DT con Digits normalizando

June 12, 2018

1 Clasificación con DecisionTree con el dataset Digits de scikit-learn normalizando los datos

http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html

1.1 Información del dataset

Casi 1800 imágenes de 8x8 con los dígitos 0..9. Cada píxel se representa con un número entre 0..15, que representa un color en la escala de grises.

En este notebook se hará el training con los datos normalizados, igual que hacen en algunos ejemplo de scikit-learn (dividen cada fila por 16 y luego le restan la media). En este notebook el training se hará directamente con los datos normalizados, sin usar ningún tipo de sampler. Para ver el mismo código pero usando RBFSampler, estará en otro notebook.

```
In [1]: from sklearn.datasets import load_digits
        from sklearn.tree import DecisionTreeClassifier
```

```
In [2]: import math
        import numpy as np
```

```
In [3]: digits = load_digits()
```

```
In [4]: data = digits.data
        target = digits.target
        N = data.shape[0]
        prop_train = 2 / 3
        N_train = math.ceil(N * prop_train)
        N_test = N - N_train
```

Normaliza los datos entre 0 y 1 y luego los centra en la media

```
In [5]: data /= 16
        data -= data.mean(axis = 0)
```

Los primeros 2/3 de los datos son de train, y el resto son de test

```
In [6]: data_train = data[:N_train]
        data_test = data[N_train:]

        target_train = target[:N_train]
        target_test = target[N_train:]
```

1.2 Hacer n_runs ejecuciones y devolver la media de puntuación de todas ellas

```
In [7]: n_runs = 50
        train_scores = []
        test_scores = []
```

```
In [8]: for i in range(n_runs):
            clf = DecisionTreeClassifier()
            clf.fit(data_train, target_train)
            train_score = clf.score(data_train, target_train)
            test_score = clf.score(data_test, target_test)
            train_scores.append(train_score)
            test_scores.append(test_score)
```

```
In [9]: print("Mean of test scores:",np.mean(test_scores))
        print("Mean of train scores:", np.mean(train_scores))
```

```
Mean of test scores: 0.77889816360601
Mean of train scores: 1.0
```

```
In [10]: print("Standard deviation of test scores:",np.std(test_scores))
         print("Standard deviation of train scores:",np.std(train_scores))
```

```
Standard deviation of test scores: 0.008533220410195561
Standard deviation of train scores: 0.0
```

1.2.1 Conclusiones

El árbol es capaz de ajustar muy bien. También es capaz de aprender, pero no generaliza demasiado, sobreajusta. De todos modos, es mejor que tirar una moneda. No parece haber ninguna mejoría entre normalizar los datos y no hacerlo.