Pattern Recognition
and Applications Lab

Performance Evaluation and Hyperparameter Estimation

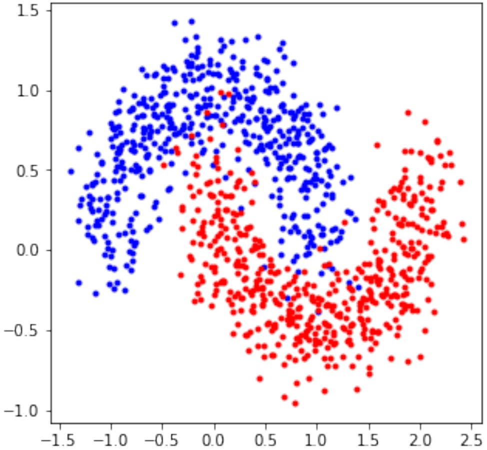
Machine Learning – Laboratory

Battista Biggio

Department of Electrical and Electronic Engineering
University of Cagliari, Italy

Running Example: Two-Moon Dataset

```
from sklearn.datasets import make_moons


x, y = make_moons(n_samples=1000,
                  noise=0.2,
                  random_state=None)
```

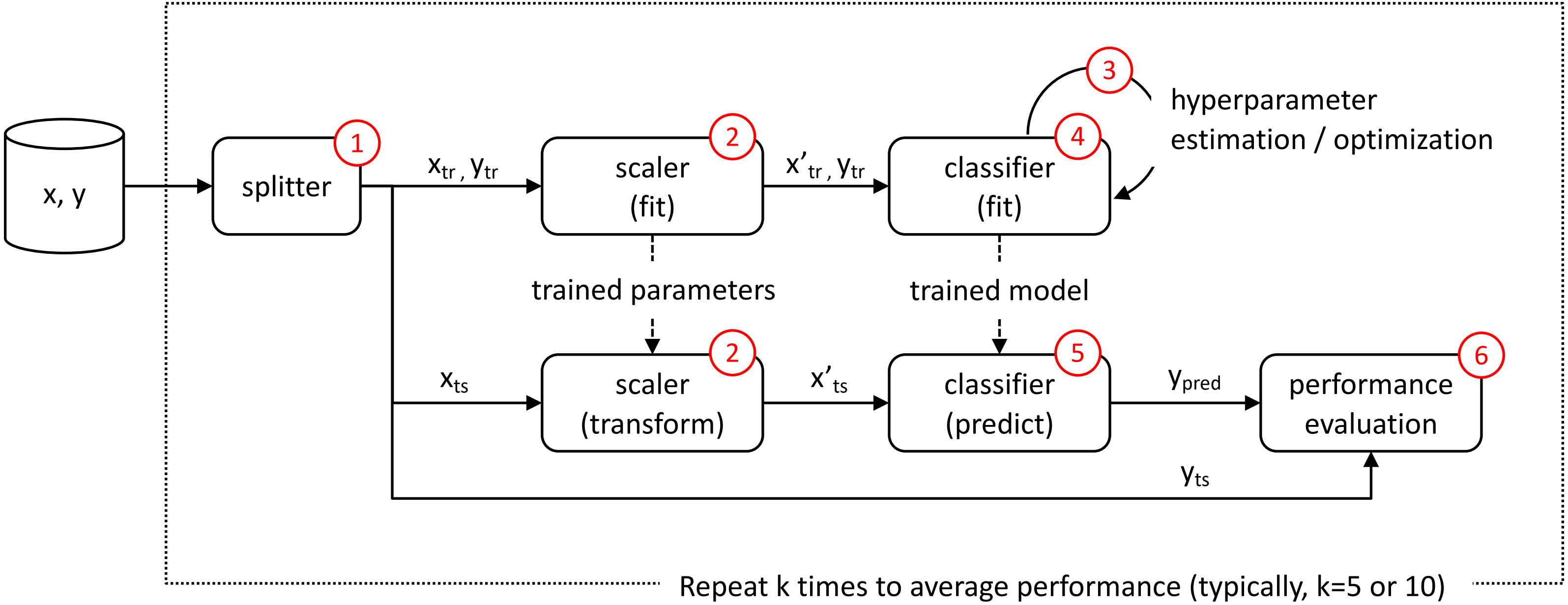


https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html

Running Example: MNIST Digit Classification

- Dataset of handwritten digits
- 10 classes: 0, 1, ..., 9
- 28x28 images corresponding to 784 feature (pixel) values
- Pixel values in {0, 1, ..., 255}

ML System Design



ML System Design

1. Sampling a training and a testing set (from the same underlying distribution)
   - e.g., splitting data `x,y` at random in `x_tr, y_tr, x_ts, y_ts`

2. Scaling training and test data (using parameters estimated on training data)
   - `from sklearn.preprocessing import MinMaxScaler`

3. Estimating classifier hyperparameters on training data

4. Fitting the classifier on training data
   - `clf.fit(x_tr, y_tr)`

5. Predicting the class labels of testing data
   - `clf.predict(x_ts)`

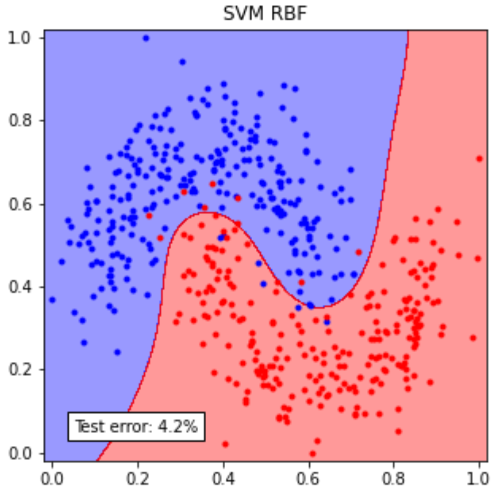
6. Evaluating accuracy or classification error

Exercise 1 – Performance Evaluation

- Implement a machine-learning pipeline to process MNIST digit data as described before
- Use default hyperparameters for some sklearn classifiers (e.g., SVM, kNN)
- Estimate classification error on different runs (i.e., data splits) and average the results
- What happens if we change the classifier hyperparameters?

Solution

```
x, y = make_moons(n_samples=1000, noise=0.2)

splitter = ShuffleSplit(n_splits=5, train_size=0.5)
scaler = MinMaxScaler()
clf = svm.SVC(kernel='rbf', C=10, gamma=10.0)
clf_name = 'SVM RBF’

for tr_idx, ts_idx in splitter.split(x, y):
    xtr = x[tr_idx, :]
    ytr = y[tr_idx]
    xts = x[ts_idx, :]
    yts = y[ts_idx]
    xtr = scaler.fit_transform(xtr)
    xts = scaler.transform(xts)
    clf.fit(xtr, ytr)
    ypred = clf.predict(xts)
    error = (ypred != yts).mean()
    print("Test error: {:.1%}".format(error))
```



```
Test error: 4.0%
Test error: 4.6%
...
Test error: 4.2%
```

Mean test error: 4.2% +/- 2.7%

Hyperparameter Estimation / Optimization

- How can we select the best hyperparameters for our classifier, i.e., the hyperparameters that we expect to perform best on the test set?

- **Remark:** we cannot look at the test set more than once. We cannot optimize the hyperparameters by iteratively testing how they perform on the test set.
  - This will lead us to overestimate the performance of the system in production!

- **Validation data.** The standard procedure is to optimize these hyperparameters using a (separate) *validation set* and compute their *expected* performance on such data

Hyperparameter Estimation / Optimization

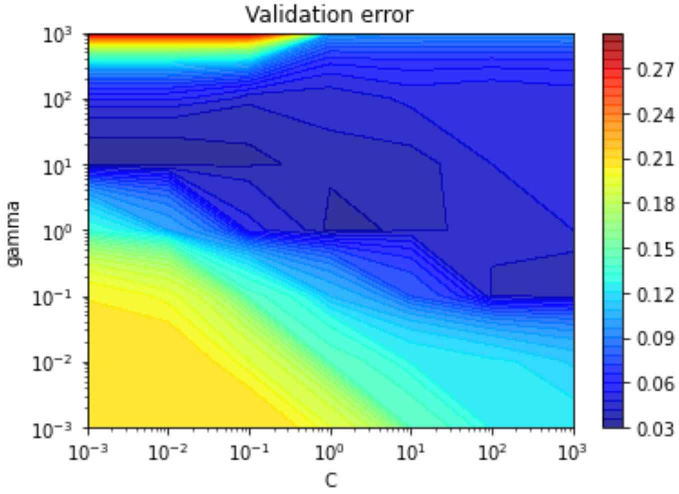
- In principle, hyperparameter optimization aims to solving the following (bi-level) optimization problem:

$$\min_{\gamma} \; L(D_{val}, \theta^*(\gamma))$$

Loss estimated on validation data

$$\text{s.t. } \theta^* = \text{argmin}_{\theta} \, \mathcal{L}(D_{tr}, \theta, \gamma)$$

Classifier training with the given hyperparameters



Open in Colab https://colab.research.google.com/github/unica-ml/ml/blob/master/notebooks/lab04.ipynb

Hyperparameter Estimation / Optimization

- Hyperparameter optimization is normally performed using grid search with k-fold cross validation (even though more complex optimization techniques can be used)

- The procedure works as follows:
  - One sets different hyperparameter values to test (the grid space)

    $$C \in \{10, 100, 1000\}$$
    $$\gamma \in \{0.1, 0.2, 0.5, 1.0\}$$

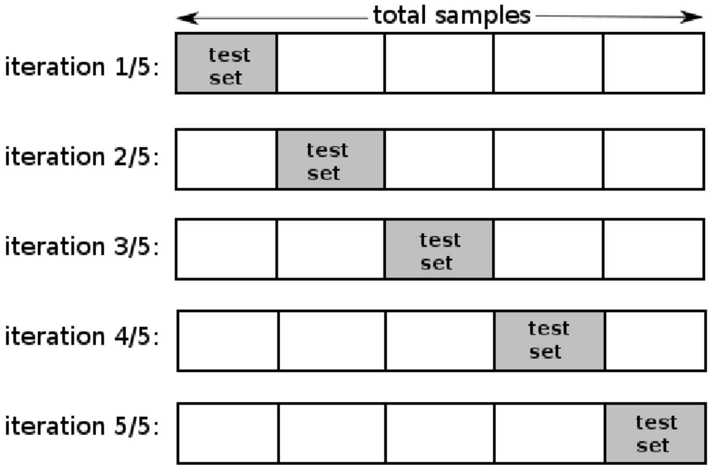
  - Then, for each set of hyperparameter values (e.g., C and gamma value pair):
    - Train the classifier on the training set with the current hyperparameters
    - Evaluate the performance on the validation set
    - Repeat K times on different training-validation splits, and average the performance

- At the end of the aforementioned procedure, for each set of hyperparameters we will get an averaged performance estimation (e.g., the mean validation loss/error)

- We can thus select the best set of hyperparameters as those that minimize the mean loss/error (or that maximize the mean validation accuracy)

Hyperparameter Optimization with k-Fold Cross Validation

- K-Fold Cross Validation
  - splits data into K folds
  - K-1 folds are used for training, 1 for validation
  - the process is repeated K times to estimate performance with a given set of classifier parameters (changing the validation fold at every iteration)



1. The process is repeated for every set of hyperparameters
2. The best set of hyperparameters is eventually selected
3. Classifier is trained using such parameters on the complete training set

Exercise 2 – Hyperparameter Optimization

- Solve again Exercise 1 but including hyperparameter optimization
  - For the linear SVM, optimize C
  - For the SVM with the RBF kernel, optimize C and gamma
  - For kNN, optimize k

- Use grid search with 5-fold cross validation to select the best hyperparameters that maximize the mean validation accuracy

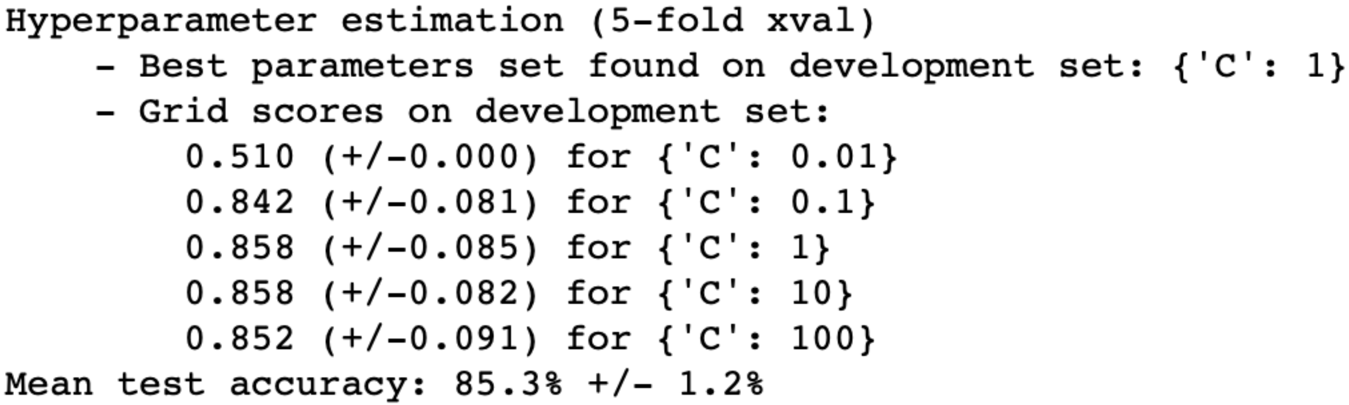
- **GridSearchCV** from sklearn already implements the whole procedure
  - https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Solution

```python
from sklearn.model_selection import GridSearchCV

# let's go back to the two-moon dataset
x, y = make_moons(n_samples=1000, noise=0.2)

clf = GridSearchCV(estimator=svm.SVC(kernel='linear'),
                   param_grid={'C': [0.01, 0.1, 1, 10, 100]})

acc = run(x, y, splitter, scaler, clf)

print("Hyperparameter estimation (5-fold xval)")
print("    - Best parameters set found on development set:", clf.best_params_)
print("    - Grid scores on development set:")
means = clf.cv_results_['mean_test_score']
stds = clf.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, clf.cv_results_['params']):
    print("        %0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))

print("Mean test accuracy: {:.1%} +/- {:.1%}\n".format(acc.mean(), 2*acc.std()))
```

Solution

- Optimization of the hyperparameter C of a linear SVM on the two-moon data
  - Note that we have mean validation accuracy estimated for each C value
  - The best value C=1 is selected and used to fit the classifier on the whole training set
  - Mean test accuracy is reasonably close to the best mean validation accuracy



```
Hyperparameter estimation (5-fold xval)
    - Best parameters set found on development set: {'C': 1}
    - Grid scores on development set:
        0.510 (+/-0.000) for {'C': 0.01}
        0.842 (+/-0.081) for {'C': 0.1}
        0.858 (+/-0.085) for {'C': 1}
        0.858 (+/-0.082) for {'C': 10}
        0.852 (+/-0.091) for {'C': 100}
Mean test accuracy: 85.3% +/- 1.2%
```