Exercises for the course
Machine Learning for Data Science
Winter Semester 2022/23

G. Montavon
Institute of Computer Science
Department of Mathematics and Computer Science
Freie Universität Berlin

Exercise Sheet 8

In this exercise, we would like to measure the correlation between two data modalities in a way that is *reproducible*. Each modality may be multi-dimensional, therefore, we consider Canonical Correlation Analysis (CCA), which is a generalization of Pearson's correlation. The solution of the CCA problem is the leading eigenvector of the generalized eigenvalue problem

$$\begin{bmatrix} 0 & C_{xy} \\ C_{yx} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_x \\ \boldsymbol{w}_y \end{bmatrix} = \lambda \begin{bmatrix} C_{xx} + \sigma^2 I & 0 \\ 0 & C_{yy} \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_x \\ \boldsymbol{w}_y \end{bmatrix}$$

The variable σ^2 is some positive hyperparameter that can be used to lower the risk of overfitting.

The following cell loads a modified version of the sklearn digits dataset consisting of 150 randomly selected instances that have been corrupted with noise. The label information is concatenated to the data in form of one-hot encodings. Pixel-values form the first modality, and label information form the second modality. The indices of the dimensions corresponding to the two modalities are given in `inds1` and `inds2` respectively.

In [2]:

```
import numpy
import sklearn
import sklearn.datasets
import utils

dataset = sklearn.datasets.load_digits()

# Concatenate the two data modalities
D = numpy.concatenate([
    dataset['data'],
    numpy.eye(10)[dataset['target']]
],axis=1)
inds1 = numpy.arange(0,64)
inds2 = numpy.arange(64,74)

# Corrupt the first modality with noise and clip/rescale pixel values
D[:,inds1] = D[:,inds1] - 8 + numpy.random.mtrand.RandomState(0).uniform(-12,12,D[:,inds1].
D[:,inds1] = numpy.clip(D[:,inds1],-8,8)/8.0

# Pick randomly 150 instances, and organize them in 5 subsets
P = numpy.random.mtrand.RandomState(1).permutation(len(D))[:150]
D = D[P].T.reshape(74,5,30)
```

Exercise 1: CCA on Holdout Data (25 P)

The CCA class and its training procedure are available in `utils.CCA`. The following class extends `utils.CCA` with a function `predict` that measures the correlation of the projected two data modalities for some test data. In other words, it compute

$$\rho = \mathrm{Corr}(\boldsymbol{w}_x^\top \boldsymbol{x}, \boldsymbol{w}_y^\top \boldsymbol{y})$$

where the correlation is measured over some test data given as input. *Implement* the `predict` function.

In [18]:

```
class CCA(utils.CCA):

    def predict(self,D):

        corr = self.wx.size
        x_proj = self.wx.reshape((1,-1)) @ D[self.inds1].reshape((self.wx.size,-1))
        y_proj = self.wy.reshape((1,-1)) @ D[self.inds2].reshape((self.wy.size,-1))
        cov = numpy.multiply(x_proj - x_proj.mean(), y_proj - y_proj.mean()).mean()
        corr = cov / (x_proj.std() * y_proj.std())

        return corr
```

The following code tests the function above by fitting a CCA model on the data subsets 1-4, and measuring the correlation obtained with the learned projections on the subset 5.

In [19]:

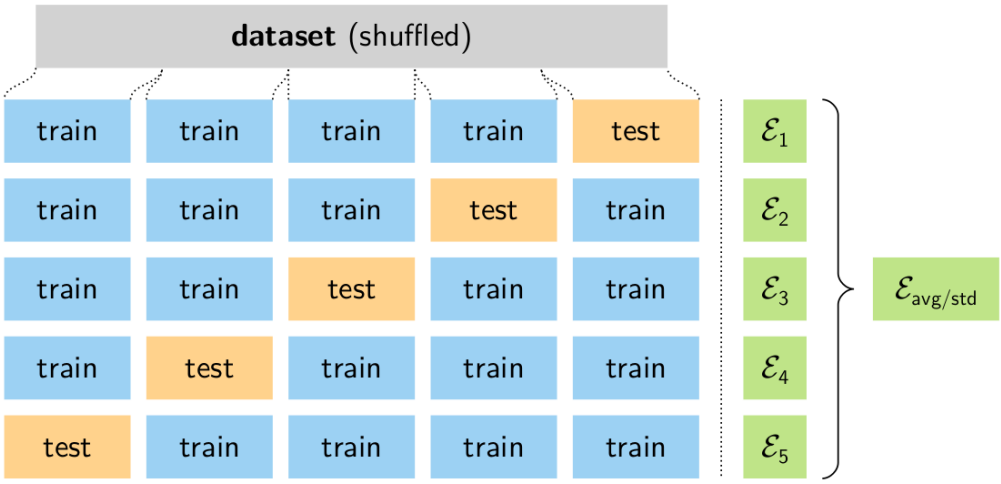
```
cca = CCA(1e-12,inds1,inds2)
cca.fit(D[:,:4])

corr = cca.predict(D[:,:4])
print(f"corr train = {corr:.3f}")

corr = cca.predict(D[:,4:])
print(f"corr test  = {corr:.3f}")
```

```
corr train = 0.947
corr test  = 0.662
```

We observe positive correlation scores, which suggest that there is a connection between pixel values and class labels despite the noise. However, there is a large gap between correlations scores on the training and test sets, which is a sign of overfitting.

Exercise 2: Implementing Cross-Validation (25 P)

First, to verify that the results above are consistent over different training/test splits, we would like to perform 5-fold cross-validation. The procedure is illustrated in the figure below.



(Note that unlike what is seen in this figure, we want to measure correlation rather than prediction error.) *Implement* a function that performs such cross-validation. The function receives the dataset and a CCA object as input and should return the average and standard deviation of the error over the multiple splits. Your implementation should call the CCA `fit` function, as well as the `predict` function you've just implemented.

In [5]:

```
def CV(D,model):

    corr_list = []
    for test_fold in range(D.shape[1]):
        index = [x for x in range(D.shape[1]) if x != test_fold]
        model.fit(D[:,index])
        corr_list.append(model.predict(D[:,test_fold]))
    mean, std = numpy.asarray(corr_list).mean(), numpy.asarray(corr_list).std()

    return mean,std
```

The following code tests the cross-validation procedure you have implemented.

In [6]:

```
cca = CCA(1e-12,inds1,inds2)

mean,std = CV(D,cca)

print(f"corr = {mean:.3f} +/- {std:.3f}")
```

```
corr = 0.580 +/- 0.081
```

We observe an average correlation score similar to the one obtained for the fixed split used in Exercise 1. We further observe that there is significant variance over the different splits.

Exercise 3: Hyperparameter Selection (25 P)

In practice, it is important to select the CCA regularization hyperparameter σ^2 in a way that the correlation on new data is maximized. Write a function that takes a list of candidate hyperparameters `sqsigmas`, and returns the hyperparameter that maximizes the average correlation as measured with cross-validation. Your function should make use of (i.e. call multiple times) the function `CV` you have implemented just above.

In [9]:

```
def selection(D,inds1,inds2,sqsigmas):
    max_corr = -10
    selected_sqsigma = None
    for sqsigma in sqsigmas:
        cca = CCA(sqsigma,inds1,inds2)
        cv_corr = CV(D, cca)[0]
        if cv_corr > max_corr:
            selected_sqsigma = sqsigma
            max_corr = cv_corr
    return selected_sqsigma
```

The following code tests the hyperparameter selection procedure.

In [10]:

```
sqsigmas = numpy.logspace(-10,10,21)
print('sqsigma = %f'%selection(D,inds1,inds2,sqsigmas))
```

```
sqsigma = 10.000000
```

We obtain a fairly large value for the optimal hyperparameter σ^2, thereby suggesting that the corresponding term in the objective is useful for reducing overfitting.

Exercise 4: Nested Cross-Validation (25 P)

Note that the parameter selection above does not provide an estimate of the correlation obtained with that optimal hyperparameter. To get one such estimate, we need to reserve one subset of the data for testing and perform hyperparameter selection on the remaining 4 subsets. An efficient procedure that combines hyperparameter selection and evaluation is called nested cross-validation and is shown in the slides. Implement this procedure. You can call the function `selection` implemented just above from your method. Note that when a parameter has been selected, you can generate your final model associated to this parameter by retraining on both subsets used for training and validation (as described in the slides).

In [20]:

```
def NestedCV(D,inds1,inds2,sqsigmas):

    corr_list = []
    for test_fold in range(D.shape[1]):
        index = [x for x in range(D.shape[1]) if x != test_fold]
        best_sigma = selection(D[:,index],inds1,inds2,sqsigmas)
        cca = CCA(best_sigma,inds1,inds2)
        cca.fit(D[:,index])
        corr_list.append(cca.predict(D[:, test_fold]))
    mean, std = numpy.asarray(corr_list).mean(), numpy.asarray(corr_list).std()
    return mean,std
```

In [21]:

```
print('corr = %.3f +/- %.3f'%NestedCV(D,inds1,inds2,sqsigmas))
```

```
corr = 0.763 +/- 0.038
```

With the hyperparameter selection procedure, the correlation coefficient on new data improves on average compared to what we have obtained in Exercise 2 with the default parameter $\sigma^2 = 10^{-12}$ (an absence of regularization). Overall, the stronger correlation obtained with this hyperparameter demonstrates the presence of a correlation between pixels and class labels in a way that is more easily reproducible.