RWTH Aachen University
Institute for Data Science in Mechanical Engineering

Document Version 1.0.2, released 15/12/2021

Bonus Point Assignment I

1 Introduction

This document gives all necessary information concerning the first bonus point assignment for the class CSME II in the winter semester 21/22. The assignment consists of this document and a Jupyter notebook which you will complete with your solutions. Each task described in this assignment document corresponds to one or more cells in the Jupyter notebook. The latter also contains more details on the precise format in some cases (e.g., you might have to fill in your implementation in a function skeleton or store your solution in a specific variable).

Formalia With this voluntary assignment, you can earn up to 5% of bonus points for the final exam. The bonus points will only be applied when passing the exam. A failing grade cannot be improved with bonus points. This assignment starts on December 1st and you have until December 22nd 23:59:59 CET to complete the tasks. The assignment has to be handed in in groups of 1 to 4 students[1]. All group members have to register in the same group on Moodle under *Groups Assignment 1*.
Please hand in your solution as a single zip file containing the notebook with your solutions. Do not clear the output of your submitted notebooks. Your solution has to be handed in via Moodle before the deadline.

Grading This might include automatic checks of your implementation via unit tests, so ensure that your notebook runs correctly from start to finish. Furthermore, be careful to follow the instructions, e.g., sometimes we ask for a manual implementation (i.e., not relying on readily available functions from machine learning libraries).

Jupyter To run the Jupyter Notebooks for this assignment, you can use the RWTH JupyterHub.
Alternatively, you can set up your environment locally (in this case we recommend using the Anaconda environment). However, please note that due to ressource constraints we can give only limited support in case your environment does not work.

A remark on the nature of the tasks In some of the subtasks we ask you to implement standard methods like data set split, normalization, running linear regression. Of course, in practice you could use a proper library like scikit-learn for such tasks. However, we consider it important that you

[1] If there is a large number of groups of size one, we might have to enforce group sizes of at least 2 or 3 due to limited grading capacity.

implement these basic tasks at least once yourself, both as a check of your understanding as well as to get more familiar with basic concepts.

Source Parts of this assignment are based on Exercise 16.1 in [2].

We appreciate if you could report any errors found in this assignment to the teaching assistants, preferably in the Bonus Point Assignment Forum in Moodle or via email (csme2@dsme.rwth-aachen.de).

2 Part A: Linear Regression for a Robotic Arm

Introduction Here we consider learning the inverse dynamics of a robot arm from data. Inverse dynamics refer to the map from motion coordinates (position /velocity /acceleration) x to the torques y. This is an important concept in robotics and finding such a map can be challenging. Here we use a learning-based approach on a popular data set originally from [1] that was generated from a seven degrees-of-freedom SARCOS anthropomorphic robot arm. The input $x \in \mathbb{R}^{21}$ are the desired position, velocity and acceleration (hence the dimensionality is $3 \times 7 = 21$) and the output $y \in \mathbb{R}^7$ are the corresponding torques. For simplicity, we use only the first output in the following.

Setup

Download both the training and test data set from http://www.gaussianprocess.org/gpml/data/ (The SARCOS data). The data is provided as `.mat` files, the common data storage type of MATLAB.

Task A.1 Load the data from sarcos_inv.mat and sarcos_inv_test.mat in numpy arrays (rows corresponding to measurement points, columns corresponding to the different variables). Only load the first torque variable for the output data. Furthermore, randomly split the data from sarcos_inv.mat into training data (80%) and validation data (20%) and use the samples from sarcos_inv_test.mat as test data. Implement this data set split manually, i.e., do not use e.g. scikit-learn's `train_test_split` method.

Task A.2 Standardize the data such that

1. Training inputs have mean 0
2. Each training input variable has variance 1
3. The training output has mean 0
4. Apply the same transformation to the validation and test data

Implement this manually, i.e., do not use a library scaler like the one provided by scikit-learn.
Standardizing data is a very common task in data science, statistics and machine learning. It is often necessary to avoid numerical problems and ensuring that different quantities involved in the learning problem are comparable.

Task A.3

a) Implement manually a function that estimates the variance of (data contained in) a 1d numpy array.

b) Estimate the variance σ_y^2 of the training output data using your function.

c) Implement a function calculating the Standardized Mean Squared Error of a 1d numpy array of predictions $\hat{z}$ given data (again a 1d numpy array) and variance σ_z^2 according to the formula

$$SMSE = \frac{1}{N \cdot \sigma_z^2} \sum_{n=1}^{N} (\hat{\mathbf{z}}_n - \mathbf{z}_n)^2$$

Simple linear regression

Task A.4 Implement manually (i.e., do not use a library function like scikit-learn `.fit`) simple linear regression (i.e., a function that is linear in the inputs, no nonlinear transformation) without bias term and run it on the training data. Evaluate the resulting SMSE on the validation data.
Hint You should get an SMSE around 0.074.

Linear regression with polynomial features

Task A.5

a) Implement a function manually (i.e., do not use an available feature transformer like the one provided by scikit-learn) that generates polynomial features of multivariate input up to a given degree. Example: For a 2-dimensional input (x_1, x_2) and degree up to 2 the function should return $(1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$. Avoid repeating features (e.g., $x_1x_2 = x_2x_1$ is the same and hence should be returned only once).

b) Use your function to run linear regression with polynomial features up to degree 2 and up to degree 3. Calculate in both cases the SMSE on the validation data.

Hint You should get an SMSE around 0.032 and 0.013.

3 Part B: Clustering

Next, we will use Gaussian basis functions (also called Radial Basis Functions). Recall from Lectures 1 and 6 that for this we need a scale parameter (how peaky the functions are) and a location parameter (where the peak of each of the basis functions is located). Intuitively, the basis functions should be located at "interesting" or "representative" points in the input space. For example, suppose the input points arise from just a few tightly concentrated groups or clusters of points, then it seems very reasonable to place a basis function at each of the cluster centers.
Finding structure in data without having access to labels is one of the goals of unsupervised learning. A classical task there is clustering (also called cluster analysis) which is exactly what we need: Finding "groups" or "clusters" of points that are somewhat similar (in some often only vaguely specified sense) and properties of these clusters (e.g., their centers, size, shape).
This part of the bonus assignment gives a self-contained introduction to one of the simplest and most popular approaches to clustering: K-Means

A brief introduction to K-Means

Consider a data set $\mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbb{R}^D$, for which we would like to find $K \in \mathbb{N}_{>0}$ clusters. K-Means is an algorithm to find reasonable cluster centers $\mu_1, \dots, \mu_K \in \mathbb{R}^D$ and even an assignment of each data point $\mathbf{x}_n$ to "its" cluster. To make things precise, define $r_{nk} = 1$ if data point n is supposed to belong to cluster k and 0 otherwise.
The starting point is to try to minimize

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|\mathbf{x}_n - \mu_k\|^2$$

over μ_k and r_{nk}. K-Means does this using a simple two-step procedure: First, minimize J for fixed μ_k. It is clear that this can be done by setting

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_{\ell=1,\dots,K} \|\mathbf{x}_n - \mu_\ell\|^2 \\ 0 & \text{otherwise} \end{cases}$$

Second, minimize J for fixed assignments over μ_k. A short calculation shows that this can be done by setting each μ_k to the center (i.e., the center of mass or the average) of all the input points belonging to it according to the current assignment.
The K-Means algorithm starts with given initial centers and repeats these two steps until a stopping criterion is fulfilled. For a more detailed description, see e.g. [3, Section 9.1].

Implementing K-Means

Task B.1 Implement the basic K-Means algorithm. Your function should take the data set (a 2d numpy array of the form $N \times D$, with N number of samples and D the dimension of each data point), initial cluster centers (a 2d numpy array of shape $K \times D$, where K is the number of clusters) and the number of iterations.

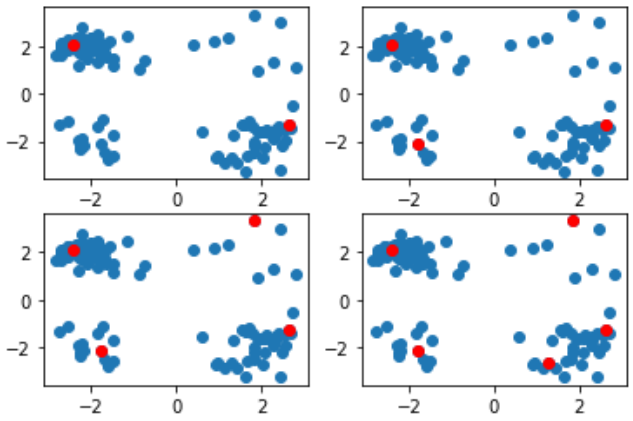

Figure 1: Test data for clustering.

Task B.2 Generate a test data set of 100 points: For each point, independently choose from four 2d normal distributions having means $(-2, 2), (-2, -2), (2, -2), (2, 2)$ and covariance matrices $0.2I, 0.2I, 0.5I, 0.5I$, respectively, with probabilities $0.3, 0.2, 0.4, 0.1$. Plot the data set as a scatter plot.

Task B.3 Run your K-Means implementation on the test data set for $K = 2, 3, 4, 5$ clusters using 5 iterations. For each value of K, plot the final cluster centers together with the data set in a scatter plot.
Since K-Means can be very sensitive to the initial cluster centers, use the kmeans++ algorithm [4] from scikit-learn (use `random_state=0`).
Hint Your plots should look somewhat like Figure 1.

4 Part C: Radial Basis Function network

We are now ready to use the Gaussian basis functions together with linear regression. Note the resulting model is known as Radial Basis Function network (RBF network).

Task C.1 Use K-Means (with kmeans++ for initialization) to find $K = 100$ cluster centers used to place the basis functions. Since the basic K-Means algorithm you implemented in Part B tends to be slow, you may want to use the faster version of K-Means provided by scikit-learn.

Task C.2 Implement the Gaussian basis functions ϕ with center μ and scale σ^2 according to $\phi(x) = \psi(\|x - \mu\|)$ where

$$\psi(r) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{r^2}{2\sigma^2}\right).$$

Transform the training input correspondingly by using the $K = 100$ cluster centers from Task C.1 and choosing $\sigma = 25$.

Task C.3 Run linear regression on the transformed data (include a bias term) and evaluate the SMSE on the validation data.
Hint You should get an SMSE around 0.043.

Open Task Using any techniques you have learned in CSME2 so far, how low can you get the SMSE (of course, avoiding overfitting on the test data)? For this open task you can use now state-of-the-art libraries like scikit-learn.
Some suggestions

- The number of Gaussian Basis functions could be changed
- Maybe a different scale parameter for Gaussian basis functions works better
- What about regularized linear regression?

Providing a solution this Open Task is not necessary to receive full marks on this Bonus Point Assignment. However, we strongly encourage you to play around with the methods you learned in the course so far and try to improve the model.

Task C.4 Evaluate your final model (either the one from Task C.3 or your improved model from the Open Task) on the test data.

References

[1] S. Vijayakumar and S. Schaal, "Locally weighted projection regression: An o (n) algorithm for incremental real time learning in high dimensional space," in Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), vol. 1. Morgan Kaufmann, 2000, pp. 288–293.

[2] K. P. Murphy, Machine learning: a probabilistic perspective. MIT press, 2012.

[3] C. M. Bishop, Pattern Recognition and Machine Learning. Springer, 2006.

[4] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," Stanford, Tech. Rep., 2006.