Problem Set 06: Cross Validation and Kernels 2

Johannes C. B. Dietschreit, Sascha Mausenberger

Due: 18.11.2024

The problems below are meant to be solved using Python functions and libraries. You can do so by either writing short scripts or using Jupyter notebooks. **Report your results in the Quiz on Moodle** to obtain a grade.

The Problems

1 Cross-Validation

In this exercise we will attempt ourselves at recognizing hand-written numbers. One of the best-known benchmarks in machine learning. This time we will simply use a data set that is part of scikit-learn itself. The code below will load the data set and draw the first 4 numbers together with the labels (the number they are supposed to represent).

```
from sklearn import datasets
import matplotlib.pyplot as plt
digits = datasets.load_digits()

_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image, label in zip(axes, digits.images, digits.target):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title("Training: %i" % label)
```

1.1 Split the dataset `digits.data` into 5 equally large folds (take only the first 1795 data points), do the same for the labels `digits.target`. Keep them in the provided order, do not shuffle the samples prior to splitting. Train five models `SVC(kernel='rbf', gamma=0.01)`, always on 4 out of the 5 folds. So that each folds has been used 4 times across the five models. For each model compute the accuracy (1 - missclassificaiton error) on the fold that was not used in the models training. Compute the mean of those 5 values and round it to 3 digits after the decimal point.

(1.5 P)

1.2 Now, what we have done before is not proper, as we used the entire data set, but we also have not optimized any hyper parameter. With the rbf kernel we have one parameter that needs adjusting, `gamma`. For this reason we will first split the data in train and test sets using the `train_test_split(test_size=0.2, random_state=42)`. This way we prevent data leakage. Then we split the training set (X and y) further into 5 folds. Use `KFold` from `sklearn.model_selection`. Using these splits you will train always 5 models as before, additionally we will scan different gamma values `gammas=np.logspace(-1, -5, 9)`. For each set

of 5 models, compute the average accuracy. Which gamma has the highest average accuracy? (1.0 P)

1.3 For the optimal gamma, train a model on the entire training set and compute the accuracy on the test set. Is it similar to the average from the folds computed before?

(0.5 P)

1.4 Now we will try to zoom in a bit more on the best gamma. We will also test a different kind of splitting, `LeaveOneOut`. This kind of split is much more costly as it creates as many splits as there are elements in the training set (hence each iteration will take about a minute, as we will be training over a thousand models instead of 5). We will scan the gammas `np.linspace(5.0e-4, 1.0e-3, 6)`, each iteration will take about a minute.

What is the optimal gamma (as predicted by the LeaveOneOut CV scan)? Report the accuracy for that gamma on the test set rounded to three digits after the decimal point.

(1.0 P)

1.5 Use `GridSearchCV`, the parameters `parameters = {'gamma':[1.e-3, 7.5e-4, 5.e-4, 2.5e-4], 'alpha':[1.e-4, 1.e-3, 1.e-2, 1.e-1]}` and kernel ridge regression with the 'rbf' kernel. Use `GridSearchCV(krr, parameters, scoring='neg_mean_squared_error')` for the grid search. Report the R2 and MAE on the test set with the optimal model. Round the result to 3 digits after the decimal point.

(1.0 P)

2 Combining Kernels

We will now deal with a more complex kernel problem, where we are going to try to combine different kernels to predict a periodic time series. We will use a data set from the Mauna Loa Observatory that collected air samples. We are interested in estimating the concentration of CO_2 and extrapolate it for future years. First, load the original dataset available in OpenML as a pandas dataframe. Use:

```
from sklearn.datasets import fetch_openml

co2 = fetch_openml(data_id=41187, as_frame=True)
df = co2.frame
```

2.1 How many years of measurements are included included in this data set?

(0.5 P)

2.2 Unfortunately the date of the measurements is encoded in three different columns `year`, `month` and `day`. We need a continuous axis to process the data.

Combine those three timestamps using the code below. The data set contains several measurements per month, which are a bit noisy. To smoothen the data, we will compute averages over the same month. To do that, create a new 'Series' using the following code

```
df['date'] = pd.to_datetime(df[['year', 'month', 'day']])
df['year_month'] = df['date'].dt.to_period('M')
df['year_month_float'] = df['year_month'].dt.year +  df['year_month'].dt
    .month / 12
monthly_avg = df.groupby('year_month')[['co2', 'year_month_float']].mean
    ().reset_index()
```

Plot the smoothed and the original data against their time stamps (do not use the float version). FYI, the CO_2 concentrations are in ppm.

(0.5 P)

2.3 Convert the `"year_month_float"` column into the feature vector and the monthly average CO_2 value into the label. To make the fitting easier, subtract the minimum of the CO_2 concentration, such that the minimum is 0 and not above 310. Do the same for the feature vector.

Then split the data into train and test set, where train makes up the first 75% of the data and test the last 25% (NOT shuffled!). Plot the label (rescaled CO_2 conc.) against the feature vector (scaled time index). Color the plot by what data belongs to train and test.

(0.5 P)

2.4 What kind of function best fits the data? Fit the train data with `KernelRidge` with `alpha=10.0`. Try every kernel in `['linear', 'poly', 'rbf', 'sigmoid', 'laplacian']`. Which produces the best forecasting fit?

(1.0 P)

2.5 The problem is that neither of these kernels fits the monthly variation, only the overall trend over the years. There is no kernel that can simultaneously fit both. Therefore, we need to employ a sum of kernels. Recall that a sum of valid kernels is also a valid kernel.

First, use the kernel that seems to be the best at predicting the entire time series and fit a model (use `alpha=1.0`) on the training set as usual. Now do a prediction for the entire time (train and test together) and subtract it from the entirety of the labels. Split these residuals exactly like the labels before, so first 75% are train, the last 25% are test.

Plot the residuals (the 100%) against the float time that starts at 0.

(1.0 P)

2.6 Now we will have to fit the **residuals**. For this we will need our own kernel, as none of the supplied ones is truly periodic. We will use the `"precomputed"` kernel and `alpha=1.0`. This means that you will have to compute the kernel matrix for all samples yourself. Use the code provided below. The function takes two `np.ndarray` as arguments and returns a kernel matrix. Then this kernel matrix is passed along with the labels to the `.fit()` attribute of the instance of `KernelRidge`. Compute the RMSD on the test part of the residual series, round your result to 3 digits after the decimal point.

```
def exp_sine_kernel(X: np.ndarray, Y: np.ndarray):
    n_data1 = X.shape[0]
    n_data2 = Y.shape[0]
    kernel_mat = np.zeros(shape=(n_data1, n_data2))
    for idx1 in range(n_data1):
        for idx2 in range(n_data2):
            d = np.linalg.norm(X[idx1] - Y[idx2])
            value = np.exp(-2.0*np.power(np.sin(np.pi*d), 2))
            kernel_mat[idx1, idx2] = value

    return kernel_mat
```

(0.5 P)

2.7 Now to put everything together. You need to load

`from sklearn.metrics.pairwise import polynomial_kernel` as we will need to use the `"precomputed"` kernel. Compute the kernel matrix (train and test) for this kernel, make sure to use `degree=3`.

You will have to add the kernel matrices for the two different kernels (the polynomial one and the exponential sine) before feeding them to the model. Compute the RMSD of this combined model for the test set (not the residual, this is all done on the full signal!) and plot the difference between the model (predicted for the entirety of X) and the original label.

(1.0 P)