuntos: Train, Validation y Test

2.1. Rol de cada conjunto

Conjunto	Propósito	Se usa para...
Train	Aprender parámetros y ajustar modelos	Calcular medias, SD, cargas PCA, coeficientes de regresión
Validation	Seleccionar hiperparámetros	Elegir nº de componentes PCA, valor de lambda, comparar modelos
Test	Evaluación final imparcial	Calcular métricas finales UNA SOLA VEZ

2.2. Proporciones recomendadas

Una división típica es:

- **60% Train / 20% Validation / 20% Test**
- Alternativa: 70% Train / 15% Validation / 15% Test

La división debe hacerse *aleatoriamente* pero con una semilla fija (set.seed) para reproducibilidad.

2.3. ¿Por qué tres conjuntos y no dos?

Si usamos el conjunto de test para elegir hiperparámetros (como el número de componentes o el valor de lambda), estamos *"contaminando"* test con decisiones de modelado. Las métricas finales ya no serían imparciales.

El conjunto de *validation* actúa como un "test intermedio" que podemos usar múltiples veces para tomar decisiones. El *test* real se reserva para el final, cuando ya no haremos más cambios.

3. Flujo de Trabajo Correcto

3.1. Orden de las operaciones

1. Cargar los datos completos
2. Análisis Exploratorio (EDA), con todo el dataset
   - Entender tipos de variables y su significado
   - Visualizar distribuciones, correlaciones, outliers
   - Identificar valores faltantes y su patrón
3. Dividir en train/validation/test (60-20-20)
4. Preprocesamiento: parámetros calculados *solo* con train
5. Aplicar transformaciones a train, validation y test usando parámetros de train
6. Ajustar modelos con train
7. Seleccionar hiperparámetros evaluando en validation
8. Evaluación final: una única vez en test

3.2. ¿Por qué el EDA se hace antes de dividir?

El EDA no "aprende parámetros" que luego se apliquen a nuevos datos. Es una fase de comprensión del problema. No estamos calculando valores para transformar datos; estamos entendiendo con qué trabajamos. Por tanto, no hay riesgo de data leakage.

3.3. ¿Por qué el preprocesamiento solo con train?

Imaginemos un escenario real: hemos entrenado un modelo y mañana llega un nuevo cliente. No podemos "recalcular la media" incluyendo ese cliente porque aún no lo teníamos. Solo podemos aplicar las transformaciones que aprendimos con los datos históricos.

4. Preprocesamiento Paso a Paso

4.1. Manejo de valores faltantes

Opciones válidas:

- **Imputar con mediana:** Calcular la mediana de cada variable *solo en train*. Guardar esas medianas. Aplicarlas a train, validation y test.
- **Eliminar variables:** Si una variable tiene más del 50% de valores faltantes, considerar eliminarla.
- **Indicador de missingness:** Crear una variable binaria que indique si el valor original era faltante.

Concepto: Las medianas se calculan una sola vez con train. Si en validation o test aparece un NA, se imputa con la mediana que ya teníamos guardada de train.

4.2. Escalado (estandarización)

La estandarización transforma las variables para que tengan media 0 y desviación estándar 1. Es necesaria para PCA, Ridge y LASSO.

Procedimiento correcto:

- Calcular media y desviación estándar de cada variable solo con train
- Guardar estos valores (son los "parámetros aprendidos")
- Aplicar la transformación a train: (x - media_train) / sd_train
- Aplicar la *misma* transformación a validation y test

Error común: Usar scale() sobre todo el dataset antes de dividir. Esto contamina train con información de validation y test.

Nota sobre glmnet: El paquete glmnet estandariza internamente por defecto. Si ya escalaste manualmente, usa standardize = FALSE.

4.3. Variables categóricas

PCA y los métodos de regularización requieren variables numéricas. Opciones:

- **Opción A:** Trabajar solo con las variables numéricas del dataset (más simple).
- **Opción B:** Convertir categóricas a dummies con model.matrix(). Los niveles deben definirse con train.

5. Análisis de Componentes Principales (PCA)

5.1. Procedimiento en R

Usar la función prcomp() del paquete base:

1. Ajustar PCA solo con los datos de train, usando scale = TRUE.
2. El objeto resultante contiene las cargas (rotation), desviaciones (sdev), y scores de train (x).
3. Para proyectar validation y test, usar predict() pasando el objeto PCA.

Importante: predict() aplica a validation y test el mismo centrado, escalado y rotación que se aprendieron con train.

5.2. Selección del número de componentes

Esta decisión se toma **usando validation, nunca test**. Criterios habituales:

- **Varianza explicada acumulada:** Retener componentes hasta 80-90% de varianza.
- **Criterio del codo (scree plot):** Buscar el "codo" en el gráfico.
- **Error en validation:** Probar diferentes números de componentes, ajustar modelo en train, evaluar en validation.

Procedimiento para elegir nº de componentes:

1. Ajustar PCA con train
2. Para k = 1, 2, 3, ... componentes:
   - Extraer los primeros k scores de train
   - Ajustar regresión lineal con esos k componentes
   - Proyectar validation y predecir
   - Calcular error (RMSE) en validation
3. Elegir el k que minimiza el error en validation

6. Regularización: LASSO y Ridge

6.1. Conceptos básicos

Ambos métodos añaden una penalización para controlar la complejidad:

- **Ridge (α = 0):** Penaliza suma de coeficientes al cuadrado. Encoge pero no hace cero. Útil con multicolinealidad.
- **LASSO (α = 1):** Penaliza suma de valores absolutos. Puede hacer coeficientes exactamente cero (selección de variables).

6.2. Selección de lambda con validation

El hiperparámetro lambda controla la intensidad de la penalización. Se elige evaluando en validation:

1. Ajustar glmnet con train para una secuencia de lambdas
2. Para cada lambda, predecir en validation
3. Calcular error en validation para cada lambda
4. Elegir el lambda que minimiza el error en validation

Alternativa con cv.glmnet: Si prefieres usar validación cruzada en lugar de un conjunto de validation separado, puedes usar cv.glmnet() aplicándolo solo sobre train. La función hace CV internamente para elegir lambda.

6.3. lambda.min vs lambda.1se

- **lambda.min:** Minimiza el error. Modelo más complejo.
- **lambda.1se:** Lambda más grande cuyo error está dentro de 1 SE del mínimo. Modelo más simple.

**Documentar cuál se eligió y por qué.

7. Combinación PCA + LASSO/Ridge

Esta combinación es más un ejercicio didáctico que una práctica habitual:

- Los componentes principales ya son **ortogonales**, por lo que Ridge aporta poco.
- LASSO puede servir para "seleccionar" solo los primeros componentes relevantes.

Procedimiento:

1. Realizar PCA solo con train
2. Extraer los scores de train
3. Ajustar LASSO/Ridge sobre esos scores
4. Seleccionar lambda usando validation
5. Para test: primero proyectar con predict(pca), luego predecir con el modelo

8. Errores Comunes a Evitar

Error	Consecuencia
Escalar con todo el dataset antes de dividir	Métricas optimistas, no reflejan rendimiento real
Hacer PCA con todo el dataset	Las cargas incorporan información de validation y test
Elegir nº de componentes o lambda mirando test	Sobreajuste al conjunto de test; métricas sesgadas
Olvidar predict() para proyectar validation/test en PCA	Se usan rotaciones diferentes, resultados inválidos
No fijar semilla (set.seed)	Resultados no reproducibles
Evaluar en test múltiples veces durante el desarrollo	Test deja de ser imparcial; usar validation para esto

9. Comparación de Modelos y Reporte

9.1. Métricas de evaluación

Para regresión:

- **RMSE:** Penaliza más los errores grandes. En unidades de la respuesta.
- **MAE:** Menos sensible a outliers. En unidades originales.
- **R^2:** Proporción de varianza explicada (0 a 1).

9.2. Estructura sugerida del informe

- **Preprocesamiento:** Justificar decisiones
- **EDA:** Visualizaciones clave
- **Modelos:** Procedimiento y selección de hiperparámetros (usando validation)
- **Comparación:** Tabla con métricas en validation para elegir modelo
- **Evaluación final:** Métricas del modelo seleccionado en test
- **Conclusiones:** ¿Qué modelo recomiendan y por qué?

10. Resumen: Lo Que Siempre Debe Cumplirse

Checklist final:

☐ Los datos se dividieron en train/validation/test **antes** de preprocesar
☐ Las medianas para imputar se calcularon **solo con train**
☐ El escalado usó media y SD **solo de train**
☐ PCA se ajustó **solo con train**; validation y test se proyectaron con predict()
☐ El número de componentes se eligió **usando validation**
☐ Lambda (LASSO/Ridge) se eligió **usando validation**
☐ Las métricas finales se calcularon en test **una sola vez**
☐ Se usó set.seed() para reproducibilidad