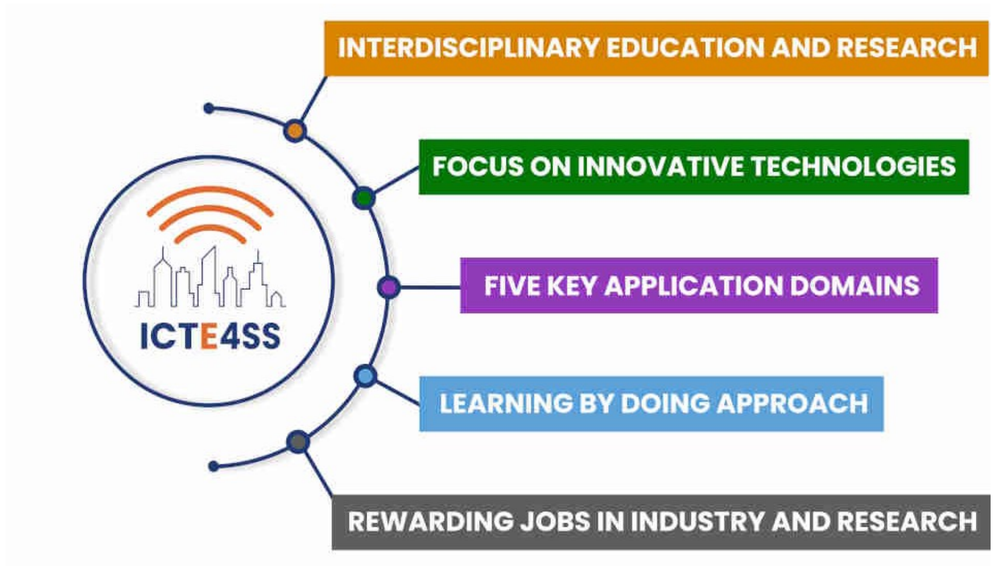
ML for Health
Laboratory # 2: K nearest neighbour-LLS linear regression on Parkinson's dataset

Prof. Monica Visintin - Politecnico di Torino



Description

We want to regress again Total UPDRS as in laboratory # 1, and we will only use LLS method. The difference is that we want to use only K nearest neighbours to train the model. We will use the validation dataset to find the "optimum" value of K.

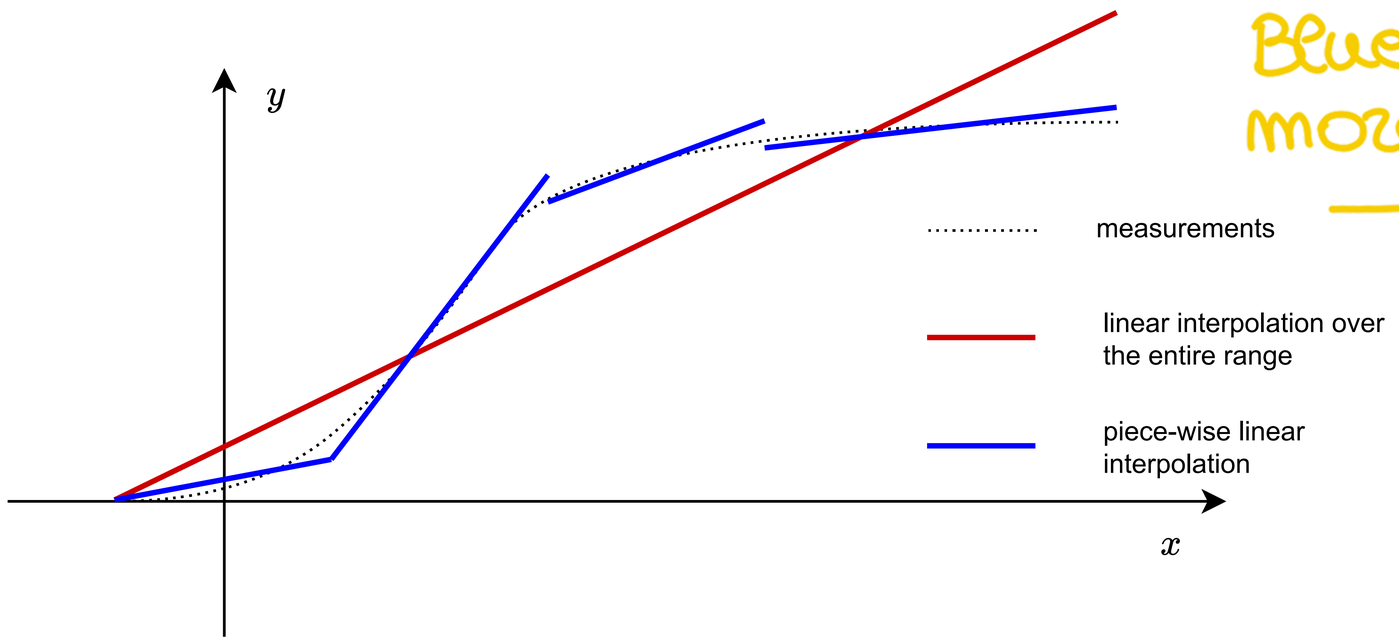
Rationale [1]

"Each function is linear if you zoom enough".
You know that Taylor's series expansion truncated at the first term (the linear part)

$$f(\mathbf{x}) \simeq f(\mathbf{x}_0) + \nabla_f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)$$

if $x - x_0$ is not ∞ large → correct

is valid only for points $\mathbf{x}$ close to $\mathbf{x}_0$. Example in one dimension:



Blue line more precise → use subset

Rationale [2]

- We want to apply the same principle to linear regression, i.e. get different linear models, depending on the position of point $\mathbf{x}_0$ (regressors) for which we want to find $y = f(\mathbf{x}_0)$ (the regressand).
- We simply need to define "closeness" in an easy way. The easiest way is to use only K closest points to $\mathbf{x}_0$ to find the optimum weight vector $\hat{\mathbf{w}}$ so that $f(\mathbf{x}_0) \simeq \mathbf{x}_0^T \hat{\mathbf{w}}$: $\mathbf{x}_0 \in \mathbb{R}^F$ is the test point that stores the regressors and $f(\mathbf{x}_0)$ is the corresponding regressand.
- To get $\hat{\mathbf{w}}$ we use only the K nearest neighbours of $\mathbf{x}_0$ in the training dataset: as a difference with respect to laboratory # 1, matrix $\mathbf{X}_{train}$ has now K rows and F columns instead of having N_{Tr} rows. Using this $\mathbf{X}_{train}$ matrix and the corresponding regressand values $\mathbf{y}_{train}$, $\hat{\mathbf{w}}$ is found applying the LLS method. k out of N_{Tr}

Note that each of the test points will have its own weight vector $\hat{\mathbf{w}}$ (so it would be more correct to call it $\hat{\mathbf{w}}(\mathbf{x}_0)$). each point

We cannot know in advance which value to use for K: we find it using the **validation** dataset.

Note: algorithm KNN (used for regression) can be used to perform a similar task, but KNN outputs the average regressand value of the K neighbours, it does not implement linear regression over the K neighbours.

Data preparation

1. As in Lab #1, load the Parkinson's disease dataset, remove the unwanted features, shuffle the data, get the training, and test datasets. Use the following features as regressors: 'sex', 'age', 'motor_UPDRS', 'Jitter(%)', 'Jitter(Abs)', 'Jitter:RAP', 'Jitter:PPQ5', 'Shimmer', 'Shimmer(dB)', 'Shimmer:APQ3', 'Shimmer:APQ5', 'Shimmer:APQ11', 'NHR', 'HNR', 'RPDE', 'DFA', 'PPE' (they are 17), and use 'total_UPDRS' as regressand.
2. As in Lab #1, set the seed.
3. As a difference with respect to Lab # 1, the data must be divided into training, **validation** and test datasets, according to the percentages: 40%, 20%, 40%. Note that with such a division, you technically have 60% of training data and 40% of test data; the training data are further divided into validation data (33%) and true training data (66%).
4. As in Lab # 1 find mean and standard deviation of each feature in the true training dataset (as if you did not know validation data and test data), and normalize the entire dataset using these means and standard deviations.
5. As in Lab # 1 extract the regressand, i.e. total UPDRS, for the training, validation and test datasets.
6. At the end you have a matrix `X_train_norm`, a vector `y_train_norm`, a matrix `X_test_norm`, a vector `y_test_norm`, a matrix `X_val_norm` a vector `y_val_norm`, all normalized, with obvious meanings. For simplicity, use Numpy arrays, not Pandas dataframes

Fixed K

min K distances

1. Set a suitable value for K (for example $K = 20$). For each $\mathbf{x}$ in the validation dataset:
   - **1.1** Find the distance (or square distance) between $\mathbf{x}$ and each of the N_{tr} points of the training dataset (use normal Numpy methods (`x**2` for the element-wise square of vector`x` and `np.sum`).
   - **1.2** Select the K points of the training dataset with the smallest distance from $\mathbf{x}$ (available methods are `np.sort` or `np.argsort`)
   - **1.3** Generate matrix $\mathbf{A}$ (K rows and F columns); generate column $\mathbf{y}$ (true values of total UPDRS for the K selected training points), generate

$$\hat{\mathbf{w}} = \left(\mathbf{A}^T\mathbf{A} + \epsilon\mathbf{I}\right)^{-1}\mathbf{A}^T\mathbf{y}$$

ε times I

where $\epsilon = 10^{-8}$ and $\mathbf{I}$ is the identity matrix (which shape?). Term $\epsilon\mathbf{I}$ is added to guarantee that it is always possible to invert the matrix (see ridge regression).

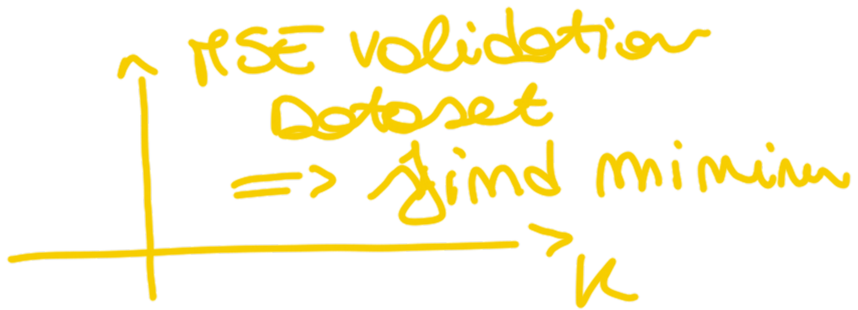
   - **1.4** Evaluate the estimated UPDRS for the validation point $\mathbf{x}$ as

$$\hat{y}(\mathbf{x}) = \mathbf{x}^T\hat{\mathbf{w}}$$

and the estimation error as

$$e(\mathbf{x}) = y(\mathbf{x}) - \hat{y}(\mathbf{x})$$

where $y(\mathbf{x})$ is the true UPDRS value for $\mathbf{x}$. In the above linear algebra equations $\mathbf{x}$ is assumed to be a column vector.



2. Evaluate the mean square error for the validation dataset.

Optimization of K

1. Try and use the script as in the previous slide with some "extreme" values of K and store on a piece of paper the validation mean square error for the selected values of K.
2. Decide possible minimum and maximum values of K: K_{min} and K_{max}, respectively.
3. Generate an external loop letting K run from K_{min} to K_{max} with a chosen appropriate step.
4. For each value of K store the validation mean square error.
5. Plot the the validation mean square error versus K, and find K_{opt} that gives the minimum validation mean square error.

Note: you might use the ball tree algorithm available in Scikit Learn (sklearn.neighbors, BallTree) to find the neighbours, and you might use grid search available in Python (sklearn.model_selection, GridSearchCV) to find the optimum value of K, but I want you to do these operations manually, so that you understand the entire process in details.

Test phase

1. Using $K = K_{opt}$, find $\hat{y}(\mathbf{x})$ for $\mathbf{x}$ in the test dataset.
2. As in Lab # 1, for the test dataset and $K = K_{opt}$ measure the mean value, the standard deviation and the mean square value of the error, the correlation coefficient, the coefficient of determination R^2, the regression line, the estimation error histogram.
3. In the script for Lab # 2, add the standard LLS regression (the same you used in Lab # 1) using the same training dataset made of 60% of the total points, so the test points are exactly the same as in point 1. above. For the test dataset measure the mean value, the standard deviation and the mean square value of the error, the correlation coefficient, the coefficient of determination R^2, the regression line, the estimation error histogram.
4. Compare the results you get with the standard LLS linear regression and the K nearest neighbour-LLS linear regression (note: if the test points are not exactly the same, the results cannot be compared). Do you see different results? which technique is the best? Remember that we do not care of execution time/complexity, we are only concerned about minimization of the estimation error.

compare 40 to 40 w/ LLS 60/40

EXAM 5. Can you measure the performance metrics for the training dataset using KNN-LLS?

↳ error/variance