Statistical Machine Learning: Exercise 3

Regression, Kernel Theory, Gaussian Processes, Latent Representation
Total Possible Points: 82

Prof. Marcus Rohrbach, Prof. Simone Schaub-Meyer

TECHNISCHE UNIVERSITÄT DARMSTADT

Summer Term 2025

Publication Date: June 12th, 21:00 PM
Due Date: July 6th, 23:59 PM

TEMPLATE: https://colab.research.google.com/drive/1QM37OJ0yJhGyHJhlzW8UGxJ266q4foF7?usp=sharing

Task 1: Regression (34 Points)

Programming Task

Complete the corresponding section of the notebook as part of this task.

In this exercise, you will implement various kinds of linear regressors using the data `lin_reg_train.txt` and `lin_reg_test.t`
The files contain noisy observations from an unknown function $f : \mathbb{R} \mapsto \mathbb{R}$. In both files, the first column represents the input and the second column represents the output. You can load the data using `numpy.loadtxt` and built-in functions for computing the mean.

For all subtasks, assume that the data is identically and independently distributed according to

$$y_i = \boldsymbol{\Phi}(\mathbf{x_i})^\top \mathbf{w} + \epsilon_i,$$

where

$$\epsilon_i \sim \mathcal{N}(0, \sigma^2),$$

and $\Phi : \mathbb{R} \to \mathbb{R}^n$ is a feature transformation such that

$$\mathbf{y} \sim \mathcal{N}(\boldsymbol{\Phi}(\mathbf{X})^\top \mathbf{w}, \sigma^2 \mathbf{I}).$$

Additionally, make sure that your implementations support multivariate inputs. The feature transformations are given in each task; if no basis function is stated explicitly, use the data as is, i.e. $\boldsymbol{\Phi}(x) = x$.

1a) Linear Features (8 Points)

Implement linear ridge regression using linear features, i.e. the data itself by filling in the ToDos of the corresponding task in the provided Colab Template. Include an additional input dimension to represent a bias term and use the ridge coefficient $\lambda = 0.01$.

1. Explain: What is the ridge coefficient and why do we use it?
2. Derive the optimal model parameters by minimizing the squared error loss function.
3. Report the root mean squared error of the train and test data under your linear model with linear features.
4. Include the resulting plot and a short description.

Solution:

1. Explain: What is the ridge coefficient and why do we use it?

In the lecture (Slide 42), the regularized least squares problem is introduced as the Maximum A-Posteriori (MAP) estimate in a Bayesian linear regression setting. The coefficient function is:

$$w = \arg\min_{w} \frac{1}{2}\|\Phi^\top w - y\|^2 + \frac{\lambda}{2}\|w\|^2$$

The term $\frac{\lambda}{2}\|w\|^2$ penalizes large weights and therefore controls the complexity of the model. A smaller λ allows the model to fit the training data more closely, while a larger λ forces the weights to remain small, reducing overfitting.

This form of regularization is also known as Ridge Regression, as described on Slides 41–42. It improves numerical stability, especially when the feature matrix has high collinearity or low rank.

The objective is to avoid unstable ”valleys”. The ones that appear when using *Least squares*, as seen in the figure 1.

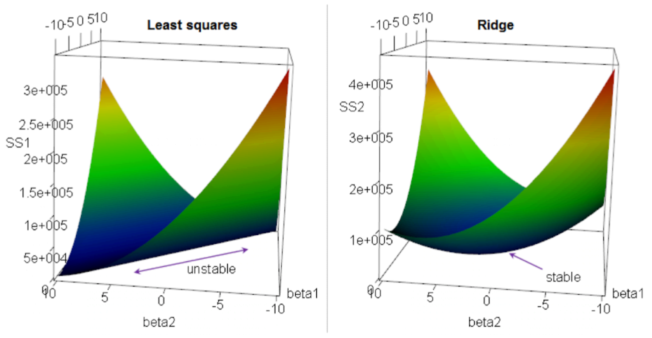


Figure 1: Example. The difference when using Ridge vs. Least squares

2. Derive the optimal model parameters by minimizing the squared error loss function.

As shown in Slide 42 of the lecture, we minimize the regularized least squares loss:

$$L(w) = \frac{1}{2}\|\Phi^\top w - y\|^2 + \frac{\lambda}{2}\|w\|^2$$

Taking the derivative with respect to w and setting it to zero:

$$\frac{\partial L}{\partial w} = \Phi(\Phi^\top w - y) + \lambda w = 0$$

Rearranging:

$$\Phi\Phi^\top w + \lambda w = \Phi y$$

Solving for w:

$$w = (\Phi\Phi^\top + \lambda I)^{-1}\Phi y$$

This is the closed-form solution for regularized linear regression (ridge regression).

3. Report the root mean squared error of the train and test data under your linear model with linear features.

The result of the RMSE of the train adn test data for the linear model and features is the following:

```
Linear Features
  • Train RMSE: 0.4121780156736108
  • Test RMSE: 0.38428816992597875
```

The RMSE values are relatively low and close to each other, indicating that the model generalizes well to unseen data and does not overfit. This is expected for a simple linear model with mild regularization on a smooth dataset.

4. Include the resulting plot and a short description.

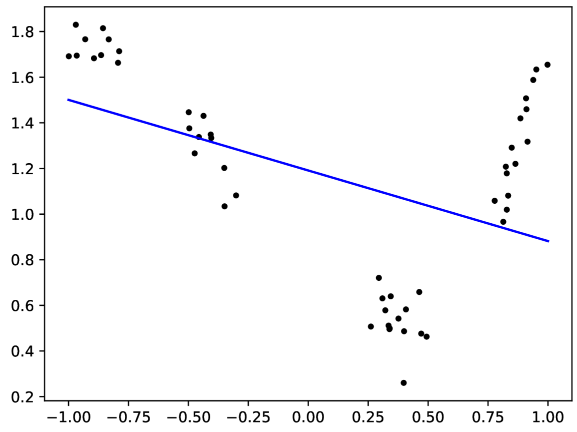


Figure 2: Linear ridge regression fit with $\lambda = 0.01$ using linear features and bias.

The model fits a straight line through the data. Since the data shows a clear nonlinear pattern, the linear model cannot capture it well. Still, the fit shows a general trend and does not overfit the noise, which explains the relatively low RMSE values.

1b) Polynomial Features (8 Points)

Implement linear ridge regression using a polynomial feature projection by filling in the ToDos of the corresponding task in the provided Colab Template. Include an additional input dimension to represent a bias term and use the ridge coefficient $\lambda = 0.01$.

For polynomials of degrees 2, 3 and 4:

1. Report the root mean squared error of the training data and of the testing data under your model with polynomial features.
2. Include the resulting plot and a short description.
3. Why do we call this method *linear* regression despite using polynomials?

Solution:

1. Report the root mean squared error of the training data and of the testing data under your model with polynomial features.

We trained ridge regression models using polynomial features of degrees 2, 3, and 4, with a regularization parameter $\lambda = 0.01$ and a bias term. These are the results:

- **Degree 2:**
  - `Train RMSE: 0.2120`
  - `Test RMSE: 0.2169`
- **Degree 3:**
  - `Train RMSE: 0.0871`
  - `Test RMSE: 0.1084`
- **Degree 4:**
  - `Train RMSE: 0.0870`
  - `Test RMSE: 0.1067`

with a higher degree of the polynomial features, the model's ability to fit the data. This, represented by the lower RMSE values. The model with degree 3 achieves a good balance between training and test error, while the degree 4 model slightly improves the test performance without overfitting.

2. Include the resulting plot and a short description.

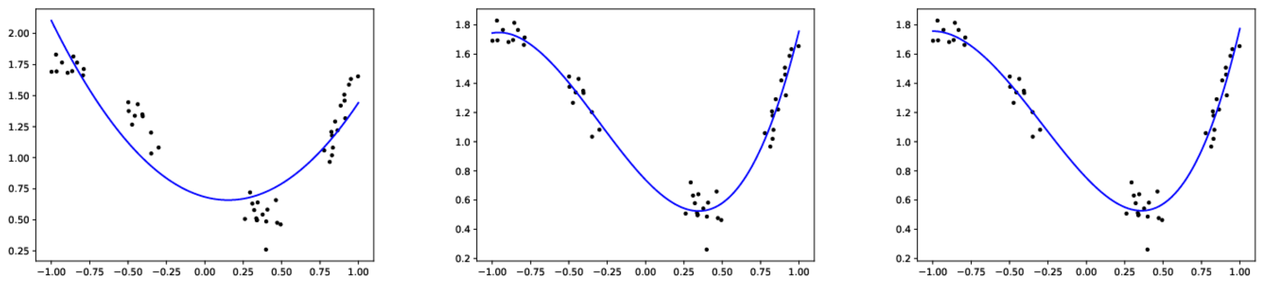


Figure 3: Polynomial ridge regression fits with degrees 2 (left), 3 (center), and 4 (right).

The degree 2 model captures the general U-shape trend but misses the fine details on the ends. The degree 3 model fits the data much more accurately. The degree 4 model performs similarly to degree 3, with no relevant visual differences. The improvement from degree 2 to 3 is significant, while the gain from degree 3 to 4 is minor.

3. Why do we call this method *linear* regression despite using polynomials?

Even though we use polynomial functions of the input, the model is still linear in the parameters w. The prediction is a weighted sum of the features, and we do not multiply or compose the parameters. That's why it's still called linear regression.

$$f(x) = w^\top \phi(x) = w_0 + w_1 x + w_2 x^2 + \cdots + w_d x^d$$

1c) Bayesian Linear Regression (10 Points)

Implement Bayesian linear ridge regression by filling in the ToDos of the corresponding task in the provided Colab Template. Assuming that **w** follows a multivariate Gaussian distribution, such that

$$\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Lambda}_0^{-1}),$$

where ridge regression dictates $\boldsymbol{\mu}_0 = \mathbf{0}$ and $\boldsymbol{\Lambda}_0 = \lambda\mathbf{I}$.

Here, $\boldsymbol{\mu}_0$ is the prior weight mean and $\boldsymbol{\Lambda}_0$ is the prior weight precision matrix, i.e. the inverse of the covariance matrix. The corresponding posterior parameters can be denoted as $\boldsymbol{\mu}_n$ and $\boldsymbol{\Lambda}_n$.

Assume $\sigma = 0.1$, use $\lambda = 0.01$, and include an additional input dimension to represent a bias term. Use all of the provided training data for a single Bayesian update.

1. State the posterior distribution of the model parameters $p(\mathbf{w} \mid \mathbf{X}, \mathbf{y})$ (no derivation required).
2. State the predictive distribution $p(\mathbf{y}_* \mid \mathbf{X}_*, \mathbf{X}, \mathbf{y})$ (no derivation required).
3. Report the RMSE of the train and test data under your Bayesian model (use the predictive mean).
4. Report the average log-likelihood of the train and test data under your Bayesian model.
5. Include the resulting plot and a short description.
6. Explain the differences between linear regression and Bayesian linear regression.

Solution:

1. State the posterior distribution of the model parameters $p(w|X, y)$ (no derivation required).

We assume that the outputs are generated from a linear model with Gaussian noise:

$$y_i = \phi(x_i)^\top w + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, \sigma^2)$$

We place a Gaussian prior on the weight vector:

$$w \sim \mathcal{N}(0, \lambda^{-1} I)$$

Given the likelihood

$$p(y \mid X, w) = \mathcal{N}(y \mid \Phi w, \sigma^2 I)$$

and the prior, we compute the posterior using Bayes' rule:

$$p(w \mid X, y) \propto p(y \mid X, w)\, p(w)$$

Because both the likelihood and prior are Gaussian, the posterior is also Gaussian:

$$p(w \mid X, y) = \mathcal{N}(w \mid \mu, \Lambda^{-1})$$

with:

$$\Lambda = \lambda I + \frac{1}{\sigma^2}\Phi^\top \Phi, \quad \mu = \frac{1}{\sigma^2}\Lambda^{-1}\Phi^\top y$$

This expression balances our prior belief (through λ) and the data evidence (through $\Phi^\top\Phi$), and matches the derivation shown in Lecture 5, Slides 38–42.

2. State the predictive distribution $p(y_*|X_*, X, y)$ (no derivation required).

For a new input x^*, the output y^* is also Gaussian-distributed, even after marginalizing over the uncertainty in w:

$$p(y^* \mid x^*, X, y) = \int p(y^* \mid x^*, w)\, p(w \mid X, y)\, dw$$

Both terms are Gaussian, so the result is Gaussian as well:

$$p(y^* \mid x^*, X, y) = \mathcal{N}(y^* \mid \mu^\top \phi(x^*),\, \sigma^2 + \phi(x^*)^\top \Lambda^{-1} \phi(x^*))$$

The mean of the predictive distribution is:

$$\mathbb{E}[y^*] = \mu^\top \phi(x^*)$$

and the variance includes both the observation noise and the model uncertainty:

$$\mathsf{Var}(y^*) = \sigma^2 + \phi(x^*)^\top \Lambda^{-1} \phi(x^*)$$

This means the model gives not only a prediction but also measure the confidence in that prediction. The further x^* is from the training data, or the less data we have, the greater the uncertainty (variance). This predictive formulation is explained conceptually in Lecture 5, Slides 49–50.

3. Report the RMSE of the train and test data under your Bayesian model (use the predictive mean).

4. Report the average log-likelihood of the train and test data under your Bayesian model.

5. Include the resulting plot and a short description.

6. Explain the differences between linear regression and Bayesian linear regression.

1d) Squared Exponential Features (8 Points)

Implement Bayesian linear ridge regression using squared exponential (SE) features by filling in the ToDos of the corresponding task in the provided Colab Template. In other words, replace your observed data matrix $\mathbf{X} \in \mathbb{R}^{n\times 1}$ by a feature matrix $\mathbf{\Phi} \in \mathbb{R}^{n\times k}$, where

$$\mathbf{\Phi}_{ij} = \exp\left(-\frac{1}{2}\beta(\mathbf{X}_i - \alpha_j)^2\right).$$

Set $k = 20$, $\alpha_j = j * 0.1 - 1$ and $\beta = 10$. Use the ridge coefficient $\lambda = 0.01$ and assume known Gaussian noise with $\sigma = 0.1$. Include an additional input dimension to represent a bias term.

1. Report the RMSE of the train and test data under your Bayesian model with SE features.

2. Report the average log-likelihood of the train and test data under your Bayesian model with SE features.
3. Include the resulting plot and a short description.
4. How can SE features be interpreted from a statisticians point of view? What are α and β in that context?

Solution:

Task 2: Gaussian Processes (12 Points)

Programming Task

Complete the corresponding section of the notebook as part of this task.

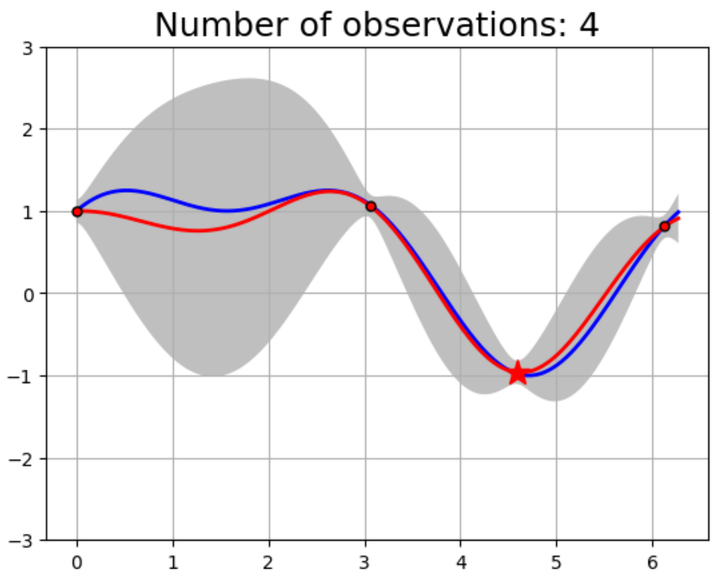
2a) GP Regression (10 Points)

Implement a Gaussian Process to fit the target function $y = \sin(x) + \cos^2(x)$ with $x \in [0, 0.005, 0.01, 0.015, \ldots, 2\pi]$. First, complete the function `rbf_kernel()`.

Use the RBF/Gaussian kernel with a kernel bandwidth of 1, an initial mean of 0 and assume a noise variance of 0.005. Begin with no target data points and, at each iteration, sample a new point from the target function according to the uncertainty of your GP (that is, sample the point where the uncertainty is the highest) and update it.

Plot your GP (mean and two times the standard deviation) after iterations 1, 2, 5, 10 and 15. In each figure, plot also the true function as ground truth and add a new marker for each new sampled point. We have provided an example of the plot at iteration 4.

Include the plots in your PDF submission.



Notes:

- The iteration number is equal to the number of sampled points, e.g. at iteration 5 there are 5 points sampled.
- You do not need to strictly follow the given example plot, e.g., using different marker shapes or colors is okay.
- The pyplot function `fill_between()` might be useful to you.
- For the RBF kernel, we use the definition on Slide 47, Lecture 6, where the vertical amplitude is set to 1.
- If multiple points have the highest uncertainty, select the one with the lowest index in the dataset.

Solution:

2b) Kernel Parameters (2 Points)

Explore various kernel hyperparameters, i.e., `rbf_sigma` in the code, and explain how this parameter affects the prediction results.

Solution:

Task 3: Principal Component Analysis (28 Points)

Programming Task

Complete the corresponding section of the notebook as part of this task.

In this exercise, you will use the dataset `iris.txt`. It contains data from three kind of Iris flowers ('Setosa', 'Versicolour' and 'Virginica') with 4 attributes: sepal length, sepal width, petal length, and petal width. Each row contains a sample while the last attribute is the label (0 means that the sample comes from a 'Setosa' plant, 1 from a 'Versicolour' and 2 from 'Virginica'). (You are allowed to use built-in functions for computing the mean, the covariance, eigenvalues and eigenvectors.)

3a) Data Normalization (3 Points)

Normalizing the data is a common practice in machine learning. Normalize the provided dataset such that it has zero mean and unit variance per dimension. Why is normalizing important? Fill in the ToDos of the corresponding task in the provided Colab Template.

Solution:

3b) Principal Component Analysis (8 Points)

Apply PCA on your normalized dataset and generate a plot showing the proportion (percentage) of the cumulative variance explained (you can use the function `numpy.cumsum`). How many components do you need in order to explain at least 95% of the dataset variance? Fill in the ToDos of the corresponding task in the provided Colab Template.

Solution:

3c) Low Dimensional Space (6 Points)

Using as many components as needed to explain 95% of the dataset variance, generate a scatter plot of the lower-dimensional projection of the data. Use different colors or symbols for data points from different classes. Fill in the ToDos of the corresponding task in the provided Colab Template. What do you observe?

Solution:

3d) Projection to the Original Space (6 Points)

Reconstruct the original dataset by using different number of principal components. Using the normalized root mean square error (NRMSE) as a metric, fill the table below (error per input versus the amount of principal components used). Fill in the ToDos of the corresponding task in the provided Colab Template.

N. of components	x_1	x_2	x_3	x_4
1				
2				
3				
4				

Remember that in the first step you normalized the data.

Solution:

3e) Kernel PCA (5 Points)

Throughout this class we have seen that PCA is an easy and efficient way to reduce the dimensionality of some data. However, it is able to detect only linear dependences among data points. A more sophisticated extension to PCA, *Kernel PCA*, is able to overcome this limitation. This question asks you to deepen this topic by conducting some research by yourself: explain what Kernel PCA is, how it works and what are its main limitations. Be as concise (but clear) as possible.

Solution:

Task 4: Kernel Construction (8 Points)

Let X be a non-empty set. A **kernel** is a symmetric function $k : X \times X \to \mathbb{R}$. The set X can have arbitrary structure (e.g., real vector space, graphs, images, strings, etc.).

A kernel k is called **positive definite** if for all finite subsets $\{x_i\}_{i=1}^n \subseteq X$, the corresponding kernel matrix $K \in \mathbb{R}^{n \times n}$ with entries $K_{ij} = k(x_i, x_j)$ is positive semi-definite (check Lecture 6 slide 9), i.e.,

$$\forall \alpha \in \mathbb{R}^n, \quad \alpha^\top K \alpha \geq 0.$$

Let $k_1, k_2 : X \times X \to \mathbb{R}$ be two positive definite kernels. We are to prove that the following operations yield valid positive definite kernels:

1. $k(x, x') = k_1(x, x') + k_2(x, x')$
2. $k(x, x') = c \cdot k_1(x, x')$, for $c \geq 0$
3. $k(x, x') = k_1(x, x') \cdot k_2(x, x')$
4. $k(x, x') = k_1(f(x), f(x'))$ for some function $f : X \to X$

Solution: