CSE517A – Homework 3

M. Neumann

04 Apr 2025

- Please keep your written answers brief and to the point. Incorrect or rambling statements can hurt your score on a question.
- If your hand writing is not readable, we **cannot give you credit**. We recommend you type your solutions in LaTeX and compile a .pdf for each answer. **Start every problem on a new page!**
- This will be due **04 Apr 2025** at **11:59pm** with an automatic 3-day extension.
- You may work in groups of at most 2 students.
- Submission instructions:
  - Start every problem on a **new page**.
  - Submissions will be exclusively accepted via **Gradescope**. Find instructions on how to get your Gradescope account and submit your work on the course webpage.

Note, that if you use *any* **resources** outside the course materials to derive (part of) your solution, you will need to cite the source in your homework submission. This also holds for **online sources**. If you collaborate with anyone other than your partner, it is your responsibility to indicate this in your submission. Course materials are the lecture notes, course books, and resources linked therein or on Canvas, plus course materials of any prerequisite course officially listed as such in the course listing.

Citing the source(s) does **not** legitimate the *copying* of existing solutions to any given problem, neither does it legitimate that another person (that is not you or your partner) directly solves any (part of the) problems for you. Please, refer to the **course syllabus** for more details.

Problem 1 (*20 points*) Another Look at SVM

You suddenly have this vision, that it might be beneficial to pre-compute all inner-products in your data set. I.e. you store a $n \times n$ matrix $K_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$

- **(a)** (*10 pts*) Imagine you start the gradient descent procedure from above with initial weights $\mathbf{w}^{(0)} = 0$ (this is just to help intuition). Take the gradient update of SVM (**hw1 p2(d)**) and show that after t gradient steps you can express the weight vector as $\mathbf{w}^{(t)} = \sum_{i=1}^{n} \alpha_i^{(t)} \mathbf{x}_i$ for some values $\alpha_i^{(t)}$. (HINT: each gradient step update the α's, if the update is correct you can always write $\mathbf{w}^{(t)} = \sum_{i=1}^{n} \alpha_i^{(t)} \mathbf{x}_i$ starting from update $t = 0$ to any update $t > 0$)

- **(b)** (*5 pts*) Take this new definition $\mathbf{w} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i$ and substitute it into the objective function of SVM. You can now write the objective function as $\mathcal{L}(\boldsymbol{\alpha})$ where $\boldsymbol{\alpha} = [\alpha_1 \ldots \alpha_n]^\top$ (no need to force the vector representation in your formula). Further, write the objective $\mathcal{L}(\boldsymbol{\alpha})$ only using the precomputed matrix entries K_{ij}.

- **(c)** (*5 pts*) Derive the gradient descent update rule of SVM (**hw1, p2(d)**) with respect to the new parameters α_i.

Convince yourself that you just **kernelized the SVM training algorithm** (in the *primal space*) ☺

Problem 2 (*20 points*) Pairwise Euclidean Distances

Many machine learning algorithms (such as k-NN or the dual form of the SVM) access their input data primarily through pairwise distances. It is therefore important that we have a fast function that computes pairwise (Euclidean) distances of input vectors.

Assume we have two sets of input vectors $\{\mathbf{x}\}$ and $\{\mathbf{z}\}$ stored in two matrices $X = [\mathbf{x}_1, \ldots, \mathbf{x}_n] \in \mathbb{R}^{d\times n}$ where $\mathbf{x}_i \in \mathbb{R}^{d\times 1}$ and $Z = [\mathbf{z}_1, \ldots, \mathbf{z}_m] \in \mathbb{R}^{d\times m}$ where $\mathbf{z}_j \in \mathbb{R}^{d\times 1}$. Our objective is to calculate the Euclidean distances between *all* $\mathbf{x}_i$ and $\mathbf{z}_j$, denoted by the matrix $D \in \mathbb{R}^{n\times m}$ such that

$$D_{ij} = d(\mathbf{x}_i, \mathbf{z}_j) = \sqrt{(\mathbf{x}_i - \mathbf{z}_j)^T (\mathbf{x}_i - \mathbf{z}_j)}$$

Although there is an execution overhead per line in Python, matrix operations are extremely optimized and very fast with the numpy package. In the following you will transform the function above, so that it can be computed solely through matrix operations *without the use of any loops at all.*

- **(a)** (*5 pts*) Show that the inner-product matrix $G \in \mathbb{R}^{n\times m}$ where

$$G_{ij} = \mathbf{x}_i^T \mathbf{z}_j$$

  can be expressed in terms of matrix multiplication.
- **(b)** (*10 pts*) Let us define two new matrices $S, R \in \mathbb{R}^{n\times m}$ where

$$S_{ij} = \mathbf{x}_i^T \mathbf{x}_i; R_{ij} = \mathbf{z}_j^T \mathbf{z}_j.$$

Show that the squared Euclidean distance matrix $D^2 \in \mathbb{R}^{n\times m}$, defined as

$$D_{ij}^2 = (\mathbf{x}_i - \mathbf{z}_j)^T(\mathbf{x}_i - \mathbf{z}_j)$$

can be expressed as a linear combination of S, R, and G. (HINT: It might help to first express D_{ij}^2 in terms of inner-products. HINT 2: Don't forget to show how you can compute S and R using numpy array operations.)

(c) (*5 pts*) Due to *numerical instability*, some of the entries in D may turn out to be less than 0. How can you avoid this? Again use matrix operations in numpy.

Problem 3 (*25 points*) Valid Kernels, Kernel Construction

(a) (*5 pts*) Consider $\mathbf{x}_i \in \mathbb{R}^3$ and $\phi(\mathbf{x}_i) \in \mathbb{R}^D$ with

$$\phi(\mathbf{x}_i) = [1, \sqrt{2}(\mathbf{x}_i)_1, \sqrt{2}(\mathbf{x}_i)_2, \sqrt{2}(\mathbf{x}_i)_3, (\mathbf{x}_i)_1^2, (\mathbf{x}_i)_2^2, (\mathbf{x}_i)_3^2, \sqrt{2}(\mathbf{x}_i)_1(\mathbf{x}_i)_2, \sqrt{2}(\mathbf{x}_i)_1(\mathbf{x}_i)_3, \sqrt{2}(\mathbf{x}_i)_2(\mathbf{x}_i)_3]^T$$

compute $\langle\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle$ as a function of $\mathbf{x}_i$ and $\mathbf{x}_j$. What is D?
Briefly argue why considering the inner products directly instead of explicitly computing the feature mappings is more efficient.

(b) (*5 pts*) For each of the following matrices show whether it is strictly positive definite, positive semidefinite, or neither:

$$A_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 0 \\ 1 & 0 & 2 \end{bmatrix}, \quad A_3 = \begin{bmatrix} 2 & 1 & -1 \\ 1 & 1 & 1 \\ -1 & 1 & 2 \end{bmatrix}.$$

(c) (*10 pts*) Show that the RBF Kernel corresponds to an inner product in an infinte dimensional space. (HINT: use Taylor expansion.)

(d) (*5 pts*) By using the **definition of a kernel**, verify that $\tilde{k}(\mathbf{x}, \mathbf{x}') = ck_1(\mathbf{x}, \mathbf{x}')$ is a valid kernel, given that $k(\mathbf{x}, \mathbf{x}')$ is a valid kernel and $c \geq 0$.

(e) **[ungraded]** Verify all other kernel construction rules as an additional practice.

HINT: to prove the rules there are a couple of different things to try. Note that not everything works for every rule.

- use the definition of a kernel (that is, construct a feature space, where $k(.,.)$ corresponds to the inner product)
- use the spectral decomposition $k_1(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{\infty} \lambda_i \phi_i(\mathbf{x})\phi_i(\mathbf{x}')$, where λ_i are the eigenvalues in the transformed feature space (cf. *Mercer's Theorem*)
- show that the Gram matrix corresponding to any arbitrary subset of elements in the input space is positive-semi definite by
  - showing that the quadratic form is non-negative for arbitrary vectors
  - showing that all eigenvalues are non-negative
- use already proven construction rules

Problem 4 (*15 points*) Clustering with Kernel k-Means

(a) (*10 pts*) Kernelize the k-means algorithm.

(b) (*5 pts*) Why is it not possible to visualize the cluster centers (even if our input space is 2d)?

Problem 5 (*20 points*) Gaussian Processes (GPs)

For simplicity you may assume zero-mean observations for the entire problem.

(a) (*5 pts*) Assuming noise-free training data $D = \{(\mathbf{x}_i, f_i)\}_{i=1,\dots,n}$ with $f_i = f(\mathbf{x}_i)$, show that the variance cov_{f_i} for the GP prediction for a *training point* $\mathbf{x}_i$ is 0.

(b) (*5 pts*) Despite being a non-parametric model, we still have to learn the kernel parameters $\boldsymbol{\theta}$ for a Gaussian process. Those so called *hyperparameters* can be learned by maximizing the probability of observing the training data given the GP prior. This can be formally expressed by the *marginal likelihood*[1] $p(\mathbf{y} \mid X, \boldsymbol{\theta})$. Luckily for standard GPR, this marginal likelihood can be computed in closed form. Derive the **analytic log marginal likelihood expression** (assuming a zero-mean GP prior[2], noisy observations with Gaussian i.i.d. noise $\epsilon \sim \mathcal{N}(0, \sigma_n)$, and a parameterized covariance/kernel function). Use $K_{\boldsymbol{\theta}}$ to indicate that the kernel matrix $K = \text{k}(X, X)$ depends on the kernel and noise model parameters $\boldsymbol{\theta}$.

(c) (*5 pts*) When implementing GPs (prediction and learning method), we aim to compute the required inverse matrix $K_{\boldsymbol{\theta}}^{-1}$ as efficient as possible. To do so, we leverage the Choleskey decomposition of $K_{\boldsymbol{\theta}} = LL^T$, where L is a lower triangular matrix. State the log marginal likelihood in terms of $\boldsymbol{\alpha}$ and L, where $\boldsymbol{\alpha} = L^\top \backslash (L \backslash \mathbf{y})$ and $\backslash$ denotes left-division indicating that we solve a system of linear equations $L\mathbf{b} = \mathbf{y}$ for $\mathbf{b}$. (i.e., $L\backslash\mathbf{y} = L^{-1}\mathbf{y} = \mathbf{b} \Leftrightarrow L\mathbf{b} = \mathbf{y}$).

(d) (*5 pts*) For learning our goal is to pick the hyperparameters $\boldsymbol{\theta}$ that maximize the log marginal likelihood (or minimize the negative log marginal likelihood). Compute the **derivative of the negative log marginal likelihood** starting from your expression derived in part (a). Once you have your derivative, state this equation again in terms of $\boldsymbol{\alpha}$ and L. This expression is used in a gradient descent procedure in a practical implementation.

[1] *marginal* refers to the fact that the model parameters $\mathbf{w}$ from the weight-space derivation of GPs (or f when using the function space view) are marginalized out.

[2] We can always make our observations have zero mean by centering the y_i's by the mean of the training observations $\bar{y}$.