

# Fundamentos de Aprendizaje Automático y Reconocimiento de Patrones

Introducción a scikit-learn

Instituto de Ingeniería Eléctrica  
Facultad de Ingeniería



UNIVERSIDAD  
DE LA REPÚBLICA  
URUGUAY

# Tabla de contenido

- ① ¿Qué es scikit-learn?
- ② Datasets
- ③ División en conjunto de Entrenamiento y Test
- ④ Preprocesamiento de datos
- ⑤ Aprendizaje supervisado
- ⑥ Métricas de desempeños

# Tabla de contenido

- 1 ¿Qué es scikit-learn?
- 2 Datasets
- 3 División en conjunto de Entrenamiento y Test
- 4 Preprocesamiento de datos
- 5 Aprendizaje supervisado
- 6 Métricas de desempeños

# ¿Qué es scikit-learn?



- Biblioteca de *machine learning* desarrollada sobre
  - python
  - numpy
  - matplotlib
  - scipy
- En el curso utilizaremos la versión **1.3.0**
  - `pip install scikit-learn==1.3.0`

# ¿Qué es scikit-learn?



- Biblioteca de *machine learning* desarrollada sobre
  - python
  - numpy
  - matplotlib
  - scipy
- En el curso utilizaremos la versión **1.3.0**
  - `pip install scikit-learn==1.3.0`



# Tabla de contenido

- 1 ¿Qué es scikit-learn?
- 2 **Datasets**
- 3 División en conjunto de Entrenamiento y Test
- 4 Preprocesamiento de datos
- 5 Aprendizaje supervisado
- 6 Métricas de desempeños

# Datasets

- Scikit-learn viene con algunos conjuntos de datos clásicos de aprendizaje automático
  - Iris plants
  - Optical recognition of handwritten digits
  - Breast cancer wisconsin (diagnostic)
  - ...
- Además cuenta con una API para descargar otros
  - California Housing
  - Kddcup 99
  - ...

Por más información visitar [API Reference: Datasets](#)



# Base de datos iris

```
from sklearn.datasets import load_iris
```

```
data = load_iris()
```

```
print(data.keys())
```

```
dict_keys(['data', 'target', 'target_names',  
          'DESCR', 'feature_names', 'filename'])
```

```
# Preguntando el valor de la clave
```

```
print(data['target_names'])
```

```
# Directamente mediante data.
```

```
print(data.target_names)
```

# Base de datos iris

```
from sklearn.datasets import load_iris
```

```
data = load_iris()
```

```
print(data.keys())
```

```
dict_keys(['data', 'target', 'target_names',  
          'DESCR', 'feature_names', 'filename'])
```

```
# Preguntando el valor de la clave
```

```
print(data['target_names'])
```

```
# Directamente mediante data.
```

```
print(data.target_names)
```

# Base de datos iris

```
from sklearn.datasets import load_iris
```

```
data = load_iris()
```

```
print(data.keys())
```

```
dict_keys(['data', 'target', 'target_names',  
          'DESCR', 'feature_names', 'filename'])
```

```
# Preguntando el valor de la clave
```

```
print(data['target_names'])
```

```
# Directamente mediante data.
```

```
print(data.target_names)
```

# Tabla de contenido

- 1 ¿Qué es scikit-learn?
- 2 Datasets
- 3 División en conjunto de Entrenamiento y Test
- 4 Preprocesamiento de datos
- 5 Aprendizaje supervisado
- 6 Métricas de desempeños

# División en conjunto de Entrenamiento y Test

```
# Se importa el modulo que hace  
# la division del conjunto  
from sklearn.model_selection import train_test_split
```

```
# Se realiza la division  
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=0)
```

```
# Se realiza la division estratificada  
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=0, stratify=y)
```

```
# Se realiza la division especificando conjunto test  
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=0, test_size=0.2)
```

# División en conjunto de Entrenamiento y Test

```
# Se importa el modulo que hace  
# la division del conjunto  
from sklearn.model_selection import train_test_split
```

```
# Se realiza la division  
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=0)
```

```
# Se realiza la division estratificada  
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=0, stratify=y)
```

```
# Se realiza la division especificando conjunto test  
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=0, test_size=0.2)
```

# División en conjunto de Entrenamiento y Test

```
# Se importa el modulo que hace  
# la division del conjunto  
from sklearn.model_selection import train_test_split
```

```
# Se realiza la division  
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=0)
```

```
# Se realiza la division estratificada  
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=0, stratify=y)
```

```
# Se realiza la division especificando conjunto test  
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=0, test_size=0.2)
```

# Tabla de contenido

- 1 ¿Qué es scikit-learn?
- 2 Datasets
- 3 División en conjunto de Entrenamiento y Test
- 4 Preprocesamiento de datos
- 5 Aprendizaje supervisado
- 6 Métricas de desempeños



# Estandarización de los datos

- StandardScaler

```
# Se importa el modulo que hace la transformacion  
from sklearn.preprocessing import StandardScaler  
  
# Se crea un scaler  
scaler = StandardScaler()  
# Se encuentran los parametros de estandarizacion  
scaler.fit(X_train)  
  
# Se estandarizan los datos de entrenamiento y test  
standardized_X = scaler.transform(X_train)  
standardized_X_test = scaler.transform(X_test)
```

# Escalado de las características

- MinMaxScaler

```
# Se importa el modulo que hace el escalado  
from sklearn.preprocessing import MinMaxScaler  
  
# Se encuentran los parametros de escalado  
scaler = MinMaxScaler().fit(X_train)  
  
# Se escalan los datos de entrenamiento y test  
scaled_X = scaler.transform(X_train)  
scaled_X_test = scaler.transform(X_test)
```

# Manejo de datos faltantes

- Imputer

```
# Se importa el modulo que maneja los datos faltantes  
from sklearn.preprocessing import Imputer  
  
# Se define la estrategia a seguir en el caso de valores faltantes  
imp = Imputer(missing_values=np.nan,  
              strategy='mean', axis=0)  
  
# Se rellenan esos valores  
X_full = imp.fit_transform(X)
```

# PolynomialFeatures

- PolynomialFeatures

```
# Se importa el modulo que hace  
# la transformacion polinomial  
from sklearn.preprocessing import PolynomialFeatures  
  
# Se define la transformacion a realizar  
poly = PolynomialFeatures(5)  
  
# Se realiza la transformacion  
Xpoly = poly.fit_transform(X)
```

# Tabla de contenido

- 1 ¿Qué es scikit-learn?
- 2 Datasets
- 3 División en conjunto de Entrenamiento y Test
- 4 Preprocesamiento de datos
- 5 Aprendizaje supervisado**
- 6 Métricas de desempeños

# Regresión lineal

```
# Se importa el modulo  
from sklearn.linear_model import LinearRegression  
  
# Se define el clasificador/regresor  
lr = LinearRegression()  
  
# Se encuentran los parametros  
lr.fit(X_train, y_train)  
  
# Se predicen los valores con el conjunto de test  
y_pred = lr.predict(X_test)
```

# Regresión lineal mediante descenso por gradiente

```
# Se importa el modulo  
from sklearn.linear_model import SGDRegressor  
  
# Se define el clasificador/regresor  
sgd = SGDRegressor()  
  
# Se encuentran los parametros  
sgd.fit(X_train, y_train)  
  
# Se predicen los valores con el conjunto de test  
y_pred = sgd.predict(X_test)
```

# Regresión lineal regularizada

```
# Se importa el modulo
from sklearn.linear_model import RidgeClassifier

# Se define el clasificador/regresor
ridge_clf = RidgeClassifier()

# Se encuentran los parametros
ridge_clf.fit(X_train, y_train)

# Se predicen los valores con el conjunto de test
y_pred = ridge_clf.predict(X_test)
```



# Regresión logística

```
# Se importa el modulo  
from sklearn.linear_model import LogisticRegression  
  
# Se define el clasificador/regresor  
log_clf = LogisticRegression(solver='sag')  
  
# Se encuentran los parametros  
log_clf.fit(X_train, y_train)  
  
# Se predicen los valores con el conjunto de test  
y_pred = log_clf.predict(X_test)
```

- Una vez que se ejecuta el método *fit* se tiene acceso a los parámetros calculados durante el fit.
- Por ejemplo, en el clasificador de regresión logística se puede acceder a los pesos y al número de iteraciones

```
# se obtienen los pesos
W = log_clf.coef_
# se obtiene el bias
b = log_clf.intercept_
# se obtiene el numero de iteraciones
n_iter = log_clf.n_iter_
```

# Red neuronal

```
# Se importa el modulo
from sklearn.neural_network import MLPClassifier

# Se define el clasificador/regresor
mlp_clf = MLPClassifier(hidden_layer_sizes=(7,),
                        activation='relu', max_iter=10000,
                        alpha=0, solver='sgd', verbose=10,
                        tol=1e-4, random_state=43,
                        learning_rate_init=.001)

# Se encuentran los parametros
mlp_clf.fit(X_train, y_train)

# Se predicen los valores con el conjunto de test
y_pred = mlp_clf.predict(X_test)
```

# Tabla de contenido

- 1 ¿Qué es scikit-learn?
- 2 Datasets
- 3 División en conjunto de Entrenamiento y Test
- 4 Preprocesamiento de datos
- 5 Aprendizaje supervisado
- 6 Métricas de desempeños**

# Accuracy

- Se puede calcular de dos formas

```
# 1. Utilizando el clasificador
```

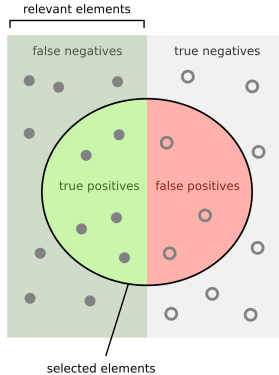
```
clf.score(X_test, y_test)
```

```
# 2. Importando la funcion accuracy_score
```

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(y_test, y_pred)
```

# Precision-Recall



How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

# Precision-Recall

- Precision =  $VP / (VP + FP)$

```
from sklearn.metrics import precision_score  
  
precision_score(y_test, y_pred)
```

- Recall =  $VP / (VP + FN)$

```
from sklearn.metrics import recall_score  
  
recall_score(y_test, y_pred)
```

# Ejercicio

Mejorar el desempeño, en términos de *accuracy*, tanto de la **red neuronal de dos capas** como del clasificador que utiliza **regresión logística**. Algunas de las modificaciones que se pueden evaluar son las siguientes:

- Modificación del *learning rate*
- Regularización mediante *weight decay*
- Regularización mediante *early stopping*
- En el caso de la red neuronal, además:
  - Modificación del número de nodos en capa oculta de la red
  - Función de *activación* utilizada
  - Inicialización