cross_validation_grouping

May 29, 2021

1 Sample grouping

We are going to linger into the concept of sample groups. As in the previous section, we will give an example to highlight some surprising results. This time, we will use the handwritten digits dataset.

[ ]:
```
from sklearn.datasets import load_digits

digits = load_digits()
data, target = digits.data, digits.target
```

We will recreate the same model used in the previous exercise: a logistic regression classifier with preprocessor to scale the data.

[ ]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(), LogisticRegression())
```

We will use the same baseline model. We will use a `KFold` cross-validation without shuffling the data at first.

[ ]:
```
from sklearn.model_selection import cross_val_score, KFold

cv = KFold(shuffle=False)
test_score_no_shuffling = cross_val_score(model, data, target, cv=cv,
                                          n_jobs=-1)
print(f"The average accuracy is "
      f"{test_score_no_shuffling.mean():.3f} +/- "
      f"{test_score_no_shuffling.std():.3f}")
```

Now, let's repeat the experiment by shuffling the data within the cross-validation.

[ ]:
```
cv = KFold(shuffle=True)
test_score_with_shuffling = cross_val_score(model, data, target, cv=cv,
                                            n_jobs=-1)
print(f"The average accuracy is "
      f"{test_score_with_shuffling.mean():.3f} +/- "
      f"{test_score_with_shuffling.std():.3f}")
```

We observe that shuffling the data improves the mean accuracy. We could go a little further and plot the distribution of the testing score. We can first concatenate the test scores.

[ ]:
```
import pandas as pd

all_scores = pd.DataFrame(
    [test_score_no_shuffling, test_score_with_shuffling],
    index=["KFold without shuffling", "KFold with shuffling"],
).T
```

Let's plot the distribution now.

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

all_scores.plot.hist(bins=10, edgecolor="black", density=True, alpha=0.7)
plt.xlim([0.8, 1.0])
plt.xlabel("Accuracy score")
plt.legend(bbox_to_anchor=(1.05, 0.8), loc="upper left")
_ = plt.title("Distribution of the test scores")
```

The cross-validation testing error that uses the shuffling has less variance than the one that does not impose any shuffling. It means that some specific fold leads to a low score in this case.

[ ]:
```
print(test_score_no_shuffling)
```

Thus, there is an underlying structure in the data that shuffling will break and get better results. To get a better understanding, we should read the documentation shipped with the dataset.

[ ]:
```
print(digits.DESCR)
```

If we read carefully, 13 writers wrote the digits of our dataset, accounting for a total amount of 1797 samples. Thus, a writer wrote several times the same numbers. Let's suppose that the writer samples are grouped. Subsequently, not shuffling the data will keep all writer samples together either in the training or the testing sets. Mixing the data will break this structure, and therefore digits written by the same writer will be available in both the training and testing sets.

Besides, a writer will usually tend to write digits in the same manner. Thus, our model will learn to identify a writer's pattern for each digit instead of recognizing the digit itself.

We can solve this problem by ensuring that the data associated with a writer should either belong to the training or the testing set. Thus, we want to group samples for each writer.

Here, we will manually define the group for the 13 writers.

[ ]:
```
from itertools import count
import numpy as np

# defines the lower and upper bounds of sample indices
# for each writer
```

```
writer_boundaries = [0, 130, 256, 386, 516, 646, 776, 915, 1029,
                     1157, 1287, 1415, 1545, 1667, 1797]
groups = np.zeros_like(target)
lower_bounds = writer_boundaries[:-1]
upper_bounds = writer_boundaries[1:]

for group_id, lb, up in zip(count(), lower_bounds, upper_bounds):
    groups[lb:up] = group_id
```

We can check the grouping by plotting the indices linked to writer ids.

[ ]:
```
plt.plot(groups)
plt.yticks(np.unique(groups))
plt.xticks(writer_boundaries, rotation=90)
plt.xlabel("Target index")
plt.ylabel("Writer index")
_ = plt.title("Underlying writer groups existing in the target")
```

Once we group the digits by writer, we can use cross-validation to take this information into account: the class containing `Group` should be used.

[ ]:
```
from sklearn.model_selection import GroupKFold

cv = GroupKFold()
test_score = cross_val_score(model, data, target, groups=groups, cv=cv,
                             n_jobs=-1)
print(f"The average accuracy is "
      f"{test_score.mean():.3f} +/- "
      f"{test_score.std():.3f}")
```

We see that this strategy is less optimistic regarding the model statistical performance. However, this is the most reliable if our goal is to make handwritten digits recognition writers independent. Besides, we can as well see that the standard deviation was reduced.

[ ]:
```
all_scores = pd.DataFrame(
    [test_score_no_shuffling, test_score_with_shuffling, test_score],
    index=["KFold without shuffling", "KFold with shuffling",
           "KFold with groups"],
).T
```

[ ]:
```
all_scores.plot.hist(bins=10, edgecolor="black", density=True, alpha=0.7)
plt.xlim([0.8, 1.0])
plt.xlabel("Accuracy score")
plt.legend(bbox_to_anchor=(1.05, 0.8), loc="upper left")
_ = plt.title("Distribution of the test scores")
```

As a conclusion, it is really important to take any sample grouping pattern into account when evaluating a model. Otherwise, the results obtained will be over-optimistic in regards with reality.