DT con Digits normalizando y usando RBFSampler

June 12, 2018

1 Clasificación con DecisionTree con el dataset Digits de scikit-learn normalizando los datos y después usando RBFSampler para 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 es 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) y después se usará el RBFSampler para generar datos nuevos. Para ver otras variantes, estará en otro notebook.

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

In [2]: import math
        import numpy as np

In [3]: import matplotlib.pyplot as plt

In [4]: digits = load_digits()

In [5]: 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

In [6]: data /= 16
        data -= data.mean(axis = 0)

In [7]: sampler = RBFSampler(n_components = 1000)

In [8]: sampler.fit(data)
```

```
Out[8]: RBFSampler(gamma=1.0, n_components=1000, random_state=None)

In [9]: data = sampler.transform(data)

In [10]: data_train = data[:N_train]
         data_test = data[N_train:]

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

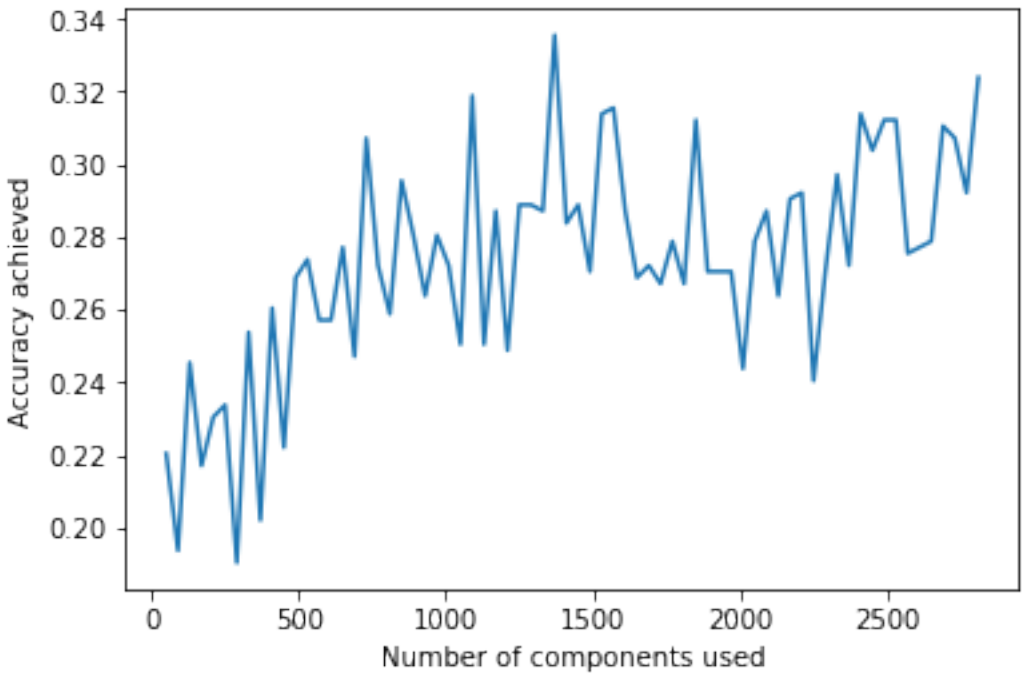
1.2 Varias ejecuciones y hacer la media

```
In [11]: n_runs = 5
         train_scores = []
         test_scores = []

In [12]: 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 [13]: print("Mean of test scores:",np.mean(test_scores))
         print("Mean of train scores:", np.mean(train_scores))

Mean of test scores: 0.2300500834724541
Mean of train scores: 1.0


In [14]: 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: 0.0038650540576929014
Standard deviation of train scores: 0.0
```

1.3 £Como cambia accuracy si incrementamos la cantidad de features extraídas?

```
In [15]: digits = load_digits()

In [16]: 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
```

```
In [17]: data /= 16
         data -= data.mean(axis = 0)

In [18]: target_train = target[:N_train]
         target_test = target[N_train:]

In [19]: components = 40 * np.arange(70) + 50
         scores = []
         n_runs = 7

In [20]: for comp in components:
             sampler = RBFSampler(n_components=comp)
             sampler.fit(data)
             dat = sampler.transform(data)
             dat_train = dat[:N_train]
             dat_test = dat[N_train:]
             clf = DecisionTreeClassifier()
             clf.fit(dat_train, target_train)
             sc = 0
             for i in range(n_runs):
                 sc += clf.score(dat_test, target_test)
             sc /= n_runs
             scores.append(sc)

In [21]: plt.plot(components, scores)
         plt.xlabel("Number of components used")
         plt.ylabel("Accuracy achieved")

Out[21]: Text(0,0.5,'Accuracy achieved')
```



```
In [22]: print("Maximum achieved score:",np.max(scores))
         print("Minimum achieved score:",np.min(scores))
```

```
Maximum achieved score: 0.335559265442404
Minimum achieved score: 0.19031719532554253
```

1.3.1 Conclusiones

Sí que se aprende algo, es mejor que tirar una moneda, pero la precisión sigue siendo bastante mala. Quizá RandomForest puede mejorar esto. Parece que incrementar la cantidad de features extraídos contribuye a mejorar la precisión, aunque parece tener mucha varianza