

Taller de Verano: 100 páginas de Machine Learning

Maikol Solís

15/1/24

Tabla de contenidos

1 Taller de verano

2 Cómo funciona el aprendizaje supervisado

Veremos el caso de las máquinas de soporte vectorial (SVM) para clasificación.

- **Paso #1: Cargar librerías**

```
import matplotlib.pyplot as plt

from sklearn import svm
from sklearn.datasets import make_blobs
from sklearn.inspection import DecisionBoundaryDisplay

from scipy.stats import distributions
from numpy import sum
import numpy as np
```

- **Paso #2: Crear datos**

Se crean 40 puntos usando la función `make_blobs`. Esta crea un conjunto de puntos separados en dos grupos.

```
X, y = make_blobs(n_samples=40, centers=2, random_state=6)
```

- **Paso #3: Crear el modelo**

```
clf = svm.SVC(kernel="linear", C=1000)
```

- **Paso #4: Entrenar el modelo**

```
clf.fit(X, y)
```

`SVC(C=1000, kernel='linear')`

- **Paso #5: Visualizar el modelo**

```

plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)

# plot the decision function
ax = plt.gca()
DecisionBoundaryDisplay.from_estimator(
    clf,
    X,
    plot_method="contour",
    colors="k",
    levels=[-1, 0, 1],
    alpha=0.5,
    linestyle=["--", "-", "--"],
    ax=ax,
)
# plot support vectors
ax.scatter(
    clf.support_vectors_[:, 0],
    clf.support_vectors_[:, 1],
    s=100,
    linewidth=1,
    facecolors="none",
    edgecolors="k",
)
plt.show()

```

./1_introduccion_files/figure-pdf/cell-6-output-1.pdf

- **Referencias**

1. <https://scikit-learn.org/stable/modules/svm.html#>
2. https://scikit-learn.org/stable/auto_examples/svm/plot_separating_hyperplane.html#sphx-gl-auto-examples-svm-plot-separating-hyperplane-py

3 Estimación de parametros bayesiano

```
alpha = 10
beta = 10
n = 20
Nsamp = 201 # no of points to sample at
p = np.linspace(0, 1, Nsamp)
deltap = 1./(Nsamp-1) # step size between samples of p

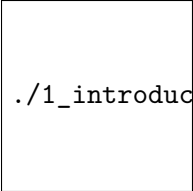
prior = distributions.beta.pdf(p, alpha, beta)

for i in range(1, 9):

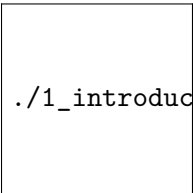
    r = 2**i
    n = (3.0/2.0)*r
    like = distributions.binom.pmf(r, n, p)
    like = like/(deltap*sum(like)) # for plotting convenience only
    post = distributions.beta.pdf(p, alpha+r, beta+n-r)

# make the figure
plt.figure()
plt.plot(p, post, 'k', label='posterior')
plt.plot(p, like, 'r', label='likelihood')
plt.plot(p, prior, 'b', label='prior')
plt.xlabel('p')
plt.ylabel('PDF')
plt.legend(loc='best')
plt.title('r/n={}/{:0f}'.format(r, n))
plt.show()
```

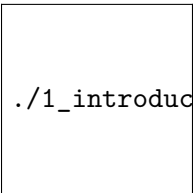
./1_introduccion_files/figure-pdf/cell-7-output-1.pdf



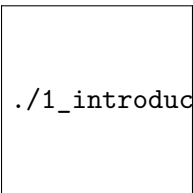
`./1_introduccion_files/figure-pdf/cell-7-output-2.pdf`



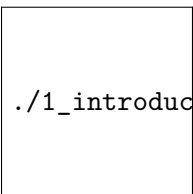
`./1_introduccion_files/figure-pdf/cell-7-output-3.pdf`



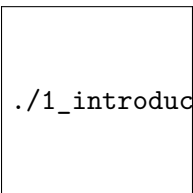
`./1_introduccion_files/figure-pdf/cell-7-output-4.pdf`



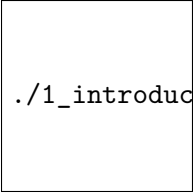
`./1_introduccion_files/figure-pdf/cell-7-output-5.pdf`



`./1_introduccion_files/figure-pdf/cell-7-output-6.pdf`



`./1_introduccion_files/figure-pdf/cell-7-output-7.pdf`



`./1_introduccion_files/figure-pdf/cell-7-output-8.pdf`

4 Día 2

4.1 Regresión Lineal

4.2 Regresión Logística

4.3 Decision Trees

4.3.1 Definición

Un árbol de decisión (DT) es un grafo no cíclico que se utiliza para tomar decisiones (clasificar). En cada nodo (rama) del grafo se evalúa uno de los *features*. Si el resultado de la evaluación es cierto (o está debajo de un umbral), se sigue la rama de la izquierda, si no se va a la derecha.

Por lo tanto, los DT son un modelo no paramétrico.

Para crear el DT, se **intenta** optimizar el promedio de la máxima verosimilitud:

$$\frac{1}{N} \sum_{i=1}^N (y_i \ln f_{ID3}(x_i) + (1 - y_i) \ln (1 - f_{ID3}(x_i)))$$

donde f_{ID3} es un DT y $f_{ID3}(x) \stackrel{\text{def}}{=} Pr(y = 1|x)$

4.3.2 Construcción

Para construir el árbol, en cada nodo de decisión, se intenta minimizar la entropía de la información.

La entropía de un conjunto \mathcal{S} viene dada por:

$$H(\mathcal{S}) \stackrel{\text{def}}{=} -f_{ID3}^{\mathcal{S}} \log_2(f_{ID3}^{\mathcal{S}}) - (1 - f_{ID3}^{\mathcal{S}}) \log_2(1 - f_{ID3}^{\mathcal{S}})$$

Y si un grupo se divide en dos, la entropía es la suma ponderada de cada subconjunto:

$$H(\mathcal{S}_-, \mathcal{S}_+) \stackrel{\text{def}}{=} \frac{|\mathcal{S}_-|}{|\mathcal{S}|} H(\mathcal{S}_-) + \frac{|\mathcal{S}_+|}{|\mathcal{S}|} H(\mathcal{S}_+)$$

4.3.3 Ejemplo

Consideremos los siguientes datos:

Atributos:

- Edad: viejo (v), media-vida(m), nuevo (nv)
- Competencia: no(n), sí(s)
- Tipo: software (swr), hardware (hwr)

Edad	Competencia	Tipo	Ganancia
v	s	swr	baja
v	n	swr	baja
v	n	hwr	baja
m	s	swr	baja
m	s	hwr	baja
m	n	hwr	sube
m	n	swr	sube
nv	s	swr	sube
nv	n	hwr	sube
nv	n	swr	sube

Cálculo de las entropías: Primero se tiene que probar todos los features para ver cuál tiene mayor ganancia de información (reduce la entropía)

Entropía total:

$$H(S) = \text{Entropía de los casos baja} + \text{Entropía de los casos sube}$$

$$H(s) = -\frac{5}{10} * \log_2\left(\frac{5}{10}\right) - \frac{5}{10} * \log_2\left(\frac{5}{10}\right) = 1$$

Ahora vamos a decidir la primera separación con las edades $H = \frac{3}{10} \cdot 0 + \frac{4}{10} \cdot 1 + \frac{3}{10} \cdot 0 = 0.4$

Ahora vamos a decidir la primera separación con la competencia $H = \frac{4}{10} \cdot 0.811 + \frac{6}{10} \cdot 0.918 = 0.8752$

Ahora vamos a decidir la primera separación con las edades $H = \frac{4}{10} \cdot 0.811 + \frac{6}{10} \cdot 0.918 = 0.8752$

Ahora vamos a decidir la primera separación con el tipo $H = \frac{6}{10} \cdot 1 + \frac{4}{10} \cdot 1 = 1$

Concluimos que lo que nos da la máxima ganancia de información es primero decidir por edades, eso nos deja dos nodos hoja y un nodo rama que debemos volver a separar.

Ahora vamos a buscar el segundo nivel, donde vamos a separar el grupo que tiene edades medias por competencia:

$$H = \frac{2}{4} \cdot 0 + \frac{2}{4} \cdot 0 = 0$$

Con esto ya se clasificaron todos los datos, puesto que terminamos solo con nodos hojas:

Esto también se puede hacer con valores numéricos, que de hecho, es lo que se puede hacer con scikit learn

y con esto se obtiene este árbol de decisión:

4.3.4 Comandos básicos en python

Estos son los comandos básicos en python

```
#| label: dibujoArbol01
#| fig-cap: "Árbol de decisión"
from sklearn import tree
X = # Lista con los features (lista de listas)
Y = # Lista con los labels
# Se define la variable que tendrá el árbol
clf = tree.DecisionTreeClassifier()
# Se calcula el árbol
clf = clf.fit(X, Y)
# Se utiliza el árbol para predecir el label de un dato nuevo
clf.predict_proba(X0)
# Se dibuja el árbol
tree.plot_tree(clf)
```

y este sería un ejemplo sencillo en python:

Primero creamos los datos

```
from sklearn import tree
from sklearn.datasets import make_blobs
from sklearn.inspection import DecisionBoundaryDisplay
import matplotlib.pyplot as plt
import numpy as np

# Creación de los datos
X, Y = make_blobs(n_samples=200, centers=4, random_state=6)
plt.scatter(X[:, 0], X[:, 1], c=Y, s=30)
plt.title("Datos originales")
```

```
plt.xlabel("x1")
plt.ylabel("x2")
plt.show()
```

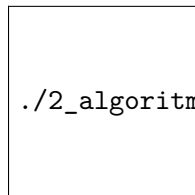


Figura 4.1: Ejemplo hecho en python: datos

Luego se crea el arbol

```
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, Y)
tree.plot_tree(clf)
plt.show()
```

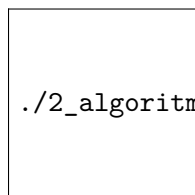


Figura 4.2: Ejemplo hecho en python: arbol

y por último, dibujamos las separaciones

```
DecisionBoundaryDisplay.from_estimator(clf, X, response_method="predict")
plt.scatter(X[:, 0], X[:, 1], c=Y, s=30)
plt.show()
```

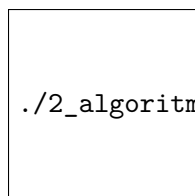


Figura 4.3: Ejemplo hecho en python: separación

y con esto se puede aplicar el árbol

```
print(clf.predict([[5.0, 1.0]]))  
print(clf.predict([[-2.0, -1.0]]))  
print(clf.predict([[6.0, -6.0]]))
```

[0]

[3]

[0]

y lo que devuelve es el número de grupo al que pertenece el dato

4.4 K-Nearest Neighbors (KNN)

4.4.1 Carga de paquetes

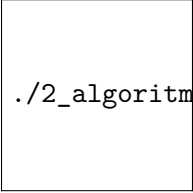
```
import matplotlib.pyplot as plt  
from sklearn.preprocessing import StandardScaler  
from sklearn.model_selection import train_test_split  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.metrics import accuracy_score  
from sklearn.inspection import DecisionBoundaryDisplay  
  
import pandas as pd  
import seaborn as sns  
from sklearn.datasets import make_blobs
```

5 Cargas datos

```
X, y = make_blobs(n_samples=1000, centers=3, random_state=6)
```

6 Visualizar los datos

```
sns.scatterplot(x=X[:,0],y=X[:,1], hue=y)
plt.show()
```



./2_algoritmos_fundamentales_files/figure-pdf/cell-8-output

7 Se normaliza y se divide los datos

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y)

# Scale the features using StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```


8 Ajuste y evaluación del modelo

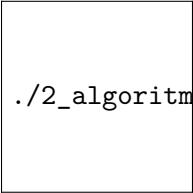
```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# predecir con el modelo
y_pred = knn.predict(X_test)

# evaluarlo
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Accuracy: 1.0

```
DecisionBoundaryDisplay.from_estimator(knn, X_train)
sns.scatterplot(x=X[:,0],y=X[:,1], hue=y)
plt.show()
```



./2_algoritmos_fundamentales_files/figure-pdf/cell-11-outpu