Técnicas de Remuestreo y Selección del Mejor Modelo

Juan F. Pérez

Departamento MACC
Matemáticas Aplicadas y Ciencias de la Computación
Universidad del Rosario

juanferna.perez@urosario.edu.co

2018

Contenidos

Introducción

Introducción

- Una muestra de datos
- Entrenamiento del modelo
- Validar su desempeño
- El modelo se ajusta muy bien a datos de entrenamiento
- Desempeño: comportamiento sobre datos que no ha observado

Ejemplo

```
from sklearn.datasets import load_iris
iris = load_iris()

X = iris.data
y = iris.target

from sklearn.neighbors import KNeighborsClassifier
modelo = KNeighborsClassifier(n_neighbors=1)

modelo.fit(X, y)
y_modelo = modelo.predict(X)

from sklearn.metrics import accuracy_score
accuracy_score(y, y_modelo)
```

Ejemplo

- ¿Qué hace este código?

Ejemplo

- ¿Qué hace este código?
- ¿Resultado?

Ejemplo

- ¿Qué hace este código?
- ¿Resultado?
- ¿Problemas?

Validación simple

Validación simple

Separar datos en conjuntos de entrenamiento y prueba

Validación simple

Separar datos en conjuntos de entrenamiento y prueba

Entrenar modelos usando SOLO los datos de entrenamiento

Validación simple

Separar datos en conjuntos de entrenamiento y prueba

Entrenar modelos usando SOLO los datos de entrenamiento

Reservar datos para probar el modelo

Validación simple

```
from sklearn.datasets import load_iris
iris = load_iris()

X = iris.data
y = iris.target

from sklearn.neighbors import KNeighborsClassifier
modelo = KNeighborsClassifier(n_neighbors=1)
```

Validación simple

```
from sklearn.cross_validation import train_test_split

X1, X2, y1, y2 = train_test_split(X, y,
    random_state=0, train_size=0.5)

modelo.fit(X1, y1)

y2_modelo = modelo.predict(X2)

from sklearn.metrics import accuracy_score
accuracy_score(y2, y2_modelo)
```

Validación simple

¿Resultado?

Validación simple

¿Resultado?

¿Precisión?

Validación simple

¿Resultado?

¿Precisión?

Verdadera validación del desempeño del método de aprendizaje

Validación simple

¿Resultado?

¿Precisión?

Verdadera validación del desempeño del método de aprendizaje

Limitaciones

Validación simple

¿Resultado?

¿Precisión?

Verdadera validación del desempeño del método de aprendizaje

Limitaciones

- Reducción en el conjunto de entrenamiento

Validación simple

¿Resultado?

¿Precisión?

Verdadera validación del desempeño del método de aprendizaje

Limitaciones

- Reducción en el conjunto de entrenamiento
- ¿Qué tan bueno es el conjunto de datos seleccionado para entrenamiento?

Validación dejando una observación fuera (LOOCV)

Validación dejando una observación fuera (LOOCV)

- Dejar el mayor número de muestras disponibles para el entrenamiento

Validación dejando una observación fuera (LOOCV)

- Dejar el mayor número de muestras disponibles para el entrenamiento
- Si hay n muestras en disponibles, dejar $n - 1$ para entrenamiento

Validación dejando una observación fuera (LOOCV)

- Dejar el mayor número de muestras disponibles para el entrenamiento
- Si hay n muestras en disponibles, dejar $n - 1$ para entrenamiento
- Queda una observación para prueba

Validación dejando una observación fuera (LOOCV)

- Dejar el mayor número de muestras disponibles para el entrenamiento
- Si hay n muestras en disponibles, dejar $n - 1$ para entrenamiento
- Queda una observación para prueba
- Repetir con cada muestra, dejando solo una para prueba a la vez

Validación dejando una observación fuera (LOOCV)

- Dejar el mayor número de muestras disponibles para el entrenamiento
- Si hay n muestras en disponibles, dejar $n - 1$ para entrenamiento
- Queda una observación para prueba
- Repetir con cada muestra, dejando solo una para prueba a la vez
- E_i: error al dejar por fuera la i-ésima observación

Validación dejando una observación fuera (LOOCV)

- Dejar el mayor número de muestras disponibles para el entrenamiento
- Si hay n muestras en disponibles, dejar $n-1$ para entrenamiento
- Queda una observación para prueba
- Repetir con cada muestra, dejando solo una para prueba a la vez
- E_i: error al dejar por fuera la i-ésima observación
- El error total será

$$E = \frac{1}{n}\sum_{i=1}^{n} E_i$$

Validación dejando una observación fuera (LOOCV)

```
from sklearn.datasets import load_iris
iris = load_iris()

X = iris.data
y = iris.target

from sklearn.neighbors import KNeighborsClassifier
modelo = KNeighborsClassifier(n_neighbors=1)
```

Validación dejando una observación fuera (LOOCV)

```
import numpy as np
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import LeaveOneOut
np.set_printoptions(threshold=np.nan)

scores = cross_val_score(modelo, X, y, cv=LeaveOneOut(
print(scores)
print(scores.mean())
```

Validación dejando una observación fuera (LOOCV)

- ¿Resultados?

Validación dejando una observación fuera (LOOCV)

- ¿Resultados?
- Máximo número de muestras para ajuste, manteniendo (una) muestras para prueba

Validación dejando una observación fuera (LOOCV)

- ¿Resultados?
- Máximo número de muestras para ajuste, manteniendo (una) muestras para prueba
- Repite ajuste n veces

Validación dejando una observación fuera (LOOCV)

- ¿Resultados?
- Máximo número de muestras para ajuste, manteniendo (una) muestras para prueba
- Repite ajuste n veces
- Puede ser costoso

Validación cruzada

Validación cruzada

- Punto intermedio entre validación simple y dejar una observación fuera

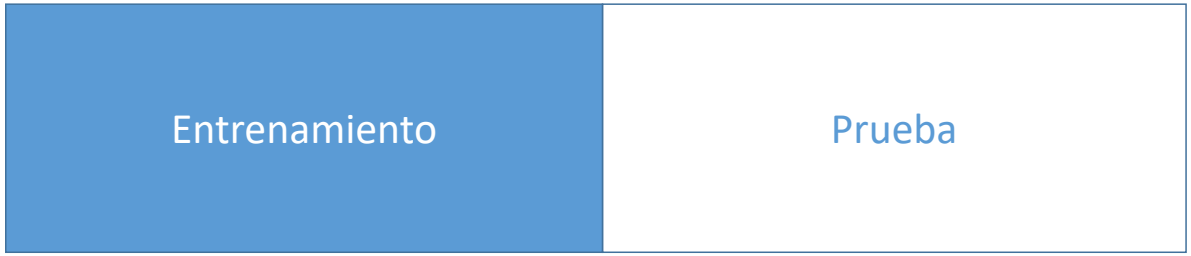
Validación cruzada

- Punto intermedio entre validación simple y dejar una observación fuera
- Realizar varios experimentos con diferentes conjuntos de entrenamiento/prueba

Validación cruzada

- Punto intermedio entre validación simple y dejar una observación fuera
- Realizar varios experimentos con diferentes conjuntos de entrenamiento/prueba
- No necesariamente tantos como número de muestras

Validación simple



Validación simple otra vez

Validación cruzada

```
from sklearn.datasets import load_iris
iris = load_iris()

X = iris.data
y = iris.target

from sklearn.neighbors import KNeighborsClassifier
modelo = KNeighborsClassifier(n_neighbors=1)
```

Validación cruzada

```
from sklearn.cross_validation import train_test_split

X1, X2, y1, y2 = train_test_split(X, y,
    random_state=0, train_size=0.5)

modelo.fit(X1, y1)

y2_modelo = modelo.predict(X2)

from sklearn.metrics import accuracy_score
print("score 1: ", accuracy_score(y2, y2_modelo) )
```

Validación cruzada

```
modelo.fit(X2, y2)

y1_modelo = modelo.predict(X1)

print("score 2: ", accuracy_score(y1, y1_modelo) )
```

Validación cruzada

- Validación cruzada con dos grupos / en dos vías

Validación cruzada

- Validación cruzada con dos grupos / en dos vías
- Validación cruzada con k grupos / en k vías

Validación cruzada

- Validación cruzada con dos grupos / en dos vías
- Validación cruzada con k grupos / en k vías
- Dividir los datos en k grupos del mismo tamaño

Validación cruzada

- Validación cruzada con dos grupos / en dos vías
- Validación cruzada con k grupos / en k vías
- Dividir los datos en k grupos del mismo tamaño
- Usar las observaciones del *primer* grupo como prueba y el resto como entrenamiento

Validación cruzada

- Validación cruzada con dos grupos / en dos vías
- Validación cruzada con k grupos / en k vías
- Dividir los datos en k grupos del mismo tamaño
- Usar las observaciones del *primer* grupo como prueba y el resto como entrenamiento
- Usar las observaciones del *segundo* grupo como prueba y el resto como entrenamiento

Validación cruzada

- Validación cruzada con dos grupos / en dos vías
- Validación cruzada con k grupos / en k vías
- Dividir los datos en k grupos del mismo tamaño
- Usar las observaciones del *primer* grupo como prueba y el resto como entrenamiento
- Usar las observaciones del *segundo* grupo como prueba y el resto como entrenamiento
- ...

Validación cruzada

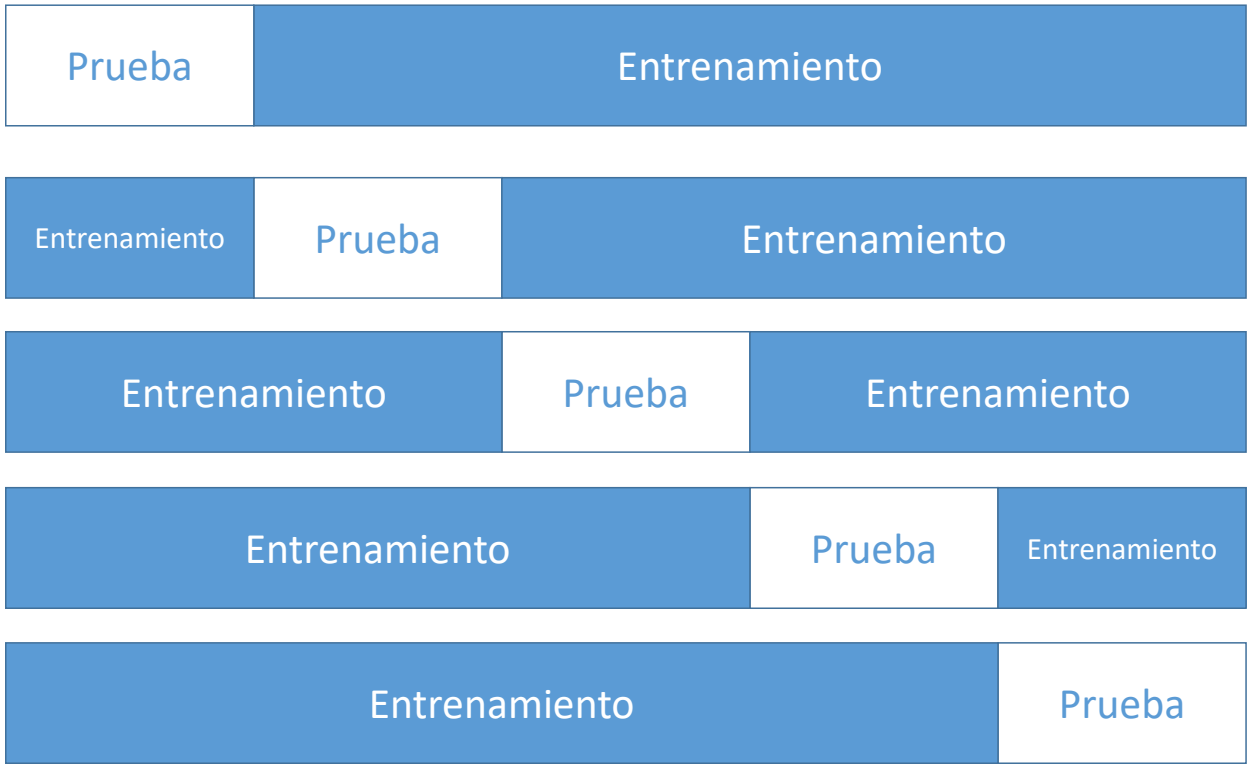
- Validación cruzada con dos grupos / en dos vías
- Validación cruzada con k grupos / en k vías
- Dividir los datos en k grupos del mismo tamaño
- Usar las observaciones del *primer* grupo como prueba y el resto como entrenamiento
- Usar las observaciones del *segundo* grupo como prueba y el resto como entrenamiento
- ...
- Usar las observaciones del *k-ésimo* grupo como prueba y el resto como entrenamiento

Validación cruzada

- Validación cruzada con dos grupos / en dos vías
- Validación cruzada con k grupos / en k vías
- Dividir los datos en k grupos del mismo tamaño
- Usar las observaciones del *primer* grupo como prueba y el resto como entrenamiento
- Usar las observaciones del *segundo* grupo como prueba y el resto como entrenamiento
- ...
- Usar las observaciones del *k-ésimo* grupo como prueba y el resto como entrenamiento
- En cada repetición calcular el error y promediar

Validación cruzada de 2 vías

Validación cruzada de 5 vías



Validación cruzada

```
from sklearn.datasets import load_iris
iris = load_iris()

X = iris.data
y = iris.target

from sklearn.neighbors import KNeighborsClassifier
modelo = KNeighborsClassifier(n_neighbors=1)
```

Validación cruzada

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(modelo, X, y, cv=2)

print(scores)
print(scores.mean())
```

Boostrap

Bootstrap

A partir de una muestra de tamaño n seleccionar una muestra de tamaño m (sin restricciones sobre ele tamaño de n y m)

Bootstrap

A partir de una muestra de tamaño n seleccionar una muestra de tamaño m (sin restricciones sobre ele tamaño de n y m)

A partir de una muestra de tamaño n seleccionar K muestras de tamaño m cada una

Bootstrap

A partir de una muestra de tamaño n seleccionar una muestra de tamaño m (sin restricciones sobre ele tamaño de n y m)

A partir de una muestra de tamaño n seleccionar K muestras de tamaño m cada una

Con cada muestra de tamaño m entrenar el modelo y usar el resultado combinado de todos los modelos

Bootstrap

A partir de una muestra de tamaño n seleccionar una muestra de tamaño m (sin restricciones sobre ele tamaño de n y m)

A partir de una muestra de tamaño n seleccionar K muestras de tamaño m cada una

Con cada muestra de tamaño m entrenar el modelo y usar el resultado combinado de todos los modelos

Ejemplo: métodos de ensamblaje (ensemble methods)

Bootstrap en clasificación

```
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=300, centers=4,
      random_state=0, cluster_std=1.0)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50,
      cmap='rainbow')
```

Bootstrap en clasificación

```
from sklearn.neighbors import KNeighborsClassifier
modelo = KNeighborsClassifier(n_neighbors=10)

from sklearn.ensemble import BaggingClassifier
bag = BaggingClassifier(modelo, n_estimators=100,
      max_samples=0.8, random_state=1)
bag.fit(X, y)
viz_clas(bag, X, y)
```

Bootstrap en clasificación

Bagging: seleccionar subconjuntos de datos para entrenar el modelo (con reemplazo)

Bootstrap en clasificación

Bagging: seleccionar subconjuntos de datos para entrenar el modelo (con reemplazo)

Clase de una nueva observación es la más popular entre los resultados de todos clasificadores

Bootstrap en clasificación

Bagging: seleccionar subconjuntos de datos para entrenar el modelo (con reemplazo)

Clase de una nueva observación es la más popular entre los resultados de todos clasificadores

Ejemplo anterior: clasificador base es K vecinos más cercanos

Bootstrap en clasificación

Bagging: seleccionar subconjuntos de datos para entrenar el modelo (con reemplazo)

Clase de una nueva observación es la más popular entre los resultados de todos clasificadores

Ejemplo anterior: clasificador base es K vecinos más cercanos

Ejemplo ya visto: clasificador base es árbol de decisión (bosque aleatorio)

Bootstrap en clasificación

Bagging: seleccionar subconjuntos de datos para entrenar el modelo (con reemplazo)

Clase de una nueva observación es la más popular entre los resultados de todos clasificadores

Ejemplo anterior: clasificador base es K vecinos más cercanos

Ejemplo ya visto: clasificador base es árbol de decisión (bosque aleatorio)

Pruebe con al menos otro clasificador visto en clase o uno nuevo

Seleccionando el mejor modelo

Buscando el mejor modelo

```
import numpy as np
def make_data(N, err=1.0, rseed=1):
    rng = np.random.RandomState(rseed)
    X = rng.rand(N, 1) ** 2
    y = 10 - 1. / (X.ravel() + 0.1)
    if err > 0:
        y += err * rng.randn(N)
    return X, y
X, y = make_data(40)
```

Buscando el mejor modelo

```
import matplotlib.pyplot as plt
import seaborn; seaborn.set()
X_test = np.linspace(-0.1, 1.1, 500)[:, None]
plt.scatter(X.ravel(), y, color='black')
```

Buscando el mejor modelo

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
def PolynomialRegression(degree=2, **kwargs):
    return make_pipeline(PolynomialFeatures(degree),
                         LinearRegression(**kwargs))
```

Buscando el mejor modelo

```
axis = plt.axis()
for grado in [1, 3, 5]:
    y_test = PolynomialRegression(grado).fit(X, y).
                        predict(X_test)
    plt.plot(X_test.ravel(), y_test,
                        label='grado={0}'.format(grado))
plt.xlim(-0.1, 1.0)
plt.ylim(-2, 12)
plt.legend(loc='best')
```

Buscando el mejor modelo

¿Cuál es el mejor modelo para los datos?

Buscando el mejor modelo

¿Cuál es el mejor modelo para los datos?

Tenemos 3 modelos candidatos

Buscando el mejor modelo

¿Cuál es el mejor modelo para los datos?

Tenemos 3 modelos candidatos

Curva de validación

Buscando el mejor modelo

¿Cuál es el mejor modelo para los datos?

Tenemos 3 modelos candidatos

Curva de validación

Al modificar un (hiper-)parámetro del modelo cómo se comporta el error de prueba

Buscando el mejor modelo

```
from sklearn.model_selection import validation_curve
grados = np.arange(0, 21)
train_score, val_score = validation_curve(
    PolynomialRegression(), X, y,
    'polynomialfeatures__degree', grados, cv=7)
```

Buscando el mejor modelo

```
plt.plot(grados, np.median(train_score, 1),
    color='blue', label='training score')
plt.plot(grados, np.median(val_score, 1),
    color='red', label='validation score')
plt.legend(loc='best')
plt.ylim(0, 1)
plt.xlabel('grados')
plt.ylabel('score')
```

Visualizando el mejor modelo

```
plt.scatter(X.ravel(), y)
lim = plt.axis()
y_test = PolynomialRegression(3).fit(X, y).
    predict(X_test)
plt.plot(X_test.ravel(), y_test);
plt.axis(lim)
```

Buscando los mejores parámetros con una grilla

Buscando los mejores parámetros con una grilla

```
from sklearn.grid_search import GridSearchCV
param_grid = {
  'polynomialfeatures__degree': np.arange(21),
  'linearregression__fit_intercept': [True, False],
  'linearregression__normalize': [True, False]
}

grid = GridSearchCV(PolynomialRegression(),
    param_grid, cv=7)
```

Buscando los mejores parámetros con una grilla

```
grid.fit(X, y)

print(grid.best_params_)
```

Buscando los mejores parámetros con una grilla

```
model = grid.best_estimator_
plt.scatter(X.ravel(), y)
lim = plt.axis()
y_test = model.fit(X, y).predict(X_test)
plt.plot(X_test.ravel(), y_test, hold=True);
plt.axis(lim)
```