ntic master
revolucionamos la comunicación

UNIVERSIDAD COMPLUTENSE MADRID

# Tema 2:
# End-to-end Classification Problem

Eduardo Fernández Carrión

# Contenido

1. Exploración de datos
2. Transformación de variables
3. Selección de variables predictivas
4. Técnicas de entrenamiento de los algoritmos
5. Métricas de evaluación (problema de clasificación)
6. Selección de algoritmos (problema de clasificación)
7. Parametrización de algoritmos

# Contenido

1. **Exploración de datos**
2. Transformación de variables
3. Selección de variables predictivas
4. Técnicas de entrenamiento de los algoritmos
5. Métricas de evaluación (problema de clasificación)
6. Selección de algoritmos (problema de clasificación)
7. Parametrización de algoritmos

# Exploración de datos

También llamada *Exploratory Data Analysis* (EDA).

La idea principal en esta fase es "conocer" los datos. Principalmente:

- Variables categóricas y numéricas.
- Distribución de los datos.
- Null values.
- Outliers.
- Balanceo de datos.
- Relaciones entre variables.

# Exploración de datos

## Reading data

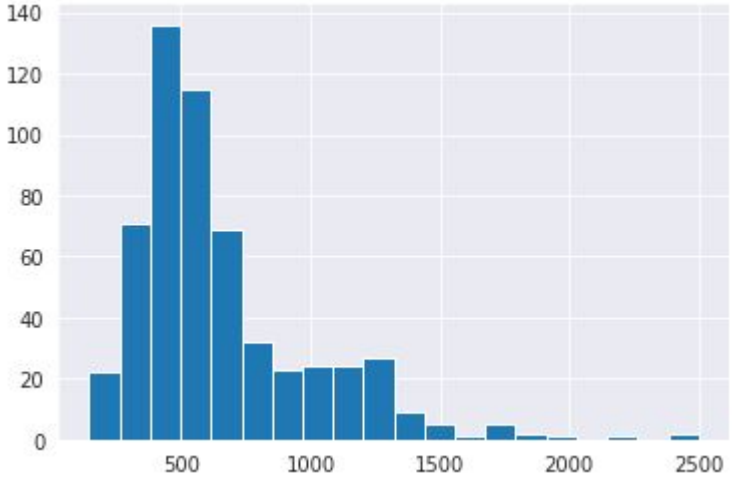
```
data = pd.read_csv('breast_cancer_data.csv')
data.head()
```

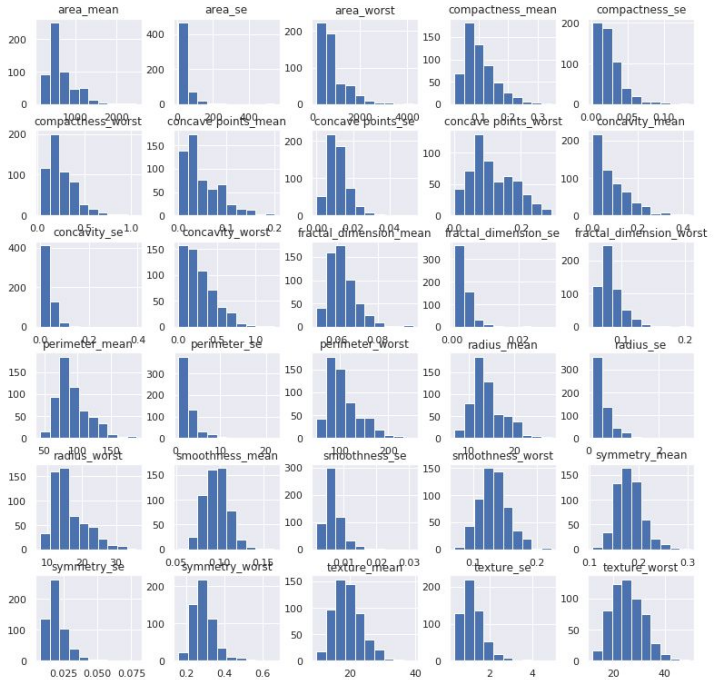
| | id | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | compactness_m |
|---|---|---|---|---|---|---|---|---|
| **0** | 842302 | M | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | 0.27 |
| **1** | 842517 | M | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | 0.07 |
| **2** | 84300903 | M | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | 0.15 |
| **3** | 84348301 | M | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | 0.28 |
| **4** | 84358402 | M | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | 0.13 |

5 rows × 33 columns

# Plots útiles para EDA

## Matplotlib.pyplot



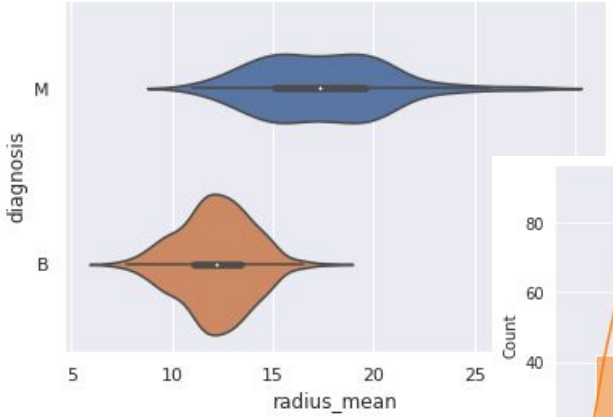

Ref.: https://matplotlib.org/gallery/index.html

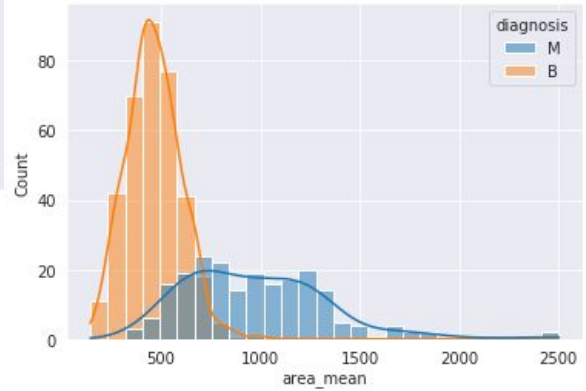
# Plots útiles para EDA

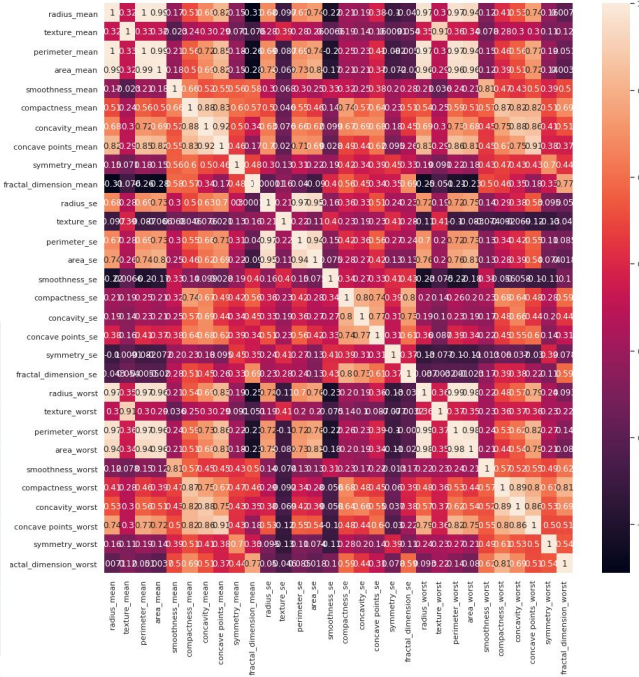
## Seaborn

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style('darkgrid')
sns.__version__
```

```
'0.11.1'
```




Ref.: https://seaborn.pydata.org/examples/index.html

# Contenido

1. Exploración de datos
2. **Transformación de variables**
3. Selección de variables predictivas
4. Técnicas de entrenamiento de los algoritmos
5. Métricas de evaluación (problema de clasificación)
6. Selección de algoritmos (problema de clasificación)
7. Parametrización de algoritmos

# Transformación de variables

También llamada *Feature engineering*.

Consiste en transformar variables existentes o crear nuevas variables a partir de las existentes.

Por defecto, podemos transformar las variables de la manera más personalizada que deseemos.

Sin embargo, suelen existir transformaciones que se repiten con bastante frecuencia.

En este ejemplo vamos a trabajar las siguientes transformaciones:

- *Rescale data*
- *Standardized data*
- *Normalize data*
- *Binarize data*

https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing

# Transformación de variables

## Rescale

$$y = \frac{x - min}{max - min}$$

Rescalamos los datos en el intervalo [0, 1].

Pros & Cons:

- Eliminamos unidades (magnitudes).
- Se compara mejor entre variables, se interpretan mejor los resultados.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))

rescaledX = scaler.fit_transform(X)
```

# Transformación de variables

## Standardize

$$y = \frac{x - \mu}{\sigma}$$

Consiste en tipificar los datos

Pros & Cons:

- Eliminamos unidades (magnitudes).
- Centraliza los valores alrededor del cero.
- No mejora skewness/kurtosis.
- Con datos no distribuidos normalmente, no ganamos mucho.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(X)

rescaledX = scaler.transform(X)
```

# Transformación de variables

## Normalize

$$y = \frac{x}{\sqrt{x_1^2 + \ldots + x_n^2}} = \frac{x}{\sqrt{\|x\|_2}}$$

Rescalamos el vector de fila a su vector unitario.

Pros & Cons:

- Si no existe distribución gaussiana, suele mostrar buenos resultados.
- Tiene sentido solo usarla con más de una feature.

```
from sklearn.preprocessing import Normalizer

scaler = Normalizer().fit(X)

normalizedX = scaler.transform(X)
```

# Transformación de variables

## Binarize

Consiste en crear una variable con valores booleanos (0-1).

A partir de un umbral en el dato, se le asigna valor 1, en caso contrario 0.

Casos de uso:

- Datos nulos vs. no nulos.
- Datos desbalanceados (con muchos ceros).
- Generación de variables dummy, como umbral la media o la mediana.

```
from sklearn.preprocessing import Binarizer

binarizer = Binarizer(threshold=0.0).fit(X)
binaryX = binarizer.transform(X)
```

# Contenido

1. Exploración de datos
2. Transformación de variables
3. **Selección de variables predictivas**
4. Técnicas de entrenamiento de los algoritmos
5. Métricas de evaluación (problema de clasificación)
6. Selección de algoritmos (problema de clasificación)
7. Parametrización de algoritmos

# Selección de variables predictivas

También llamada *Feature selection*.

Consiste en utilizar criterios de selección de las variables más robustas y con mayor capacidad predictora.

En este ejemplo vamos a trabajar las siguientes técnicas:

- *Chi-squared test*
- *Recursive Feature Elimination (RFE)*
- *Principal Component Analysis (PCA)*
- *Feature Importance*

https://scikit-learn.org/stable/modules/feature_selection.html#

# Selección de variables predictivas

## Chi-squared test

Podemos elegir variables robustas basándonos en tests estadísticos.

El test Chi-squared contrasta cada variable con la variable target, es un test de independencia que destaca aquellas que son relevantes para la clasificación.

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X, y)

fit.scores_
fit.transform(X)
```

# Selección de variables predictivas

## Feature Importance

A los algoritmos (tipo regresión) podemos consultarles tras ser entrenados por el valor de sus coeficientes de las variables para saber su relevancia en la predicción de las variable objetivo.

A los algoritmos (tipo árbol) podemos consultarles tras ser entrenados por la importancia de las variables (que está basado en el factor de impuridad de Gini)

https://victorzhou.com/blog/gini-impurity/,

http://papers.neurips.cc/paper/4928-understanding-variable-importances-in-forests-of-randomized-trees.pdf

```
from sklearn.ensemble import ExtraTreesClassifier

model = ExtraTreesClassifier()
model.fit(X, y)
print(model.feature_importances_)
```

# Selección de variables predictivas

## Recursive Feature Elimination (RFE)

RFE elimina una a una las variables que menos importancia tienen para el modelo.

Ajusta el algoritmo tantas veces como variables tenemos menos el número de variables deseadas.

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
rfe = RFE(model, 3)
fit = rfe.fit(X, y)
```

# Selección de variables predictivas

## Principal Component Analysis (PCA)

PCA es una técnica para reducción de variables basándose en la extracción de los autovalores y autovectores de la matriz de varianzas-covarianzas. Es decir, localizando aquellas variables que más varianza acumulan.

```
from sklearn.decomposition import PCA

pca = PCA(n_components=3)
fit = pca.fit(X)
```

# Contenido

1. Exploración de datos
2. Transformación de variables
3. Selección de variables predictivas
4. **Técnicas de entrenamiento de los algoritmos**
5. Métricas de evaluación (problema de clasificación)
6. Selección de algoritmos (problema de clasificación)
7. Parametrización de algoritmos

# Técnicas de entrenamiento

También llamada *model evaluation.*

La mejor manera de evaluar el rendimiento de un algoritmo sería hacer predicciones para nuevos datos de los que ya se conocen las respuestas.

La segunda mejor manera es utilizar técnicas de remuestreo que permitan hacer estimaciones precisas de cuán bien se desempeñará su algoritmo con los nuevos datos.

En este ejemplo vamos a trabajar las siguientes técnicas:

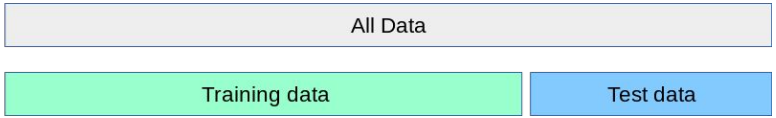
- *Train Test sets*
- *k-folds Cross-Validation*
- *Leave One Out (LOO) Cross-Validation*
- *Shuffle Split Cross-Validation*

https://scikit-learn.org/stable/modules/classes.html#module-sklearn.model_selection

# Técnicas de entrenamiento

## Train Test sets

Subdividimos el dataset original en dos partes: train y test. Entrenamos el algoritmo con train y hacemos predicciones con test.




Pros & Cons:

- Es la técnica más sencilla y rápida. Ideal para datasets muy grande.
- No suele ajustar muy bien con modelos muy sesgados.

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2)
model = LogisticRegression()
model.fit(X_train, Y_train)
result = model.score(X_test, Y_test)
```

# Técnicas de entrenamiento

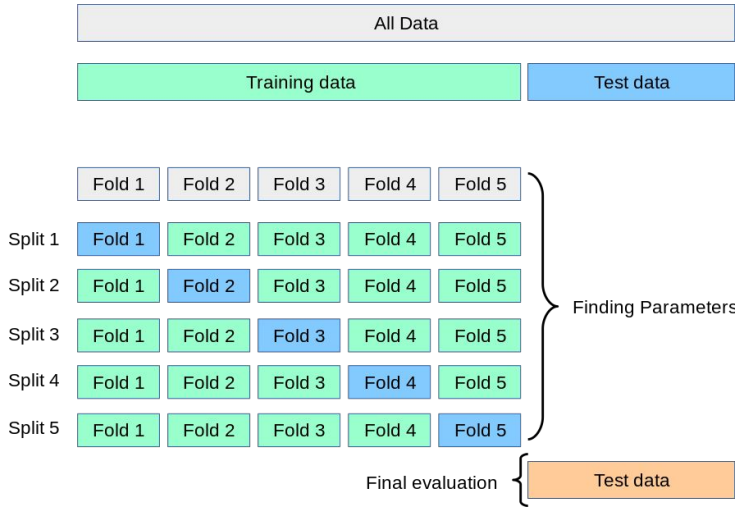
## k-folds Cross-Validation

Subdividimos el dataset original en *k* partes. A cada parte la llamamos *fold*. El algoritmo ahora se entrena con *k-1* folds y se testea con el fold restante. Se repite este proceso *k* veces.

Pros & Cons:

- Es más lenta, tarda más tiempo computacional.
- Es más robusta a modelos sesgados.

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

kfold = KFold(n_splits=10, random_state=7)
model = LogisticRegression()
results = cross_val_score(model, X, y, cv=kfold)
```



# Técnicas de entrenamiento

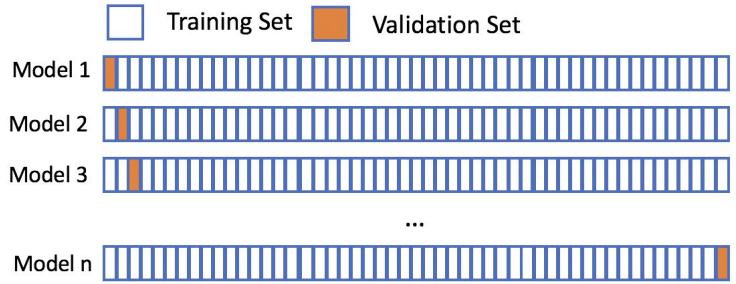
## LOO Cross-Validation

Subdividimos el dataset original en *k* folds, pero *k* será el número de registros.

Pros & Cons:

- Es muy lenta, y muy costosa computacionalmente.
- Es más robusta a modelos sesgados.

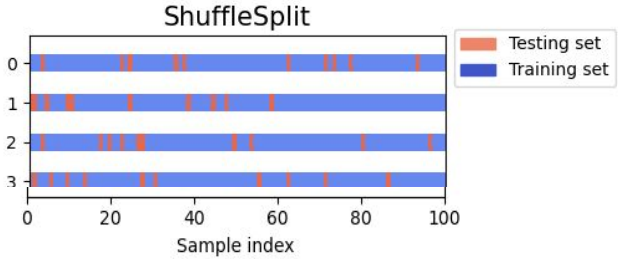
```
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score

loocv = LeaveOneOut()
model = LogisticRegression()
results = cross_val_score(model, X, y, cv=loocv)
```



# Técnicas de entrenamiento

## Shuffle Split Cross-Validation

Es otra variante de cross validation en la que realizar un train/test split completamente aleatorio cada vez.




Pros & Cons:

- Cada repetición puede incluir los mismos datos en cada conjunto.

```
from sklearn.model_selection import ShuffleSplit

kfold = ShuffleSplit(n_splits=10, test_size=0.2, random_state=2)

model= LogisticRegression()
results = cross_val_score(model, X, y, cv=kfold)
```

# Contenido

1. Exploración de datos
2. Transformación de variables
3. Selección de variables predictivas
4. Técnicas de entrenamiento de los algoritmos
5. **Métricas (problema de clasificación)**
6. Algoritmos (problema de clasificación)
7. Parametrización de algoritmos

# Métricas

Las métricas sirven principalmente para evaluar la bondad de cualquier modelo.

Las métricas influyen además en el entrenamiento de cualquier modelo, ya que son la función de pérdida (*loss function*) que el algoritmo siempre trata de optimizar.

Dependiendo del tipo de problema, las métricas son diferentes. En este ejemplo, trabajaremos las siguientes métricas de clasificación:

- Confusion matrix
- Accuracy
- Logarithmic loss
- AUC - Area Under ROC Curve

https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics

# Métricas

## Matriz de confusión

Tras entrenar un algoritmo de clasificación, podemos examinar los resultados en esta matriz.

A partir de esta matriz podemos crear diversas métricas: *accuracy, recall, f1, precision*...

| | Predicted 0 | Predicted 1 |
|---|---|---|
| Actual 0 | TN | FP |
| Actual 1 | FN | TP |

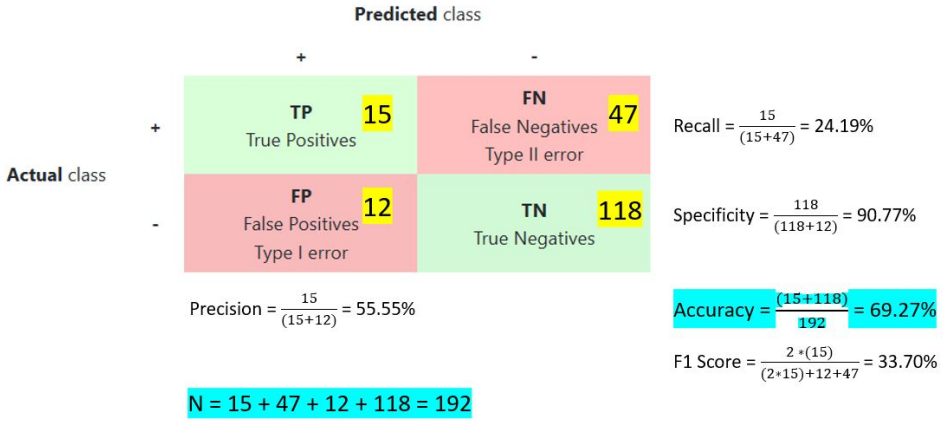
```
from sklearn.metrics import confusion_matrix

model = LogisticRegression()
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
matrix = confusion_matrix(Y_test, predicted)
```

# Métricas

## Accuracy

Es la proporción de los registros correctamente clasificados (TP+TN).

- Como accuracy, otras métricas pueden extraerse de la matriz de confusión: *precision, recall, ...*




```
kfold = KFold(n_splits=10, random_state=7)
model = LogisticRegression()
scoring = 'accuracy'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
```

# Métricas

## Negative Logarithmic Loss

Basada en la Entropía y la función log-likelihood (verosimilitud).

$$L_{\log}(y, p) = -(y \log(p) + (1 - y) \log(1 - p)) \qquad y \in \{0, 1\}$$
$$p = \Pr(y = 1)$$

```
kfold = KFold(n_splits=10, random_state=7)
model = LogisticRegression()
scoring = 'neg_log_loss'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
```

# Métricas

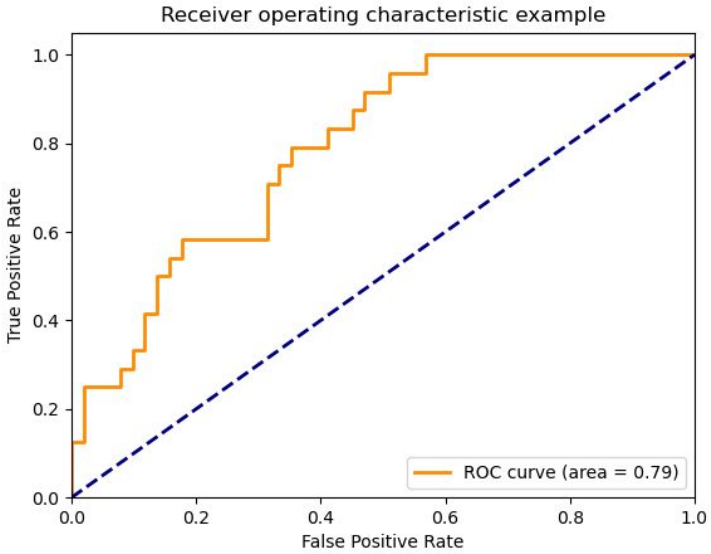
## AUC

*Area Under Receiver Operating Curve (Area Under ROC).*

https://en.wikipedia.org/wiki/Receiver_operating_characteristic

ROC nos permite estimar la relación entre la tasa de Falsos Positivos y Verdaderos Positivos.

- El área máxima es 1 y el mínimo es 0.5.




```
kfold = KFold(n_splits=10, random_state=7)
model = LogisticRegression()
scoring = 'roc_auc'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
```

# Contenido

1. Exploración de datos
2. Transformación de variables
3. Selección de variables predictivas
4. Técnicas de entrenamiento de los algoritmos
5. Métricas de evaluación (problema de clasificación)
6. **Selección de algoritmos (problema de clasificación)**
7. Parametrización de algoritmos

# Selección de algoritmos

Para los modelos de clasificación, existen numerosos algoritmos.

Pueden ser lineales o no lineales. En este ejemplo, trabajaremos dos lineales y cinco no lineales:

- *Logistic Regression*
- *Linear Discriminant Analysis*
- *k-Nearest Neighbors*
- *Naive Bayes*
- *Classification Trees*
- *Support Vector Machines*
- *Random Forest*

https://scikit-learn.org/stable/supervised_learning.html

# Selección de algoritmos

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

```
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('DTC', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('RFC', RandomForestClassifier()))
models.append(('SVM', SVC()))
```

# Selección de algoritmos

```
results = []
names = []
scoring = 'accuracy'
for name, model in models:
    kfold = KFold(n_splits=10, random_state=7)
    cv_results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

# Contenido

1. Exploración de datos
2. Transformación de variables
3. Selección de variables predictivas
4. Técnicas de entrenamiento de los algoritmos
5. Métricas de evaluación (problema de clasificación)
6. Selección de algoritmos (problema de clasificación)
7. **Parametrización de algoritmos**

# Parametrización de algoritmos

También llamada *Model tuning* o *Model Hyper-Parameterization*.

Cada algoritmo ofrece diferentes opciones de parametrización. Dependiendo del algoritmo elegido como base del modelo, ahora tendremos que tomar la mejor configuración para que los resultados sean mejores.

En este ejemplo vamos a evaluar el algoritmo de *Decision Tree* que tiene las siguientes parametrizaciones:

```
model = DecisionTreeClassifier()
model
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

# Parametrización de algoritmos

## GridSearchCV

Vamos a configurar alguna de estas parametrizaciones y ejecutar varias pruebas de modelos para ver, como respuesta, qué parametrización ofrece resultados óptimos:

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier

```
from sklearn.model_selection import GridSearchCV


param_grid = {'max_depth': [3, 6, 12, 24],
              'criterion': ['gini', 'entropy']}
model = DecisionTreeClassifier()

rsearch = GridSearchCV(estimator=model, param_grid=param_grid)
rsearch.fit(X, Y)
```

ntic master
revolucionamos la comunicación

UNIVERSIDAD COMPLUTENSE MADRID