Last updated: November 24, 2022

Bonus Points Assignment I

Fundamentals of Machine Learning

Prof. Dr. Sebastian Trimpe

Introduction

In this bonus points assignment, you will apply methods from the lecture to a real data set obtained from the SARCOS seven degree-of-freedom robotic arm (Figure 1) [1]. More precisely, your goal is to learn the *inverse dynamics* of the robot, that is, a map from input motion coordinates (positions, velocities) and desired behavior (accelerations) to the necessary control input (forces and torques). Inverse dynamics are an important concept in robotics, and finding them is a notoriously difficult problem.

We propose to learn inverse dynamics via linear regression. We will begin with simple polynomial regression, and work our way to more advanced basis functions called *radial basis functions.* These basis functions have hyperparameters that are non trivial to set; we propose a method involving *clustering* to set them.

The rest of this assignment is divided as follows. We begin by implementing linear regression with basis functions by hand. Then, we introduce and implement the K-means clustering algorithm. Finally, we use K-means to set the hyperparameters of radial basis functions used for linear regression.

Figure 1: SARCOS anthropomorphic robot arm [2].

Logistics

Formalia This voluntary assignment can earn you up to 5% of bonus points for the final exam. The bonus points will only be applied if you pass the exam without them; a failing grade cannot be improved with bonus points. This assignment starts on *November 28th* and is due on *December 19th 23:59:59 CET.*

It has to be submitted in groups of 1 to 4 students. All group members have to register in the same group on Moodle under *Groups Assignment I.* Hand in your work as a single zip archive named `submission_bpa1_[GROUP NUMBER].zip` that you upload on Moodle under the appropriate assignment. Do not clear the output of your notebook(s).

Grading Grading will involve automated tests, so **do not change predefined functions or variable names** as some of the tests depend on them. Please make sure your solutions run correctly without raising exceptions. The lines with `assert` statements are tests for you to check the outcome of your code. Questions marked as *(Optional)* are not graded and can be skipped without consequences on your grade.

Allowed Libraries The libraries you are allowed to use change as you progess through the assignment. We always indicate clearly what these allowed libraries are, and you can always use built-in Python functionalities. The purpose is to have you implement most algorithms and functions by hand once so you have a clear idea of what is hiding behind them. Then, you use higher-performance implementations for applications. You are not allowed to use any library that is not already installed in the virtual environment we provide.

Jupyter You may use the RWTH JupyterHub to run and edit Jupyter Notebooks(profile "[FML] Fundamentals of Machine Learning"). Alternatively, you can set up your environment locally, as explained on Moodle, or use Google Colab for collaborative work. Due to resource constraints, we will not provide support for alternatives to the JupyterHub.

Please report any mistakes found in this assignment to the teaching assistants, on the appropriate Moodle forums or via email (fml@dsme.rwth-aachen.de).

1 Linear Regression for a Robotic Arm

In this section, we implement standard linear regression to learn inverse dynamics on the SARCOS data set. For simplicity, we restrict ourselves to learning only one torque, while the original data set has seven such actuations. The following problems are partially based on [3].

Allowed Libraries In this section, you may only use `numpy` and `scipy.io` for loading data.

Setup

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

Save the two files in the `data` folder of the assignment.

1.1. (a) Load the data from `sarcos_inv.mat` and `sarcos_inv_test.mat` in two numpy arrays.

Each row corresponds to a measurement time, and each column corresponds to a variable. The first 21 columns constitute the state, and the remaining 7 columns are the outputs.

(b) Split the loaded data into six numpy arrays, `xs_train`, `ys_train`, `xs_valid`, `ys_valid`, `xs_test`, and `ys_test` as follows:

- The training and validation data come from `sarcos_inv.mat`, and the test data comes from `sarcos_inv_test.mat`;
- The data from `sarcos_inv.mat` is randomly split into training data (80%) and validation data (20%);
- The arrays `xs_...` contain the state of the robot (21 columns);
- The arrays `ys_...` contain the first output torque (1 column).

We now want to implement helper functions to normalize the SARCOS data and evaluate the models that we will build later.

1.2. Implement the following functions.

(a) Implement the function `my_variance(xs)` that takes in a 1-dimensional numpy array `xs` and computes the sample variance of `xs` as a `float` value.

(b) Implement the function `my_mse(z1, z2)` that takes in two 1-dimensional numpy of the same shape and computes their *Mean Squared Error* according to this formula:

$$\mathrm{MSE} = \frac{1}{N}\sum_{n=1}^{N}(\hat{\mathbf{z}}_n - \mathbf{z}_n)^2 \tag{1}$$

Physical measurements often come in different units, and the numerical values are therefore not directly comparable. Additionally, different measurements may have different orders of magnitude, which can result in numerical problems. A good practice to avoid such issues in data science, statistics, and machine learning is to standardize data.

1.3. Standardize the data such that

- Each training input variable has mean 0;
- Each training input variable has variance 1;
- The training output has mean 0;

- The same transformation (i.e., computed from the training data) is applied to the validation and test data.

Simple Linear Regression

1.4. Implement the function `my_linear_regression(phi, ys)` that takes in a matrix **phi** $\in \mathbb{R}^{N \times D}$ of N D-dimensional features and a vector of N targets, **ys** $\in \mathbb{R}^N$, as numpy arrays and computes the numpy array of the optimal weights $w \in \mathbb{R}^D$ according to the least-squares criterion.
Train a linear regression model with linear basis functions, and evaluate the resulting MSE on the validation data.

Hint. *You should get an MSE of about* 31.

Linear Regression with Polynomial Features

We now want to use polynomial features to improve our results. We start by implementing a polynomial feature transformation manually for the simple case of degree 2 polynomials.

1.5. Implement the function `my_quadratic_features(xs)` that takes in a matrix **xs** $\in \mathbb{R}^{N \times D}$ of N D-dimensional data points as a numpy array and computes N polynomial features up to degree 2 including a bias term (i.e., x^0) as a numpy array. Avoid repeating features (e.g., $x_1 \cdot x_2 = x_2 \cdot x_1$ is the same and hence should be returned only once).

Example. *For a 2-dimensional input* (x_1, x_2) *the function should return* $(1, x_1, x_2, x_1 \cdot x_2, x_1^2, x_2^2)$.

A natural next step is to implement a function to obtain unique polynomial features of arbitrary degree. Unfortunately, implementing such a function efficiently is challenging. Indeed, there are $\sum_{k=1}^{M} \binom{D+k-1}{k-1}$ polynomial features of maximum degree M for a D-dimensional input. You can have a look at [4, Section 2.2] to get an idea as to why.

Most libraries, such as scikit-learn, have it already implemented.

1.6. *(Optional)* Can you come up with a function that computes polynomial features of a multivariate input up to an arbitrary given degree without repetition?

Allowed Libraries From now on, you may use scikit-learn's `PolynomialFeatures` preprocessor. Refer to the sklearn documentation on how to use `PolynomialFeatures`.

1.7. Generate polynomial features of up to degree 3 and including a bias term from the standardized training and validation data. Run linear regression using your polynomial features and evaluate the MSE on the validation data with polynomial features.

Hint. *You should get an MSE of about* 6.8.

2 Clustering

A property of polynomial basis functions is their global effect on the prediction: every training example has a noticeable influence on the prediction, even if the query point is far from the sample. In fact, polynomial regression often predicts unbounded functions as the norm of the input increases.

Radial Basis Functions This property might be unwarranted in some cases, e.g., if prior knowledge indicates that training points far from the query point are uninformative. Then, one might prefer *local* features, such as features that only vary when the query point is close to informative training points. This is especially relevant when the training data comes in tightly concentrated *clusters* of points; a relevant feature is then the distance to each of these clusters. More generally, one can define basis functions that depend on the distance to such clusters:

$$\phi(x) = \psi(\|x - \mu\|),$$

where μ is the position of the cluster and for some function ψ. Such a ϕ is called a *radial basis function*. An example seen in the lecture is that of Gaussian radial basis functions, where ψ is the normalized Gaussian function.

Finding Cluster Centers A challenge posed by the above procedure is finding suitable cluster centers μ. Finding such structure in data, without access to labels, is one of the goals of *unsupervised learning*. In particular, *clustering* is a branch of unsupervised learning that achieves exactly what we need: finding groups of similar points (called "clusters") and properties of these clusters (e.g., their centers, sizes, shapes). This part of the bonus assignment gives a self-contained introduction to one of the simplest and most popular clustering algorithms: *K-Means*.

A Brief Introduction to K-Means

Consider a data set $\mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbb{R}^D$, which we want to partition in $K \in \mathbb{N}_{>0}$ clusters such that the total distance between the points in the same clusters is minimal. More formally, we want to solve the following optimization problem:

$$\underset{(r_{nk})_{n\in\{1,\dots,N\},k\in\{1,\dots,K\}}}{\operatorname{argmin}} \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk} \cdot \|\mathbf{x}_n - \mu_k\|^2 =: \underset{(r_{nk})}{\operatorname{argmin}}\, J(r, \mu), \tag{2}$$

where the variable $r_{nk} \in \{0, 1\}$ is interpreted as assigning the input $\mathbf{x}_n$ to cluster k, and where μ_k is the average of the inputs assigned to cluster k. Importantly, the cluster centers (μ_k)'s vary with the choice of (r_{nk})'s.

Solving this optimization problem is hard[1]. The K-means algorithm solves it approximately by repeating a 2-step procedure:

1. Minimize J over the (r_{nk})'s for fixed cluster centers (μ_k), that is, assign each point to the closest cluster center:

$$r_{nk} = \begin{cases} 1 & \text{if } k = \operatorname{argmin}_{\ell=1,\dots,K} \|\mathbf{x}_n - \mu_\ell\|^2, \\ 0 & \text{otherwise;} \end{cases} \tag{3}$$

[1] It is proven to be NP-hard.

2. Minimize J over the cluster centers (μ_k)'s for fixed assignments (r_{nk})'s. Simple computations show that this corresponds to setting μ_k to the average of the points of the cluster:

$$\mu_k = \frac{\sum\limits_{n=1}^{N} r_{nk}\mathbf{x}_n}{\sum\limits_{n=1}^{N} r_{nk}}. \tag{4}$$

The algorithm starts with given initial centers, and repeats the above procedure until a stopping criterion is fulfilled. For a more detailed description, see e.g. [5, Section 9.1].

Implementing K-Means

Allowed Libraries In the following questions, you may only use `numpy` and `matplotlib`.

2.1. Implement the basic K-Means algorithm. Your function `my_kmeans(xs, init_centers, n_iter)` takes in a matrix $\mathbf{xs} \in \mathbb{R}^{N\times D}$ of N D-dimensional data points as a numpy array, initial centers $\mathbf{c}_{\text{init}} \in \mathbb{R}^{K\times D}$ as a numpy array and the number of iterations `n_iter` as an integer. Your function should compute the final centers $\mathbf{c}_{\text{final}} \in \mathbb{R}^{K\times D}$ as a numpy array after performing `n_iter` iterations of the K-Means algorithm.

We now want to test our implementation of K-Means on a simple, visual example. For this, we start by generating a toy dataset.

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

We then need initial cluster centers. The K-means algorithm is very sentitive to choice of these initial cluster centers. We propose to use an established heuristic available in scikit-learn for this.

Allowed Libraries For the next question, you should use the `kmeans++` algorithm [6] from `scikit-learn` to initialize cluster centers. Use `random_state=0`.

2.3. Run your K-Means implementation on the toy 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. You should get a result similar to Figure 2.

3 Radial Basis Functions Network

Let us return to the problem of finding inverse dynamics for the SARCOS robot arm. We now have a principled way of setting the centers of radial basis functions for linear regression. In what follows, we propose to use Gausian radial basis functions. This combination of radial basis functions and of linear regression is an example of a *radial basis functions network* (RBF network), a class of neural networks.

The version of K-means that you implemented in the previous section is quite slow, so we propose to use scikit-learn's implementation instead.

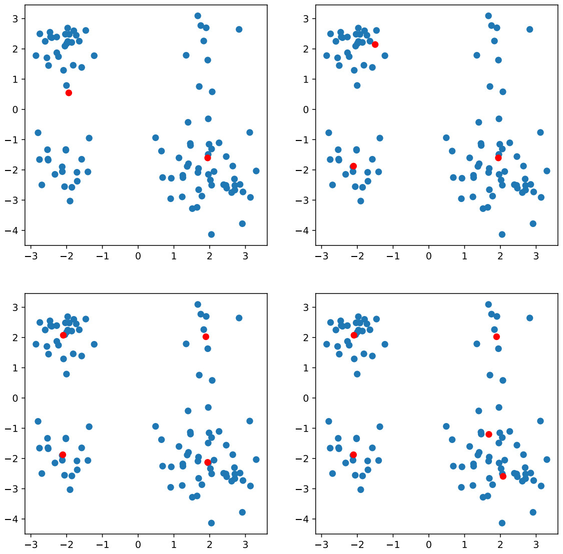

Figure 2: Result of 5 iterations of K-means for $K = 2, 3, 4, 5$ clusters. Data points are in blue, found cluster centers in red.

Allowed Libraries In the following, you may only use `numpy`, `matplotlib`, the `KMeans` implementation provided by `sklearn.cluster`, and `kmeans++` for cluster initialization.

3.1. Fill the function `find_centers`, which uses K-Means to find $K = 100$ cluster centers used to place the basis functions.

3.2. Fill the function `my_gaussian`, which takes as input an array of distances r and computes the element-wise Gaussian $\psi(r)$, defined as

$$\psi(r) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{r^2}{2\sigma^2}\right). \tag{5}$$

3.3. Fill the function `compute_rbf_features`, which takes as input the standardized data (cf. Question 1.3), the cluster centers, and the scale σ and outputs the feature matrix. Include a bias term.

3.4. Run linear regression on the transformed data with $K = 100$ and $\sigma = 25$. Evaluate the MSE on the validation set.

Hint. *You should get an MSE of around* 18.3.

Allowed Libraries From now on, you may use all of scikit-learn.

3.5. *(Optional)* Using any techniques you have learned in FML so far, modify the above model to decrease the validation MSE as much as possible. Remember that you are not allowed to use the validation data for training.
Some suggestions:

- Change the number of Gaussian basis functions;

- Try different scale parameters;
- Try regularization.

3.6. Evaluate your final model 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)*, Morgan Kaufmann, vol. 1, 2000, pp. 288–293.

[2] S. Vijayakumar, A. D'souza, and S. Schaal, "Incremental online learning in high dimensions," *Neural computation*, vol. 17, no. 12, pp. 2602–2634, 2005.

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

[4] *Multinomial theorem*, https://en.wikipedia.org/wiki/Multinomial_theorem, Last visit: Nov. 23, 2022.

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

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