modelos_supervisados

March 2, 2023

[ ]:

[1]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
```

[ ]:

1 1. Modelos de regresión

[2]:
```
# Cargamos el dta

california = fetch_california_housing()
california_df = pd.DataFrame(california.data, columns=california.feature_names)
california_df['target'] = california.target
california_df
```

[2]:
```
        MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0       8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1       8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2       7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85
3       5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85
4       3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85
…        …       …        …         …          …           …         …
20635   1.5603      25.0  5.045455   1.133333       845.0  2.560606     39.48
20636   2.5568      18.0  6.114035   1.315789       356.0  3.122807     39.49
20637   1.7000      17.0  5.205543   1.120092      1007.0  2.325635     39.43
20638   1.8672      18.0  5.329513   1.171920       741.0  2.123209     39.43
```

```
20639  2.3886      16.0  5.254717   1.162264      1387.0  2.616981     39.37

       Longitude  target
0        -122.23   4.526
1        -122.22   3.585
2        -122.24   3.521
3        -122.25   3.413
4        -122.25   3.422
…           …      …
20635    -121.09   0.781
20636    -121.21   0.771
20637    -121.22   0.923
20638    -121.32   0.847
20639    -121.24   0.894

[20640 rows x 9 columns]
```

[3]:
```
# Dividimos en train y test

X_train, X_test, y_train, y_test = train_test_split(california_df.
 ↪drop('target', axis=1), california_df['target'], test_size=0.2,␣
 ↪random_state=42)
```

1.0.1 1. Regresión Lineal

[4]:
```
# Creamos la regresión y ajustamos el train

model = LinearRegression()
model.fit(X_train, y_train)
```

[4]:
```
LinearRegression()
```

[5]:
```
# Evaluamos el resultado

y_pred = model.predict(X_test)
print("MSE: ", mean_squared_error(y_test, y_pred))
print("R^2: ", r2_score(y_test, y_pred))
```

```
MSE:  0.5558915986952437
R^2:  0.5757877060324512
```

1.0.2 1. SVM

Aquí estamos utilizando SVR() de Scikit-Learn para crear un modelo de regresión de vectores de soporte (SVM). Especificamos el kernel lineal (kernel='linear'), lo que significa que nuestro modelo es un SVM lineal. Luego ajustamos el modelo a los datos de entrenamiento utilizando fit(), y utilizamos predict() para hacer predicciones en los datos de prueba. Finalmente, calculamos el error cuadrático medio (mean_squared_error()) y el coeficiente de determinación (r2_score()) para evaluar el rendimiento del modelo.

[ ]:
```
svm = SVR(kernel='linear') # en este caso, usamos un modelo lineal
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean squared error:", mse)
print("Coefficient of determination (R^2):", r2)
```

[ ]:
```

```

1.0.3 2. Arbol de decision

En este código, estamos utilizando DecisionTreeRegressor() de Scikit-Learn para crear un modelo de árbol de decisión. Especificamos la profundidad máxima del árbol (max_depth=10) y una semilla aleatoria (random_state=42). Luego ajustamos el modelo a los datos de entrenamiento utilizando fit(), y utilizamos predict() para hacer predicciones en los datos de prueba. Finalmente, calculamos el error cuadrático medio (mean_squared_error()) y el coeficiente de determinación (r2_score()) para evaluar el rendimiento del modelo.

[6]:
```
dt = DecisionTreeRegressor(max_depth=10, random_state=42)
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean squared error:", mse)
print("Coefficient of determination (R^2):", r2)
```

```
Mean squared error: 0.4154681981618525
Coefficient of determination (R^2): 0.6829476865157171
```

[ ]:
```

```

[ ]:
```

```

1.0.4 3. Random Forest

En este código, estamos utilizando RandomForestRegressor() de Scikit-Learn para crear un modelo de bosque aleatorio. Especificamos el número de árboles (n_estimators=100), la profundidad máxima de cada árbol (max_depth=10) y una semilla aleatoria (random_state=42). Luego ajustamos el modelo a los datos de entrenamiento utilizando fit(), y utilizamos predict() para hacer predicciones en los datos de prueba. Finalmente, calculamos el error cuadrático medio (mean_squared_error()) y el coeficiente de determinación (r2_score()) para evaluar el rendimiento del modelo.

[7]:
```
rf = RandomForestRegressor(n_estimators=100, max_depth=10, random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean squared error:", mse)
print("Coefficient of determination (R^2):", r2)
```

```
Mean squared error: 0.2965447516723708
Coefficient of determination (R^2): 0.7737006105754448
```

[ ]:

[ ]:

1.0.5 4. Gradient Boosting

En este código, estamos utilizando GradientBoostingRegressor() de Scikit-Learn para crear un modelo de gradient boosting. Especificamos el número de árboles (n_estimators=100), la profundidad máxima de cada árbol (max_depth=5) y una semilla aleatoria (random_state=42). Luego ajustamos el modelo a los datos de entrenamiento utilizando fit(), y utilizamos predict() para hacer predicciones en los datos de prueba. Finalmente, calculamos el error cuadrático medio (mean_squared_error()) y el coeficiente de determinación (r2_score()) para evaluar el rendimiento del modelo.

[8]:
```
gb = GradientBoostingRegressor(n_estimators=100, max_depth=5, random_state=42)
gb.fit(X_train, y_train)
y_pred = gb.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean squared error:", mse)
print("Coefficient of determination (R^2):", r2)
```

```
Mean squared error: 0.24765057253278291
Coefficient of determination (R^2): 0.8110127626985352
```

[ ]:

2 2. Modelos de clasificación

[16]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
```

[11]:
```
# Cargamos el dta de clasificación
# Cargar el conjunto de datos
california = load_breast_cancer()
california_df = pd.DataFrame(california.data, columns=california.feature_names)
california_df['target'] = california.target
breast_cancer = california_df
breast_cancer
```

[11]:
```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0          17.99         10.38          122.80     1001.0          0.11840
1          20.57         17.77          132.90     1326.0          0.08474
2          19.69         21.25          130.00     1203.0          0.10960
3          11.42         20.38           77.58      386.1          0.14250
4          20.29         14.34          135.10     1297.0          0.10030
..           …             …               …         …               …
564        21.56         22.39          142.00     1479.0          0.11100
565        20.13         28.25          131.20     1261.0          0.09780
566        16.60         28.08          108.30      858.1          0.08455
567        20.60         29.33          140.10     1265.0          0.11780
568         7.76         24.54           47.92      181.0          0.05263

     mean compactness  mean concavity  mean concave points  mean symmetry  \
0             0.27760         0.30010              0.14710         0.2419
1             0.07864         0.08690              0.07017         0.1812
2             0.15990         0.19740              0.12790         0.2069
3             0.28390         0.24140              0.10520         0.2597
4             0.13280         0.19800              0.10430         0.1809
..                …               …                    …              …
564           0.11590         0.24390              0.13890         0.1726
565           0.10340         0.14400              0.09791         0.1752
566           0.10230         0.09251              0.05302         0.1590
567           0.27700         0.35140              0.15200         0.2397
568           0.04362         0.00000              0.00000         0.1587
```

```
     mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
0                   0.07871  ...          17.33           184.60      2019.0
1                   0.05667  ...          23.41           158.80      1956.0
2                   0.05999  ...          25.53           152.50      1709.0
3                   0.09744  ...          26.50            98.87       567.7
4                   0.05883  ...          16.67           152.20      1575.0
..                      ...  ...            ...              ...         ...
564                 0.05623  ...          26.40           166.10      2027.0
565                 0.05533  ...          38.25           155.00      1731.0
566                 0.05648  ...          34.12           126.70      1124.0
567                 0.07016  ...          39.42           184.60      1821.0
568                 0.05884  ...          30.37            59.16       268.6

     worst smoothness  worst compactness  worst concavity  \
0             0.16220            0.66560           0.7119
1             0.12380            0.18660           0.2416
2             0.14440            0.42450           0.4504
3             0.20980            0.86630           0.6869
4             0.13740            0.20500           0.4000
..                ...                ...              ...
564           0.14100            0.21130           0.4107
565           0.11660            0.19220           0.3215
566           0.11390            0.30940           0.3403
567           0.16500            0.86810           0.9387
568           0.08996            0.06444           0.0000

     worst concave points  worst symmetry  worst fractal dimension  target
0                  0.2654          0.4601                  0.11890       0
1                  0.1860          0.2750                  0.08902       0
2                  0.2430          0.3613                  0.08758       0
3                  0.2575          0.6638                  0.17300       0
4                  0.1625          0.2364                  0.07678       0
..                    ...             ...                      ...     ...
564                0.2216          0.2060                  0.07115       0
565                0.1628          0.2572                  0.06637       0
566                0.1418          0.2218                  0.07820       0
567                0.2650          0.4087                  0.12400       0
568                0.0000          0.2871                  0.07039       1

[569 rows x 31 columns]
```

[13]:
```
# Dividimos test y train

X_train, X_test, y_train, y_test = train_test_split(breast_cancer.
↪drop('target', axis=1), breast_cancer['target'], test_size=0.2,␣
↪random_state=42)
```

```
[13]:      mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
      68         9.029         17.33           58.79      250.5          0.10660
      181       21.090         26.57          142.70     1311.0          0.11410
      63         9.173         13.86           59.20      260.9          0.07721
      248       10.650         25.22           68.01      347.0          0.09657
      60        10.170         14.88           64.55      311.9          0.11340
      ..           …             …              …          …               …
      71         8.888         14.64           58.79      244.0          0.09783
      106       11.640         18.33           75.17      412.5          0.11420
      270       14.290         16.82           90.30      632.6          0.06429
      435       13.980         19.62           91.12      599.5          0.10600
      102       12.180         20.52           77.22      458.7          0.08013

           mean compactness  mean concavity  mean concave points  mean symmetry  \
      68            0.14130         0.31300              0.04375         0.2111
      181           0.28320         0.24870              0.14960         0.2395
      63            0.08751         0.05988              0.02180         0.2341
      248           0.07234         0.02379              0.01615         0.1897
      60            0.08061         0.01084              0.01290         0.2743
      ..                …               …                    …              …
      71            0.15310         0.08606              0.02872         0.1902
      106           0.10170         0.07070              0.03485         0.1801
      270           0.02675         0.00725              0.00625         0.1508
      435           0.11330         0.11260              0.06463         0.1669
      102           0.04038         0.02383              0.01770         0.1739

           mean fractal dimension  …  worst radius  worst texture  \
      68                  0.08046  …        10.310          22.65
      181                 0.07398  …        26.680          33.48
      63                  0.06963  …        10.010          19.23
      248                 0.06329  …        12.250          35.19
      60                  0.06960  …        11.020          17.45
      ..                      …    …           …              …
      71                  0.08980  …         9.733          15.67
      106                 0.06520  …        13.140          29.26
      270                 0.05376  …        14.910          20.65
      435                 0.06544  …        17.040          30.80
      102                 0.05677  …        13.340          32.84

           worst perimeter  worst area  worst smoothness  worst compactness  \
      68             65.50       324.7           0.14820            0.43650
      181           176.50      2089.0           0.14910            0.75840
      63             65.59       310.1           0.09836            0.16780
      248            77.98       455.7           0.14990            0.13980
      60             69.86       368.6           0.12750            0.09866
      ..               …           …                 …                 …
      71             62.56       284.4           0.12070            0.24360
```

```
106          85.51      521.7          0.16880             0.26600
270          94.44      684.6          0.08567             0.05036
435         113.90      869.3          0.16130             0.35680
102          84.58      547.8          0.11230             0.08862

     worst concavity  worst concave points  worst symmetry  \
68           1.25200               0.17500          0.4228
181          0.67800               0.29030          0.4098
63           0.13970               0.05087          0.3282
248          0.11250               0.06136          0.3409
60           0.02168               0.02579          0.3557
..               …                     …              …
71           0.14340               0.04786          0.2254
106          0.28730               0.12180          0.2806
270          0.03866               0.03333          0.2458
435          0.40690               0.18270          0.3179
102          0.11450               0.07431          0.2694

     worst fractal dimension
68                   0.11750
181                  0.12840
63                   0.08490
248                  0.08147
60                   0.08020
..                       …
71                   0.10840
106                  0.09097
270                  0.06120
435                  0.10550
102                  0.06878

[455 rows x 30 columns]
```

[ ]:

2.0.1 1. Regresión Logisticas

[17]:

```
lr = LogisticRegression(random_state=42)
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", acc)
```

```
print("Precision:", prec)
print("Recall:", rec)
print("F1 Score:", f1)
```

```
Accuracy: 0.9736842105263158
Precision: 0.9722222222222222
Recall: 0.9859154929577465
F1 Score: 0.979020979020979
```

```
/Users/adrian_gr/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[18]:
```
cm = confusion_matrix(y_test, y_pred)
print("Matriz de Confusión:")
print(cm)
```

```
Matriz de Confusión:
[[41  2]
 [ 1 70]]
```

[ ]:
```
```

2.0.2 2. Arbol de decision

[19]:
```
dt = DecisionTreeClassifier(max_depth=5, random_state=42)
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", acc)
print("Precision:", prec)
print("Recall:", rec)
print("F1 Score:", f1)
```

```
Accuracy: 0.9473684210526315
```

```
Precision: 0.9577464788732394
Recall: 0.9577464788732394
F1 Score: 0.9577464788732394
```

[20]:
```python
cm = confusion_matrix(y_test, y_pred)
print("Matriz de Confusión:")
print(cm)
```

```
Matriz de Confusión:
[[40  3]
 [ 3 68]]
```

[ ]:
```python
```

2.0.3 3. Random Forest

[21]:
```python
rf = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", acc)
print("Precision:", prec)
print("Recall:", rec)
print("F1 Score:", f1)
```

```
Accuracy: 0.9649122807017544
Precision: 0.958904109589041
Recall: 0.9859154929577465
F1 Score: 0.9722222222222222
```

[22]:
```python
cm = confusion_matrix(y_test, y_pred)
print("Matriz de Confusión:")
print(cm)
```

```
Matriz de Confusión:
[[40  3]
 [ 1 70]]
```

[ ]:
```python
```

2.0.4 4. Gradiant Boosting

[23]:
```
gb = GradientBoostingClassifier(n_estimators=100, max_depth=5, random_state=42)
gb.fit(X_train, y_train)
y_pred = gb.predict(X_test)

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", acc)
print("Precision:", prec)
print("Recall:", rec)
print("F1 Score:", f1)
```

```
Accuracy: 0.9649122807017544
Precision: 0.958904109589041
Recall: 0.9859154929577465
F1 Score: 0.9722222222222222
```

[24]:
```
cm = confusion_matrix(y_test, y_pred)
print("Matriz de Confusión:")
print(cm)
```

```
Matriz de Confusión:
[[40  3]
 [ 1 70]]
```

[ ]:
```
```

[ ]:
```
```