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

```
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

```
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}
```