cross_validation_ex_01

June 6, 2024

1 Exercise M2.01

The aim of this exercise is to make the following experiments:

- train and test a support vector machine classifier through cross-validation;
- study the effect of the parameter gamma of this classifier using a validation curve;
- use a learning curve to determine the usefulness of adding new samples in the dataset when building a classifier.

To make these experiments we first load the blood transfusion dataset.

Note

If you want a deeper overview regarding this dataset, you can refer to the Appendix - Datasets description section at the end of this MOOC.

[2]:
```python
import pandas as pd

blood_transfusion = pd.read_csv("../datasets/blood_transfusion.csv")
data = blood_transfusion.drop(columns="Class")
target = blood_transfusion["Class"]
```

Here we use a support vector machine classifier (SVM). In its most simple form, a SVM classifier is a linear classifier behaving similarly to a logistic regression. Indeed, the optimization used to find the optimal weights of the linear model are different but we don't need to know these details for the exercise.

Also, this classifier can become more flexible/expressive by using a so-called kernel that makes the model become non-linear. Again, no requirement regarding the mathematics is required to accomplish this exercise.

We will use an RBF kernel where a parameter `gamma` allows to tune the flexibility of the model.

First let's create a predictive pipeline made of:

- a `sklearn.preprocessing.StandardScaler` with default parameter;
- a `sklearn.svm.SVC` where the parameter `kernel` could be set to `"rbf"`. Note that this is the default.

[17]:
```python
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVC
```

```
pipeline = make_pipeline(StandardScaler(), SVC(kernel='rbf'))#default
```

Evaluate the generalization performance of your model by cross-validation with a `ShuffleSplit` scheme. Thus, you can use `sklearn.model_selection.cross_validate` and pass a `sklearn.model_selection.ShuffleSplit` to the `cv` parameter. Only fix the `random_state=0` in the `ShuffleSplit` and let the other parameters to the default.

[21]:
```
from sklearn.model_selection import cross_validate, ShuffleSplit

cv = ShuffleSplit(random_state=0)
cross_validation = cross_validate(pipeline, data, target, cv=cv)
cv_results = pd.DataFrame(cross_validation)
cv_results.head()
```

[21]:
```
   fit_time  score_time  test_score
0  0.030830    0.004443    0.680000
1  0.024393    0.003870    0.746667
2  0.026360    0.004577    0.786667
3  0.024811    0.004135    0.800000
4  0.025568    0.003937    0.746667
```

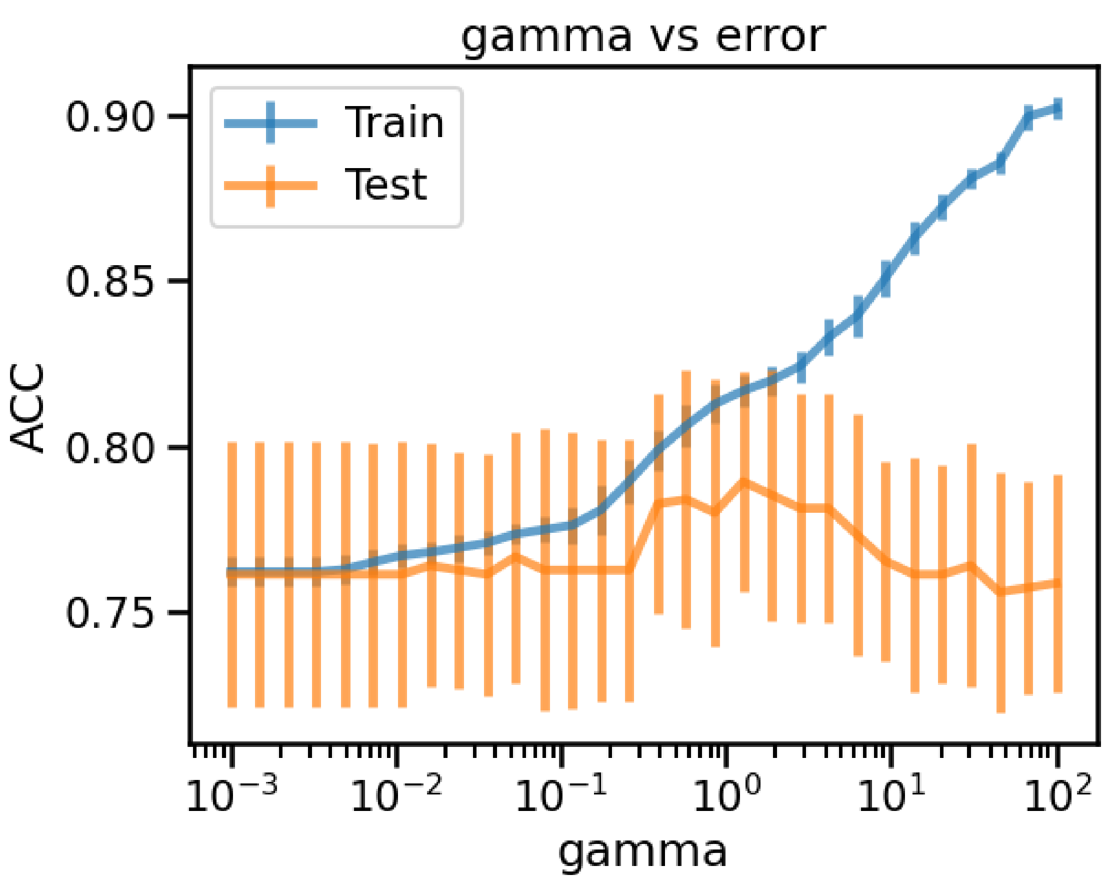
As previously mentioned, the parameter `gamma` is one of the parameters controlling under/over-fitting in support vector machine with an RBF kernel.

Evaluate the effect of the parameter `gamma` by using `sklearn.model_selection.ValidationCurveDisplay`. You can leave the default `scoring=None` which is equivalent to `scoring="accuracy"` for classification problems. You can vary `gamma` between `10e-3` and `10e2` by generating samples on a logarithmic scale with the help of `np.logspace(-3, 2, num=30)`.

Since we are manipulating a `Pipeline` the parameter name is `svc__gamma` instead of only `gamma`. You can retrieve the parameter name using `model.get_params().keys()`. We will go more into detail regarding accessing and setting hyperparameter in the next section.

[28]:
```
from sklearn.model_selection import ValidationCurveDisplay
import numpy as np
param_range = np.logspace(-3, 2, num=30)
display = ValidationCurveDisplay.from_estimator(pipeline, data, target,␣
 ↪param_name="svc__gamma", param_range = param_range, cv=cv, scoring=None,␣
 ↪std_display_style = "errorbar", n_jobs=2, errorbar_kw={"alpha":0.7})

_= display.ax_.set(xlabel="gamma", ylabel="ACC", title= "gamma vs error")
```

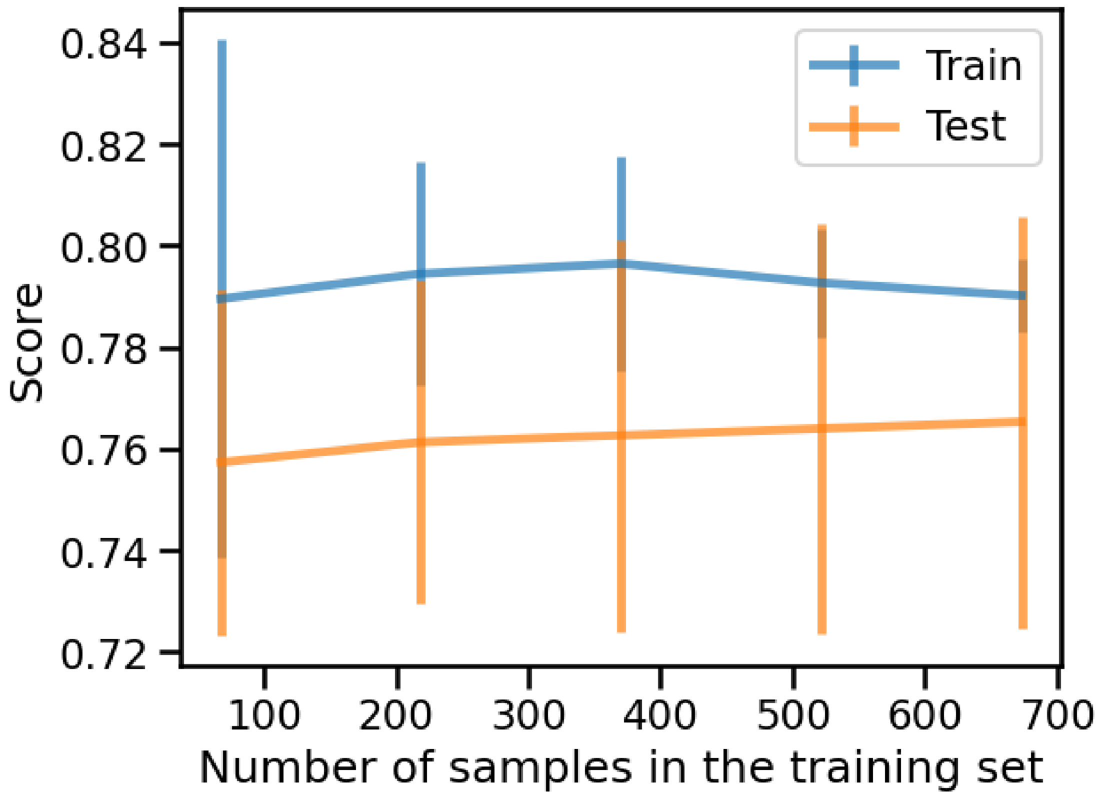


Now, you can perform an analysis to check whether adding new samples to the dataset could help our model to better generalize. Compute the learning curve (using `sklearn.model_selection.LearningCurveDisplay`) by computing the train and test scores for different training dataset size. Plot the train and test scores with respect to the number of samples.

[32]:
```python
from sklearn.model_selection import LearningCurveDisplay
import numpy as np
import matplotlib.pyplot as plt

train_sizes = np.linspace(0.1, 1.0, num=5, endpoint=True)

display = LearningCurveDisplay.from_estimator(pipeline, data, target,␣
  ↪train_sizes=train_sizes, cv=cv, n_jobs=2, random_state=0,␣
  ↪std_display_style="errorbar", errorbar_kw={"alpha":0.7})
plt.show()
```


Train
Test
Score
0.84
0.82
0.80
0.78
0.76
0.74
0.72
100
200
300
400
500
600
700
Number of samples in the training set

```
[ ]:
```