10907 Pattern Recognition

Lecturers
Prof. Dr. Ivan Dokmanić ⟨ivan.dokmanic@unibas.ch⟩

Tutors
Felicitas Haag ⟨felicitas.haag@unibas.ch⟩
Alexandra Spitzer ⟨alexandra.spitzer@unibas.ch⟩
Cheng Shi ⟨cheng.shi@unibas.ch⟩
Vinith Kishore ⟨vinith.kishore@unibas.ch⟩

Problem set 2

Math

Exercise 1 (Matrix calculus basics - $\star$)**.**

Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{x} \in \mathbb{R}^n$. Consider the scalar function

$$f(\boldsymbol{x}) \;=\; \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x}.$$

1. Show that the gradient with respect to x is

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = (\boldsymbol{A}+\boldsymbol{A}^\top)\boldsymbol{x}.$$

   Hint: There are multiple ways to go about this, one is to write $f(\boldsymbol{x}) = \sum_{i,j} \boldsymbol{x}_i \boldsymbol{A}_{ij} \boldsymbol{x}_j$ and differentiate w.r.t. $\boldsymbol{x}_k$; another way is to try to understand $f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} = (\boldsymbol{A}\boldsymbol{x})^\top \boldsymbol{x} = g(\boldsymbol{x})^\top \boldsymbol{x}$ with $g(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}$ and then apply the chain rule.

2. Show that if $\boldsymbol{A}$ is symmetric, then $\nabla_x f(\boldsymbol{x}) = 2\boldsymbol{A}\boldsymbol{x}$.

3. Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{x} \in \mathbb{R}^n$. Show that

$$\|\boldsymbol{A}\boldsymbol{x}\|^2 \;=\; \boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{A}\,\boldsymbol{x}.$$

4. Let $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{R}^{m\times n}$. Show that

$$\nabla_X \tfrac{1}{2}\|\boldsymbol{X}-\boldsymbol{Y}\|^2 = \boldsymbol{X}-\boldsymbol{Y}.$$

Exercise 2 (Multivariate linear regression - $\star\star$)**.**

In multivariate linear regression we define the residual sum of squares (RSS) as

$$\mathrm{RSS}(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y}\|_2^2, \tag{1}$$

where

- $\boldsymbol{X}$ is the $N \times D$ data matrix, containing D-dimensional features for N data points.
- $\boldsymbol{w}$ is the weight vector.
- $\boldsymbol{y}$ is the vector of target values for each training sample.

Solve the following tasks.

1. Partial derivatives: Show that

$$\frac{\partial \mathrm{RSS}(\boldsymbol{w})}{\partial w_k} = \|\boldsymbol{x}_{\cdot,k}\|_2^2 w_k - \boldsymbol{x}_{\cdot,k}^T(\boldsymbol{y} - \boldsymbol{X}_{\cdot,-k}\boldsymbol{w}_{-k})$$

   where

- $\boldsymbol{x}_{\cdot,k}$ is the k-th column of $\boldsymbol{X}$;
- $\boldsymbol{X}_{\cdot,-k}$ is the data matrix $\boldsymbol{X}$ without the k-th column;
- $\boldsymbol{w}_{-k}$ is the weight vector without the k-th component.

Hint: Separate the k-th weight from the rest by writing $\boldsymbol{X}\boldsymbol{w} = \boldsymbol{x}_{\cdot,k}w_k + \boldsymbol{X}_{\cdot,-k}\boldsymbol{w}_{-k}$.

2. Optimal weights: We define the prediction residuals as

$$\boldsymbol{r}_k = \boldsymbol{y} - \boldsymbol{X}_{\cdot,-k}\boldsymbol{w}_{-k}$$

for $k = 1, \dots, D$. Prove that if $\frac{\partial \text{RSS}}{\partial w_k} = 0$, then the optimal weight for the k-th feature (k-th coordinate or dimension or pixel) is given by

$$\widehat{w}_k = \frac{\boldsymbol{x}_{\cdot,k}^T \boldsymbol{r}_k}{\|\boldsymbol{x}_{\cdot,k}\|^2}. \tag{2}$$

Interpretation Given current coefficients $\boldsymbol{w}_{-k}$ the optimal update for coordinate k is the projection of $\boldsymbol{x}_{\cdot,k}$ onto the current residual $\boldsymbol{r}_k$. Iterating these closed-form updates over all coordinates converges to the global minimizer because RSS is a convex quadratic.

Exercise 3 (Classification: optimal decision thresholds under a reject option - ⋆⋆)**.**

In many applications, the classifier is allowed to "reject" a test example rather than classifying it into one of the classes. Consider, for example, a case in which the cost of a misclassification is 10 CHF and the reward (negative cost) of correct prediction is 0.5 CHF. On the other hand, the cost of an additional human evaluation is only 3 CHF. We can summarize this by the following loss matrix:

Decision	true label Y	
$\hat{Y}$	Y=0	Y=1
predict $\hat{Y} = 0 \vert x$	-0.5	10
predict $\hat{Y} = 1 \vert x$	10	-0.5
reject	3	3

1. Let us denote $p_1 = p(Y = 1|x)$ as the posterior probability of class 1 given the observed feature vector x. Show that in general, for this loss matrix and any posterior probability $p_1 \in [0, 1]$, there will be two thresholds θ_0 and θ_1 such that the optimal decision is to predict $\hat{Y} = 0$ if $p_1 < \theta_0$, reject if $\theta_0 < p_1 < \theta_1$ and predict $\hat{Y} = 1$ if $p_1 > \theta_1$.
2. Compute θ_0 and θ_1.
3. Now change the costs so that a correct prediction yields -1 and a wrong prediction costs 20 (the rejection cost remains 3). Recompute the thresholds.

Notation note. We use uppercase letters for random variables (e.g., Y) and lowercase for realized values or given observations (e.g., x). Hats mark decisions, e.g., $\hat{Y}$.

Exercise 4 (Logistic Regression - ⋆⋆)**.**

Consider data in $\mathcal{X} = \mathbb{R}^2$ with binary labels $\mathcal{Y} = \{+1, -1\}$. The joint distribution of data and labels is given as

$$p(\boldsymbol{x}, y) = \frac{1}{4\pi} \exp\left(-\frac{1}{2}\|\boldsymbol{x} - y\mathbf{1}\|_2^2\right) \tag{3}$$

where $\mathbf{1} = [1, 1]^T$ is the all-one vector. Assume $P(y = 1) = P(y = -1) = \frac{1}{2}$.

- Determine $P[y = 1|\boldsymbol{x}]$ and $P[y = -1|\boldsymbol{x}]$.
- Compute the weight vector for logistic regression which satisfies

$$\begin{bmatrix} P[y=-1|\boldsymbol{x}] \\ P[y=1|\boldsymbol{x}] \end{bmatrix} = f_{\boldsymbol{w}}(\boldsymbol{x}) := \begin{bmatrix} e^{\boldsymbol{w}_1^\intercal \boldsymbol{x}}/(e^{\boldsymbol{w}_1^\intercal \boldsymbol{x}} + e^{\boldsymbol{w}_2^\intercal \boldsymbol{x}}) \\ e^{\boldsymbol{w}_2^\intercal \boldsymbol{x}}/(e^{\boldsymbol{w}_1^\intercal \boldsymbol{x}} + e^{\boldsymbol{w}_2^\intercal \boldsymbol{x}}) \end{bmatrix}$$

  Determine all possible values of $\boldsymbol{w} = (\boldsymbol{w}_1, \boldsymbol{w}_2)$ (Hint: Consider the shift invariance.)

Exercise 5 (Properties of convolution - ⋆⋆)**.**

Consider the discrete convolution as defined in class, for simplicity in 1D,

$$(x * h)[n] = \sum_{m=-\infty}^{\infty} x[n-m]h[m].$$

Note: in practice we work with filters h of "finite support"—that is, such that $h[n] = 0$ for $|n| > L$ for some $L < \infty$, but there is no difficulty in working with infinite support filters as long as they decay sufficiently fast. In fact, the math is cleaner.

Show that

- convolution is commutative, $x * h = h * x$;
- it is also associative, $(x * h) * g = x * (h * g)$, meaning that we do not have to write the parentheses.

Consider now cross-correlation, defined rather similarly

$$(x \star h)[n] = \sum_{m=-\infty}^{\infty} x[n+m]h[m].$$

Check whether

- cross-correlation is commutative. (prove it)
- cross-correlation is associative. (prove it)

Exercise 6 (Gradient descent for logistic regression - ⋆⋆⋆)**.**

Let $y^{(n)} \in \{e_1, \dots, e_K\} \subset \mathbb{R}^K$ be one-hot labels and $W \in \mathbb{R}^{K\times d}$. For scores $Wx^{(n)} \in \mathbb{R}^K$ define the softmax probabilities

$$\hat{y}_k^{(n)} = \frac{\exp\big((Wx^{(n)})_k\big)}{\sum_{j=1}^K \exp\big((Wx^{(n)})_j\big)}.$$

and the cross-entropy

$$\ell\big(\hat{y}^{(n)}, y^{(n)}\big) \;=\; -\sum_{k=1}^K y_k^{(n)} \log \hat{y}_k^{(n)}.$$

and the empirical risk

$$R_N(W) \;=\; \frac{1}{N}\sum_{n=1}^N \ell\big(\hat{y}^{(n)}, y^{(n)}\big).$$

Derive the gradient of the empirical risk $R_N(W)$ with respect to W and show that

$$\nabla_W R_N(W) \;=\; \frac{1}{N}\sum_{n=1}^N \big(\hat{y}^{(n)} - y^{(n)}\big)\big(x^{(n)}\big)^\top.$$

Exercise 7 (Translation invariance - ⋆⋆⋆)**.**

Let $x \in \mathbb{R}^{M\times N}$ denote a (grayscale) image and let $h \in \mathbb{R}^{P\times Q}$ be a finite-support filter (impulse response). Define the linear convolution operator H acting on images by

$$(Hx)[m,n] \;=\; (x * h)[m,n] \;=\; \sum_i \sum_j x[m-i,\, n-j]\; h[i,j],$$

with *linear* convolution (assume zero padding outside the image support).
Define the 2-D translation operator $T_{a,b}$ by

$$(T_{a,b}x)[m,n] \;=\; x[m-a,\, n-b],$$

again with zero padding outside the image domain. $T_{a,b}$ shifts the image by a rows and b columns (down/right for positive a, b).

1. Show that convolution commutes with translation, i.e.,

$$T_{a,b}\, H \;=\; H\, T_{a,b}.$$

   (Hint: write both sides as sums and change indices.)

2. Show further that $T_{a,b}$ is itself a convolution with some filter ("impulse response")—which one?

Exercise 8 (Neural network and backpropagation - ⋆)**.**

Consider the one-hidden-layer neural network defined by the function:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{W}_2\sigma(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2, \tag{4}$$

where $\boldsymbol{\theta} = \{\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{b}_1, \boldsymbol{b}_2\}$ represents the set of trainable parameters, and $\sigma(\cdot)$ is an element-wise non-linear activation function. These parameters are optimized using gradient descent update,

$$\boldsymbol{\theta}(t+1) = \boldsymbol{\theta}(t) - \gamma\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}(t)),$$

applied to a dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ with the loss function.

$$L(\boldsymbol{\theta}) = \frac{1}{2N}\sum_{i=1}^{N}(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i)^2.$$

- In a deep (more than one layer) neural network, we sometimes call $\boldsymbol{h}_1 = \sigma(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1)$ as hidden (or intermediate) layer output, or hidden features. Given that the input is $\boldsymbol{x} \in \mathbb{R}^K$ and the label is a scalar $y \in \mathbb{R}$, and assuming the hidden layer dimension is F, what are the dimensions of trainable parameters $\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{W}_1, \boldsymbol{W}_2$?

- For a single training datum $\boldsymbol{x}_1 = [1,1]^\intercal$ and $y_1 = 1$, calculate the updated parameters $\boldsymbol{\theta}(2) = \{\boldsymbol{W}_1(2), \boldsymbol{W}_2(2), \boldsymbol{b}_1(2), \boldsymbol{b}_2(2)\}$ after the second step of gradient descent iteration. Use the initial values:

$$\boldsymbol{W}_1(0) = [1,1], \quad \boldsymbol{W}_2(0) = \boldsymbol{b}_1(0) = \boldsymbol{b}_2(0) = 0,$$

  with the activation function $\sigma(x) = \frac{1}{1+e^{-x}}$ and learning rate as $\gamma = 0.1$.

Coding

Instruction for optional coding submission We use Gradescope to give you a platform where your coding solutions are automatically evaluated. A reminder: Gradescope is optional and has no impact on your final grade–it is simply a tool to help you check whether your code works correctly.
You should have received an invitation to the Gradescope course page. Please go to the course page `10907 Pattern Recognition 2025`, where you will see the corresponding item `Coding: Problem set 2`. To check your solution, upload your completed files `logistic.py`, `classifier.py` and `filter.py` to Gradescope and let the autograder run.

Exercise 9 (Logistic Regression - ⋆⋆)**.**

Logistic regression is a simple and efficient classifier for two-class problems. First, you will complete the functions that are core to the logistic regression classifier and then apply the classifier to a simple dataset.

1. Implement six functions that form the logistic regression classifier in the file `logistic.py`. The functions to be completed are:

   (a) `sigmoid()`: This takes an array as input and gives the sigmoid of each value of the array. The sigmoid function for a point $z \in \mathbb{R}$ is defined as :

   $$\text{sigmoid}(z) = \frac{1}{1+e^{-z}}$$

   (b) `predict_score()`: Takes the weight vector $w \in \mathbb{R}^D$ of size D and a matrix $X \in \mathbb{R}^{N \times D}$, with N data points of size D, and outputs the logistic regression probability (score). For a single data point $x \in \mathbb{R}^D$, the score is defined as:

   $$\text{score}(x, w) = \frac{1}{1+e^{-w^T x}}$$

   Note: The above definition is for a single example. However, the function takes multiple examples as input in the form of a matrix. You can do this without using a `for` loop.

   (c) `predict()`: Predicts the class 0 or 1 using the data matrix X, weight vector w, and a threshold. This is defined as:

   $$\text{predict}(x, w, \text{threshold}) = \mathbb{1}_{\text{score}(x,w) \geq \text{threshold}}.$$

   (d) `cross_entropy_loss()`: Outputs the cross-entropy loss (CE) given the predicted probabilities and the true labels. This is defined as:

   $$CE(y_{\text{pred}}, y) = -\frac{1}{N}\left(\sum_{i=1}^{N} y[i]\log(y_{\text{pred}}[i]) + (1-y[i])\log\left(1-y_{\text{pred}}[i]\right)\right).$$

   Note: We use the natural logarithm.

   (e) `gradient(X,w,y)`: Computes $\nabla_w CE(\text{sigmoid}(Xw), y)$ and returns it. This is defined as:

   $$\begin{aligned} y_{\text{pred}} &= \texttt{predict\_score}(X, w) \\ \texttt{gradient}(X, w, y) &= \nabla_w CE(y_{pred}, y) \end{aligned}$$

   (f) `train()`: Using an initial weight vector w_{init}, learn a new estimate of the weight vector using gradient descent for a predefined number of iterations (`epochs`) and a given learning rate `lr`. The pseudocode for the training algorithm is given in Algorithm 1.

Algorithm 1 Gradient Descent

```
Require: $X \in \mathbb{R}^{N \times D}$, $w\_init \in \mathbb{R}^D$, $y\_true \in \{0,1\}^N$, $epochs \geq 0$, $lr \geq 0$
  $w \leftarrow w\_init$
  $losses = [\,]$
  $i \leftarrow 0$
  while $i <$ epochs do
      $i \leftarrow i + 1$
      $y\_pred \leftarrow predict\_score(X, w)$
      $loss \leftarrow cross\_entropy\_loss(y\_pred, y\_true)$
      $losses.append(loss)$
      $w \leftarrow w - lr * gradient(X, w, y\_true)$
  end while
```

2. Once implemented, you will apply the classifier to the rice dataset provided in the file `Rice.csv`. The dataset consists of various physical features of two types of grains known as 'Cammeo' and 'Osmancik'. The features include: Area, Perimeter, Major Axis Length, Minor Axis Length, Eccentricity, Convex Area, and Extent observed for different grains of rice from the two species. More details about the data is present in [1].

   We have provided you with a skeleton file called `classifier.py` with all the necessary libraries loaded. You will first preprocess the dataset, apply the classifier, and report any preprocessing you do and test accuracy. Your task is divided as follows:

   (a) Preprocess: Once the data is loaded, you need to convert the target variable `y` into binary variable. You can choose either of the two species as class 1. If required, you can preprocess the dataset provided by the variable `X`. Since all features are numeric and on different scales, consider preprocessing such as:

      i. Standardization: makes the numerical feature zero mean and unit variance.
      ii. Min-Max Scaling: scales features to a specified range (e.g. [0, 1]).
      iii. Log Transformation: can be used for features with skewed distributions to make them more normally distributed.

      You can choose one of the above or anything else you can think of and apply it to the dataset.

   (b) Split the data into training and test sets using the `train_test_split` function from the sklearn library.

   (c) Train the classifier on the training set and evaluate it on the test set (i.e., compute test accuracy and plot the loss over epochs). Experiment with different learning rates and numbers of epochs. *Note:* A loss curve plots the number of epochs on the x-axis and the loss value on the y-axis.

Exercise 10 (DL warm up - ⋆⋆).

Develop one-hidden-layer neural networks as described in Exercise 8 with $\sigma(\cdot) = \text{ReLU}(\cdot)$. Train this model using the following dataset, which consists of 9 data points sampled from the quadratic function $y = x^2$:

x	-1.2	-0.9	-0.6	-0.3	0	0.3	0.6	0.9	1.2
y	1.44	0.81	0.36	0.09	0.	0.09	0.36	0.81	1.44

Table 1: Training dataset sampled from $y = x^2$

Illustrate the performance of your trained network by plotting the results for $x \in [-1.5, 1.5]$. Compare the results with different hidden layer dimensions: $F = 2, 5, 10, 50, 100$, and 500. Be aware that the loss may not always diminish to zero.

Your plots cannot be autograded by Gradescope. To help you calibrate your results, we provide two example plots at the end of the sheet for hidden sizes $F = 2$ and $F = 500$. Use these as qualitative references to compare the shape of your fitted curves and the overall trend as F increases. Small numerical or visual differences are expected due to initialization and optimization choices. What matters is that your curves exhibit similar behavior (e.g., limited capacity and visible underfitting for $F = 2$, much higher flexibility and closer fit for $F = 500$).

Exercise 11 (Convolution and Filtering - ⋆⋆).

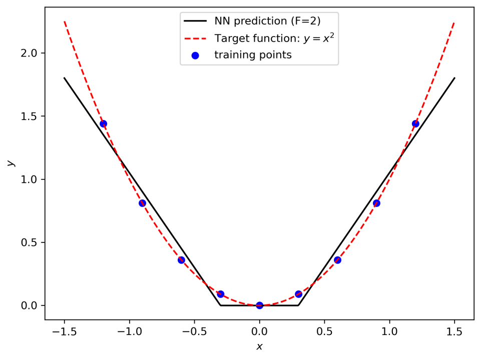
Convolution and filtering are fundamental operations when working with signals or images. We have provided you with a skeleton code in the file `filter.py` to perform basic 2D convolutions and some simple Gaussian low-pass and high-pass filtering. 实现卷积的两种方式

1. Convolutions can be implemented in two ways, either in the Fourier domain or the image domain. In both cases, the function takes an image of size $K \times K$ and a filter of size $k \times k$ and `mode` as input. If `mode = 'same'` the output has the same dimension as the input image. In this case, we want to compute linear rather than circular convolution, so you will have to zero-pad the input image accordingly and then crop the result to the target dimensions. If `mode='valid'`, then you should only return that part of the output image that does not depend on zero padding. Thus the output is of size $(K-k+1)\times(K-k+1)$. Your task is to implement:
   (a) `convolve2d()`: performs convolution directly in the image domain, using for loops.
   (b) `convolve2d_fft()`: performs convolution in the Fourier domain.
   Note:
      i. Use NumPy's FFT routine.
      ii. While performing convolution in the Fourier domain, you need to first perform the filtering and then crop the result.

   (Optional) Compare computation time of `convolve2d` and `convolve2d_fft` on a large image of your choice.

2. A Gaussian low-pass filter retains only smooth features of the image while removing high-frequency, potentially noisy components. For a given standard deviation η, the filter is of spatial dimension $(2m+1)\times(2m+1)$ where $m = \lceil 4\eta \rceil$. The weights of the filter are defined as:
$$W^{\text{low-pass}}[n_1, n_2] = ce^{-(n_1^2+n_2^2)/(2\eta^2)},$$
   where $n_1 \in \{-m, -m+1, \dots, 0, \dots, m\}$, $n_2 \in \{-m, -m+1, \dots, 0, \dots, m\}$ and c is chosen such that $\sum_{n_1=-m}^{m}\sum_{n_2=-m}^{m} W[n_1, n_2] = 1$. The high-pass filter is just the complement of the low-pass, $W^{\text{high-pass}}[n_1, n_2] = \delta[n_1, n_2] - W^{\text{low-pass}}[n_1, n_2]$, which allows only the high-frequency components of the image. The function $\delta[n_1, n_2]$ is an identity filter defined as
$$\delta[n_1, n_2] = \begin{cases} 1 & \text{if } n_1 = 0 \text{ and } n_2 = 0 \\ 0 & \text{otherwise.} \end{cases}$$
   Applying this filter doesn't change the image. Your task is to implement:
   (a) `gaussian_lowpass()`: Which takes the standard deviation η as input and outputs a Gaussian low-pass filter of size $(2m+1)\times(2m+1)$ where $m = \lceil 4\eta \rceil$.
   (b) `gaussian_highpass()`: Outputs a Gaussian high-pass filter.
   (c) Using any one of the images in the scikit-image dataset (link), show the effect of a high-pass and low-pass filter by plotting the chosen image along with the low-pass filtered and high-pass filtered images side by side. Report the variance and filter size used. Report the convolution operation used and what is the computational complexity of this convolution?
   Note: For a color image, make sure to convert it into a grayscale image first.

Reference plots for Exercise 8



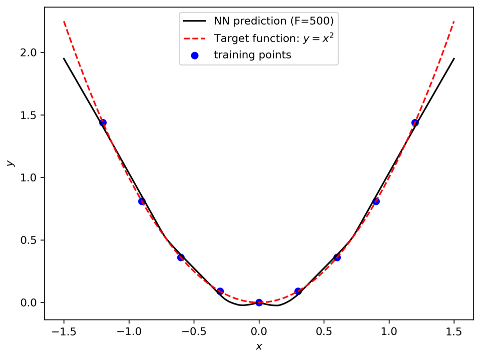


Figure 1: Sample plots showing the results of the one-hidden-layer neural network with $F = 2$ (left) and $F = 500$ (right).

References

[1] Rice (Cammeo and Osmancik), UCI Machine Learning Repository, 2019, `https://doi.org/10.24432/C5MW4Z`.