Clase 01 - Introducción

Aprendizaje Automático Aplicado

Agenda

- Herramientas
- Tipos de variables
- Atributos Faltantes
- Estandarización de los datos
- Selección de atributos
- Reducción de dimensiones
- Evaluación
- Búsqueda de hiperparametros
- Pipelines
- Clustering

Herramientas

- Jupyter notebook / Google Colab
- numpy
- matplotlib
- scikit-learn
- keras/Tensorflow (Fuera de alcance, por ahora!)

Jupyter Notebook

Instalación local

Corre en su máquina

Tantas sesiones como la máquina soporte

Gestionar ambientes

Instalable con anaconda o pip

Google Colab

Herramienta de Google

Corre en la nube

Una sesión por usuario

Bibliotecas estándares provistas

No tiene estado

Acceso a GPU

Jupyter Notebook/Google Colab

- Permite escribir y ejecutar código Python, organizado en celdas
- Dos tipos de celda: Markdown y Código
- Markdown: Permite documentar el trabajo en un mismo lugar
- Código: permite escribir y ejecutar código Python
- La ejecución se hace celda a celda, y se conserva el estado de una celda a la siguiente
- El usuario es responsable de qué y cuándo ejecutar
- Para ejecutar una celda de código: `ctrl + enter` / ▸

Ejemplo Colab.ipynb

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se guardaron todos los cambios

Comentar Compartir

+ Código + Texto

RAM Disco Editando

Esto es markdown

Sirve para documentar nuestro trabajo, junto con el codigo. Es muy comodo para realizar informes, **se pueden escribir formulas**:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Entre otras cosas....

Para editar esta celda de Markdown, basta hacerle doble click.

[3]
```
a = 1
b = 2
a + b
```

```
3
```

[4]
```
a * b
```

```
2
```

0 s se ejecutó 21:01

Numpy

- Biblioteca para cálculo con matrices de manera eficiente
- Está implementada en C y optimiza el uso de memoria
- El tipo básico es el numpy array
- Tiene un mecanismo muy eficiente llamado *Broad Casting*: las operaciones se aplican a toda la matriz/vector

```
[6] import numpy as np

    a = np.array([[1,2,3], [4,5,6]])
    a

    array([[1, 2, 3],
           [4, 5, 6]])

[7] a * 2

    array([[ 2,  4,  6],
           [ 8, 10, 12]])

[8] a.shape

    (2, 3)
```

NumPy

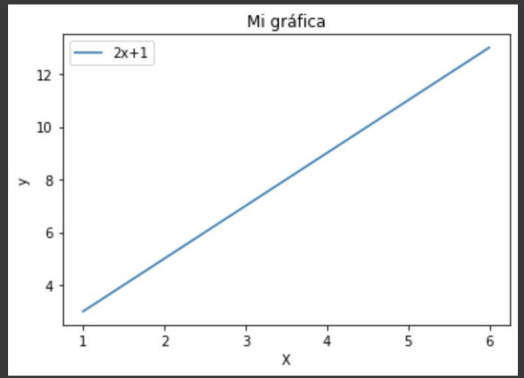
Matplotlib

- Biblioteca para hacer gráficas
- Los elementos de la gráfica ‘se agregan’ utilizando funciones como “`plot`” o “`scatter`” y después se muestra la gráfica usando “show”

```
[ ] import numpy as np
    import matplotlib.pylab as plt

    x = np.array([1,3,6])
    y = 2 * x + 1

    plt.plot(x, y, label='2x+1')
    plt.legend() # Mostramos las etiquetas
    plt.title('Mi gráfica') # Agregamos el titulo
    plt.xlabel('X') # Ponemos nombre al eje x
    plt.ylabel('y') # Ponemos nombre al eje y
    plt.show() # Mostramos el grafico construido hasta ahora
```



Scikit-learn

- Kit de herramientas de machine learning
- Algoritmos clásicos de ML ya implementados
- API claramente definida para facilitar su uso
  - Los parámetros del algoritmo se pasan todos en el constructor del objeto
  - Todos los parámetros tienen siempre valores por defecto
  - Luego tienen dos métodos principales:
    i. `fit`, para “entrenar” el algoritmo
    ii. `predict` o `transform`, para predecir con el objeto ya entrenado
- Muy bien documentada

Datos de ejemplo: Boston

```python
from sklearn.datasets import load_boston

data = load_boston()

print(data.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.
```

Atributos

Atributos, features, características...

Es la forma en que representamos la realidad para nuestro modelo.

Vamos a verlo como un vector.

Ejemplo: Quiero predecir el precio de una casa, la casa va a estar representada por sus **atributos:**

(superficie, #plantas, #cuartos, #baños, barrio, gastos comunes)

Dos tipos de atributos

Numéricos

1. Toman valores en cierto intervalo
2. Potencialmente infinitos valores
3. Se debe normalizar, para que no tenga más relevancia que otros en ciertos algoritmos.

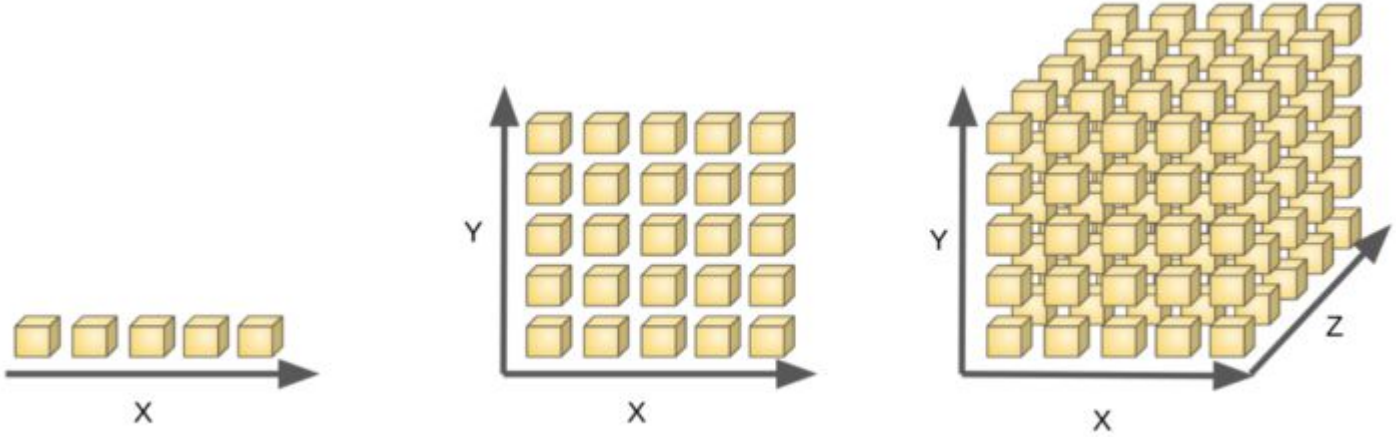
Categóricos

1. Una cantidad finita de posibilidades
2. Algunos algoritmos necesitan que se procesen.
    a. Ordinal Encoding
    b. One-Hot Encoding

Atributos categóricos

- Hay algoritmos que no manejan bien atributos categóricos.
- Para poder utilizar los atributos categóricos que tenemos, es necesario convertirlos a numéricos
- Dos tipos:
  - Ordinales: (malo, regular, bueno), (nada, poco, muchísimo), (común, especial, premium)
  - Nominales: (blanco, rojo, azul), (Uruguay, Argentina, Brasil), (auto, moto, bicicleta),
- Debemos tener en cuenta siempre las **consecuencias** de esta conversión, algunas de ellas son:
- Ordinal Encoding (`sklearn.preprocessing.`**`OrdinalEncoder`**):
  - malo = 0; regular = 1; bueno = 2
  - ¿Qué problemas ocasiona esto?
- One-Hot Encoding (`sklearn.preprocessing.`**`OneHotEncoder`**): cada valor posible, pasa a un binario si/no
  - auto -> [es_auto, no_es_moto, no_es_biciclera] -> [**1**,0,0]
  - bicicleta -> [no_es_auto, no_es_moto, es_biciclera] -> [0,0,**1**]
  - ¿Qué problemas ocasiona esto?

Maldición de la dimensionalidad



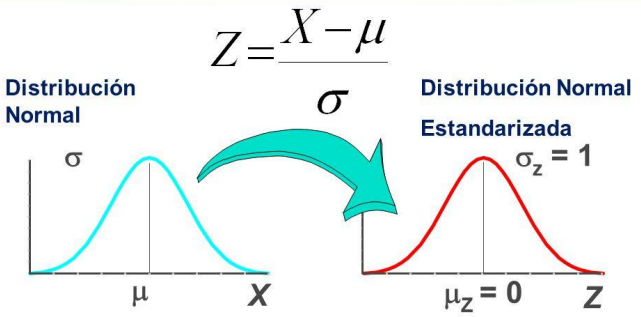
Problemas con atributos

- Atributos Faltantes:
  - Eliminar esos registros
  - Eliminar ese atributo
  - Rellenar siguiendo algún criterio
    - Promedio o más frecuente
    - Promedio o más frecuente junto a otro criterio
    - Entrenar un clasificador
- Manejo de valores desconocidos: que pasa si tenia previsto (auto, moto, bicicleta) y debo codificar camioneta?

Estandarización de los datos

- Es necesario estandarizar los datos.
- Evita que la magnitud de las cantidades tenga más peso en algunos clasificadores:
  - Ejemplo (peso, altura, presión máxima, presión mínima) -> (90, 175,12.6, 7.2)
- Llevar los datos a un mismo rango:
  - Normalizarlos: forzar que tengan media 0 y desviación estándar 1 (`sklearn.preprocessing.`**`StandardScaler`**)
  - Llevarlos al rango [0,1] (`sklearn.preprocessing.`**`MinMaxScaler`**)

$$Z = \frac{X - \mu}{\sigma}$$



Selección de atributos

Como vimos antes, no es cierto que cuanto más, mejor. Entonces, tenemos que ver cómo podemos reducir la cantidad de atributos, y quedarnos con aquellos realmente útiles. Para esto debemos considerar algunas cosas:

- Un atributo que está muy correlacionado con nuestro target, va a ser muy útil
- Si tengo dos atributos que están muy correlacionados entre si, puedo eliminar uno sin perder mucha información
- Si tengo un atributo que tiene muy poca varianza, no está aportando mucho
- Estrategias más complejas:
    - Usar un modelo para ver cuales son las más relevantes
    - Recursive Feature Elimination
    - Sequential Feature Elimination (backward/forward)

Distintas estrategias llevan a distintos conjuntos de atributos, a priori ninguno es mejor que otro.

Selección de atributos

La mayoría vienen implementadas en scikit, siguiendo la estructura que vimos antes:

```
[6] from sklearn.datasets import load_digits
    from sklearn.feature_selection import SelectKBest, chi2

    X, y = load_digits(return_X_y=True)
    print(X.shape)

    selector = SelectKBest(chi2, k=20).fit(X, y)

    X_new = selector.transform(X)
    print(X_new.shape)

    (1797, 64)
    (1797, 20)
```

```
[10] from sklearn.datasets import make_friedman1
     from sklearn.feature_selection import RFE
     from sklearn.svm import SVR

     X, y = make_friedman1(n_samples=50, n_features=10, random_state=0)

     print(X.shape)

     estimator = SVR(kernel="linear")
     selector = RFE(estimator, n_features_to_select=5, step=1)
     selector = selector.fit(X, y)

     X_new = selector.transform(X)
     print(X_new.shape)

     (50, 10)
     (50, 5)
```

Reducción de dimensiones: PCA

Principal Component Analysis

Método de reducción de dimensionalidad

Sirve para visualización

Técnica exploratoria

En python: `sklearn.decomposition.PCA`

PCA

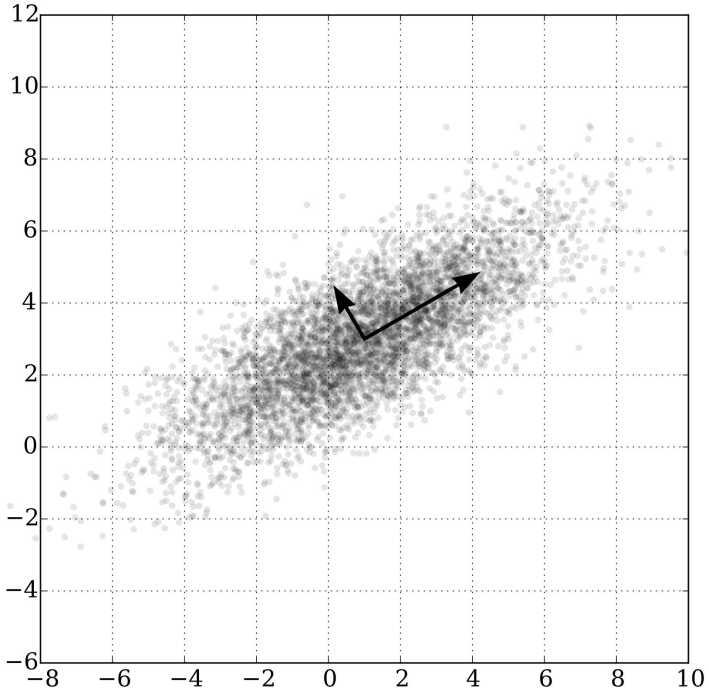
El objetivo es encontrar un conjunto de ejes sobre los cuales los datos tengan la mayor varianza

De esta forma al proyectar los datos quedan “separados” lo más posible

Los datos pasan a una nueva representación de menor dimensionalidad

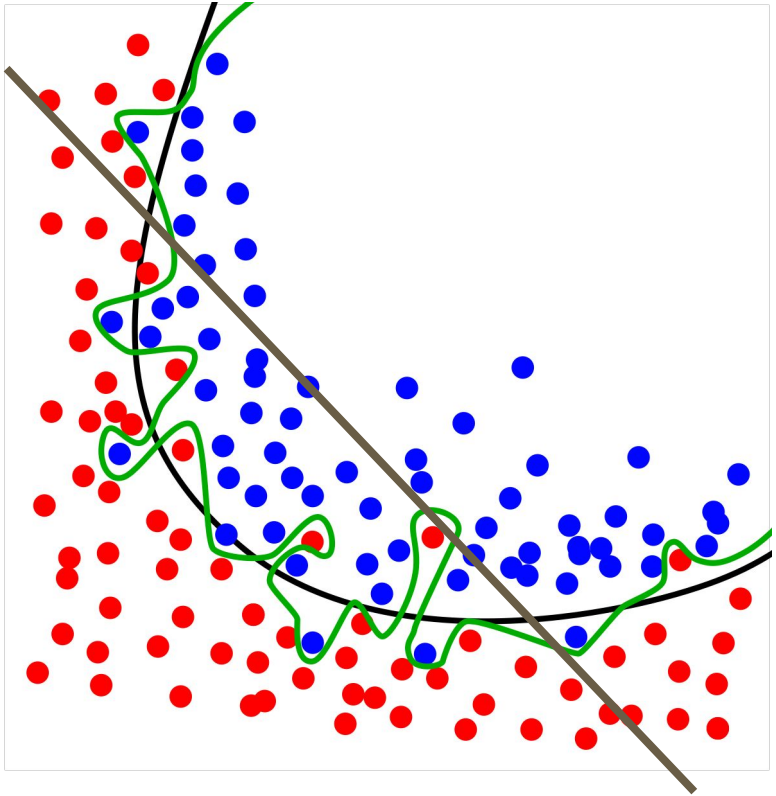
Es una transformación lineal

PCA



Evaluación de Modelos

¿Qué línea separa mejor los puntos?



Un modelo que explica bien todos los datos no necesariamente explica bien la realidad.

Un modelo se **sobreajusta** cuando explica demasiado bien los datos que vio, pero falla cuando se le presentan nuevos.

Esto generalmente se traduce en un error mucho mayor en los datos nuevos.

En la práctica es análogo a memorizar datos

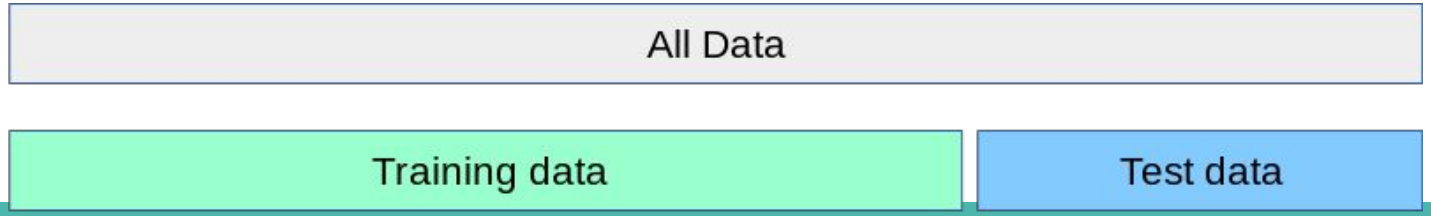
Train - Test split

Queremos un modelo que funcione bien en **todas las instancias** que se va a encontrar en producción.

Durante el desarrollo, tenemos acceso solo a una cantidad, potencialmente enorme pero finita, de instancias.

Solución: separar una parte de esos datos, generalmente entre el 10 y 30%, que voy a dejar fuera de todo el proceso

El conjunto de test no lo vamos a usar **hasta último momento**, ya que es lo más parecido a lo que vamos a encontrar en producción. Estos conjuntos **deben ser independientes**



Train - Val - Test Split

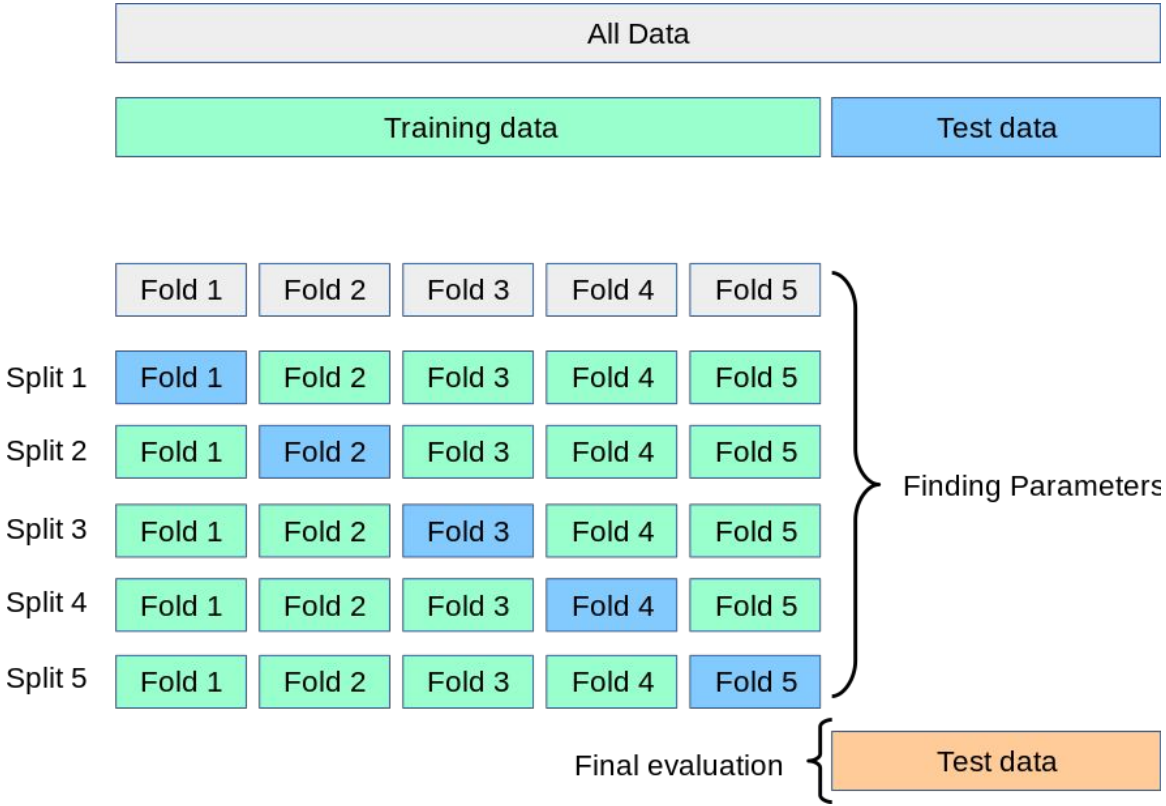
scikit: `sklearn.model_selection.train_test_split`

Si usamos el conjunto de test para tomar decisiones intermedias, corro riesgo de sobreajustarme a los datos de test!

Para evitar esto las opciones más frecuentes son volver a partir en entrenamiento y validación o utilizar **validación cruzada.**

Va a depender de la cantidad de datos disponibles

Validación cruzada



Métricas

Es importante definir qué métrica(s) usar antes de empezar

La métrica a usar depende del problema

Elegir una métrica a optimizar, pero igualmente monitorear las demás

Métricas: clasificación

- En problemas de clasificación, casi todas las métricas parten de la **matriz de confusión**
- Permite comparar el valor real, con el resultado de la clasificación.

		Resultado de la predicción		
		Positivo	Negativo	
Valor actual	Positivo	TP	FN	TP + FN
	Negativo	FP	TN	FP + TN

Métricas: clasificación

La primera medida que se viene a la mente es el acierto:

$$Acierto = \frac{TP + TN}{TP + FP + TN + FN}$$

¿Qué problemas pueden haber?

Métricas: clasificación

Estas medidas sí tienen en cuenta qué pasa en las distintas clases de nuestro dataset.

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN} \qquad F1 = 2\frac{P * R}{P + R}$$

- Miden cosas distintas.
- Dependiendo del sistema, conviene priorizar una o la otra.
- El f1-score pondera ambas.

Métricas: regresión

Regresión: el modelo predice una **cantidad**

¿Por qué no probar con el acierto? (Accuracy)

Real (ground truth)	Predicho	Acierto
20000	20000	1
35000	35001	0
40000	20000	0

Métricas: regresión - MAE

$$\mathrm{MAE}(y, \hat{y}) = \frac{1}{n_{\text{ejemplos}}} \sum_{i=0}^{n_{\text{ejemplos}}-1} |y_i - \hat{y}_i| .$$

Error absoluto medio (Mean Absolute Error)

- Todos los errores pesan lo mismo
- Mantiene las unidades
- Variante sencilla: tomar la mediana en vez del promedio

Métricas: regresión - MSE

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{n_{\text{ejemplos}}} \sum_{i=0}^{n_{\text{ejemplos}}-1} (y_i - \hat{y}_i)^2.$$

Error cuadrático medio (Mean Squared Error)

- NO todos los errores pesan lo mismo
- NO Mantiene las unidades
- Variante sencilla: tomar la raíz cuadrada, RMSE

Métricas: regresión - Error Máximo

$$\text{Max Error}(y, \hat{y}) = max(|y_i - \hat{y}_i|)$$

- **MUY** sensible a outliers
- Mantiene las unidades

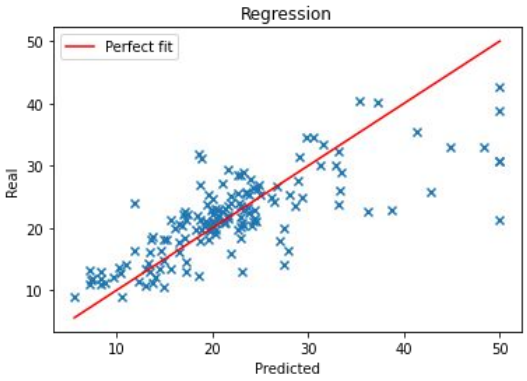
Métricas: regresión - R^2

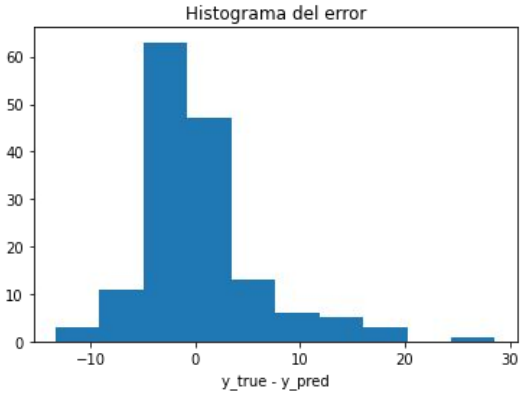
$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Coeficiente de determinación R^2

- Modelo predice siempre promedio, da 0
- El máximo es 1, y es un ajuste perfecto
- No hay unidades

Métricas: regresión





Es útil ver el error gráficamente como un scatter entre (`y_true, y_pred`)
Podemos hacer un histograma, debería quedar centrado en cero y con desviación estándar pequeña.

Métricas en scikit

Todas estas métricas están implementadas en scikit, en el módulo *metrics.*

`sklearn.metrics` : Metrics

See the Model evaluation: quantifying the quality of predictions section and the Pairwise metrics, Affinities and Kernels section of the user guide for further details.

The `sklearn.metrics` module includes score functions, performance metrics and pairwise metrics and distance computations.

Búsqueda de hiperparámetros

Búsqueda de hiperparámetros

Consiste en encontrar los mejores hiperparámetros para nuestros algoritmos

Por ejemplo, en KNN: cantidad de vecinos k y ponderación de vecinos

Busco el k que obtiene mejores resultados, utilizando validación cruzada o un conjunto de validación: **nunca sobre el conjunto de test**

En la medida que agrego hiperparámetros a probar, crece exponencialmente la cantidad de combinaciones a probar.

Cómo buscarlos?

Búsqueda de hiperparámetros: manual search

Es lo más natural cuando empezamos: probar a mano

Que pasa si pongo k=10? Mejor lo aumento a k=20... quizá un intermedio k=15... cuál era el mejor, el 10 o el 20?

Muy ineficiente! **Evitarlo**

Búsqueda de hiperparámetros: grid search

Definir una grilla de parámetros, y probar todas sus combinaciones:

k \ ponderación	Uniforme	Proporcional
10	0.58	0.60
15	0.75	0.64
20	**0.93**	0.90
25	0.66	0.54

Búsqueda de hiperparámetros: grid search

```python
from sklearn.datasets import load_iris
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV, train_test_split

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

parameters = {'n_neighbors': [10,15,20,25],
              'weights': ['uniform', 'distance']}

knn = KNeighborsClassifier()

clf = GridSearchCV(knn, parameters, cv=10, scoring='accuracy')
clf.fit(X_train, y_train)

print(f'Best parameters found: {clf.best_params_}\n')


y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
Best parameters found: {'n_neighbors': 10, 'weights': 'uniform'}

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      0.94      0.97        18
           2       0.92      1.00      0.96        11

    accuracy                           0.98        45
   macro avg       0.97      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

Búsqueda de hiperparámetros: random search

Definir una grilla de parámetros, y probar **algunas** sus combinaciones

k \ ponderación	Uniforme	Proporcional
10	0.58	0.60
12	0.75	-
14	**0.93**	0.90
16	-	0.87
18	0.90	-
20	-	-
22	0.66	0.64
24	-	0.54

Búsqueda de hiperparámetros: random search

```python
from sklearn.datasets import load_iris
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import RandomizedSearchCV, train_test_split

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

parameters = {'n_neighbors': [10,12,14,16,18,20,22,24,26,28,30],
              'weights': ['uniform', 'distance']}

knn = KNeighborsClassifier()

clf = RandomizedSearchCV(knn, parameters, cv=10, scoring='accuracy',
                         random_state=0, n_iter=15)
clf.fit(X_train, y_train)

print(f'Best parameters found: {clf.best_params_}\n')

y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
Best parameters found: {'weights': 'distance', 'n_neighbors': 22}

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      0.94      0.97        18
           2       0.92      1.00      0.96        11

    accuracy                           0.98        45
   macro avg       0.97      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

Pipelines

Breve repaso...

Por ahora vimos varios pasos a concatenar:

1) escalamos atributos
2) seleccionamos los mejores
3) entrenamos el clasificado
4) evaluamos el clasificador

```python
from sklearn.datasets import load_digits
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectKBest, chi2

# load and split dataset
X, y = load_digits(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# 1) scale the atributes to range (0,1)
scaler = MinMaxScaler()
scaler.fit(X_train, y_train)

X_train_scaled = scaler.transform(X_train)

# 2) select the best 100 atributes
selector = SelectKBest(chi2, k=20)
selector.fit(X_train_scaled, y_train)

X_train_scaled_selected = selector.transform(X_train_scaled)

# 3) fit the classifier with scaled-selected attributes
knn = KNeighborsClassifier(n_neighbors=10, weights='distance')

knn.fit(X_train_scaled_selected, y_train)
```

Breve repaso...

Por ahora vimos varios pasos a concatenar:

1) escalamos atributos
2) seleccionamos los mejores
3) entrenamos el clasificado
4) evaluamos el clasificador

Observaciones...

1) Salteamos la elección de hiperparámetros
2) Es incómodo recordar cada paso, y debe ser siempre en el mismo orden

```
[8] from sklearn.metrics import classification_report

    # 1) scale test atributes
    X_test_scaled = scaler.transform(X_test)

    # 2) select ony the best atributes
    X_test_scaled_selected = selector.transform(X_test_scaled)

    # 3) make the prediction
    y_pred = knn.predict(X_test_scaled_selected)

    # 4) evaluate it
    print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        45
           1       0.87      1.00      0.93        52
           2       1.00      0.94      0.97        53
           3       0.96      0.91      0.93        54
           4       0.98      0.98      0.98        48
           5       1.00      0.96      0.98        57
           6       1.00      1.00      1.00        60
           7       0.88      0.96      0.92        53
           8       0.98      0.82      0.89        61
           9       0.93      0.98      0.96        57

    accuracy                           0.95       540
   macro avg       0.96      0.96      0.95       540
weighted avg       0.96      0.95      0.95       540
```

Pipelines!

Una lista de transformadores

Cada uno con un nombre, arbitrario pero único

La lista debe terminar en un clasificador/regresor (algo que implemente un predict)

El pipe en si mismo es un clasificador!

```
[10] from sklearn.pipeline import Pipeline

     pipe = Pipeline([
                ('scaler', MinMaxScaler()),
                ('selector', SelectKBest(chi2, k=20)),
                ('knn', KNeighborsClassifier(n_neighbors=10, weights='distance'))
            ])

     pipe.fit(X_train, y_train)

     y_pred = pipe.predict(X_test)

     print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        45
           1       0.87      1.00      0.93        52
           2       1.00      0.94      0.97        53
           3       0.96      0.91      0.93        54
           4       0.98      0.98      0.98        48
           5       1.00      0.96      0.98        57
           6       1.00      1.00      1.00        60
           7       0.88      0.96      0.92        53
           8       0.98      0.82      0.89        61
           9       0.93      0.98      0.96        57

    accuracy                           0.95       540
   macro avg       0.96      0.96      0.95       540
weighted avg       0.96      0.95      0.95       540
```

Pipelines

El pipe en si mismo es un clasificador!

Por lo tanto, puedo aplicar lo que vimos: grid/random search

Con esto puedo elegir los mejores pasos en el pipeline y sus hiperparámetros

Precaución: la combinación de parámetros debe tener sentido

```python
from sklearn.model_selection import RandomizedSearchCV

pipe = Pipeline([
          ('scaler', MinMaxScaler()),
          ('selector', SelectKBest(chi2, k=20)),
          ('knn', KNeighborsClassifier(n_neighbors=10, weights='distance'))
        ])

parameters = {'scaler': [MinMaxScaler(), None],
              'selector__k': [10,20,40,60],
              'knn__n_neighbors': [10,12,14,16,18,20,22,24,26,28,30],
              'knn__weights': ['uniform', 'distance']}

clf = RandomizedSearchCV(pipe, parameters, cv=10, scoring='accuracy',
                         random_state=0, n_iter=100)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
clf.best_params_
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        45
           1       0.95      1.00      0.97        52
           2       1.00      0.96      0.98        53
           3       0.96      1.00      0.98        54
           4       1.00      0.98      0.99        48
           5       0.98      0.96      0.97        57
           6       0.98      1.00      0.99        60
           7       0.95      1.00      0.97        53
           8       1.00      0.92      0.96        61
           9       0.98      0.98      0.98        57

    accuracy                           0.98       540
   macro avg       0.98      0.98      0.98       540
weighted avg       0.98      0.98      0.98       540

{'knn__n_neighbors': 10,
 'knn__weights': 'distance',
 'scaler': None,
 'selector__k': 60}
```