LINMA2472 Algorithms in Data Science

January 2025

Instructions

1. Write your name, surname, NOMA on each sheet.
2. Write in the frames, not outside (your exam will be scanned and only the text within the frame will appear on-screen for the grader)
3. The exam lasts 3 hours.
4. The back face (verso) of each sheet is not read, you can use it as draft paper.
5. You can use your own blank paper as draft paper.
6. Calculators and other electronic devices are forbidden.
7. You must hand in everything at the end of the exam : questions, answers, drafts.
8. Please hand in the questions in the correct order.

UCLouvain EPL Faculty January 2025	**Exam : LINMA2472** Algorithms in data science	First Name : Last Name : NOMA :

Question 1 : Kernels for prediction of gene expression

One would like to predict, from the DNA sequence of an individual (or an appropriate subsequence), the level of expression of a particular gene (for instance, in order to estimate the predisposition of said individual to a disease). A DNA (sub)sequence is a string of n letters taken in a four-letter alphabet (A,C,G,T).

The gene expression level is a real number. We thus must build a map from n-letter strings to the reals. We may want to infer this map from N examples (s_i, r_i) (for $i = 1, \ldots, N$), where s_i is an n-letter string (a DNA sample) and r_i a real number (a gene expression level). This is a regression task. Here we propose a kernel ridge regression to solve it.

Given two strings s, s' of length n on the A,C,G,T alphabet, and an integer $\ell < n$ let us consider $k_\ell(s, s')$ as the number of ℓ-letter strings present (at least once) in both s and s'. For instance if $s = ACACACGT$ and $s' = ACGTCACA$, and $\ell = 3$, we find $k_3(s, s') = 4$, because ACA, CAC, ACG and CGT appear (at least once) in both s and s'.

— Build a map $s \mapsto \phi(s) \in \mathbb{R}^d$ into a feature space $\mathbb{R}^d$ so that $k_\ell(s, s') = \langle \phi(s), \phi(s') \rangle$. Express d as a function of ℓ and n. Here $\langle ., . \rangle$ is the usual scalar product in $\mathbb{R}^d$.

Solution:

Thus $k_\ell(., .)$ is a kernel map indeed. Call K_ℓ the corresponding N-by-N kernel matrix.

— Justify that the regression map found by the kernel ridge regression method (for any given choice of ℓ) can be written as $s \mapsto \sum_{i=1}^N w_i k_\ell(s_i, s)$, for some w_i to be found. You may use facts and theorems stated in the lectures (and cite them clearly).

Solution:

— Formulated in terms of $\mathbf{w} = (w_1, \ldots, w_N)$, the kernel ridge regression problem can be written as

$$\min_{\mathbf{w} \in \mathbb{R}^N} L(\mathbf{w})$$

for some objective function. Write down the objective function $L(\mathbf{w})$. Explain briefly each term.

Solution:

— What is the computational complexity (i.e., computation time) of deriving the regression map (i.e., computing the vector w) ? Justify briefly. Your expression should depend of N, n, ℓ.

You may accept and use the following facts or assumptions : that $\ell \ll n$; that matrix K is best computed entrywise (all N^2 entries separately) ; that computing $k_\ell(s, s')$ is done (for $\ell \ll n$) in time $\mathcal{O}(\ell n)$ (which means "in time no larger than $c\ell n$, for some constant c") for each given pair of strings s,s' ; that solving an N-by-N linear system $Ax = b$ (for some N-by-N matrix A and some vector b) is done in time $\mathcal{O}(N^3)$ in general.

Solution:

— What is the computational complexity (i.e., computation time) of computing the regression map on a new string s ? Justify briefly. Your expression should depend of N, n, ℓ. Here we assume that the kernel matrix K and the vector w have already been computed.

Solution:

— **Bonus** We could use a classification approach instead of regression. We can consider that the gene of interest is either "expressed" (if $r \geq r_0$, for some appropriate threshold r_0) or "non-expressed" (if $r < r_0$). We may now use a Kernel SVM method. Compare this approach (in terms of respective strengths or weaknesses) with the regression approach on three different aspects (e.g. relevance for the ap-

plication, computational complexity, simplicity, interpretability, etc.). This is an open question, with a range of sensible answers.

Solution:

NB : many machine-learning methods have been proposed by researchers for this important problem, involving kernels (more sophisticated than the one above), or CNN, transformers, etc.

Question 2 : Automatic Differentiation and Attention

Consider the matrices $V \in \mathbb{R}^{d_v \times n_{\text{ctx}}}, K \in \mathbb{R}^{d_k \times n_{\text{ctx}}}$ and an attention head applied to a vector q : $a(q) = V\text{softmax}(K^\top q/\sqrt{d_k})$ where the i-th entry of $\text{softmax}(x)$ is $\exp(x_i)/\sum_j \exp(x_j)$. Suppose that you want to compute Jacobian L of $a(q)$, which is the derivative of all outputs of the attention head with respect to each entry of the vector q. In other words, $L_{ij} = \partial a_i/\partial q_j$.

— Find the matrix J to be used to compute the derivative of a with respect to the j-th entry of q with the formula

$$\partial a/\partial q_j = VJK^\top e_j/\sqrt{d_k}.$$

Hint : The entries of the matrix J can be written purely in terms of the entries of the vector $s = \text{softmax}(K^\top q/\sqrt{d_k})$.

Solution: Let $x = K^\top q/\sqrt{d_k}$. We have

$$\begin{aligned} J_{ij} &= \partial s_i/\partial x_j \\ &= \frac{\partial}{\partial x_j}\frac{\exp(x_i)}{\sum_{k=1}^{n_{\text{ctx}}}\exp(x_k)} \\ &= \frac{\exp(x_i)}{\sum_{k=1}^{n_{\text{ctx}}}\exp(x_k)}\frac{\partial x_i}{\partial x_j} - \frac{\exp(x_i)}{(\sum_{j=1}^{n_{\text{ctx}}}\exp(x_j))^2}\frac{\partial}{\partial x_j}\sum_{k=1}^{n_{\text{ctx}}}\exp(x_k) \\ &= s_i\frac{\partial x_i}{\partial x_j} - s_i s_j. \end{aligned}$$

So the entries of J are :

$$\begin{aligned} J_{ii} &= s_i - s_i^2 \\ J_{ij} &= -s_i s_j \qquad\qquad i \neq j. \end{aligned}$$

— The previous question corresponds to *forward* differentiation. Using *reverse* differentiation, you would like now to compute the gradient of a_i with respect to the vector q. How can you compute this gradient vector via matrix-vector products ? *Hint :* $\partial a_i/\partial q_j$ is the scalar product between $\partial a/\partial q_j$ and e_i.

Solution: As J is diagonal, it is its own transpose.

$$\begin{aligned}\partial a_i/\partial q_j &= \langle VJK^\top e_j/\sqrt{d_k}, e_i\rangle \\ &= \langle e_j, KJ^\top V^\top e_i/\sqrt{d_k}\rangle \\ \partial a_i/\partial q &= KJV^\top e_i/\sqrt{d_k}\end{aligned}$$

— Depending on d_v, d_k, n_{ctx}, which one will be faster between forward and reverse differentiation to compute the **full** Jacobian matrix $\partial a/\partial q$? Why ?

Solution: Forward diff concatenates $\partial a/\partial q_j$ horizontally for each j and reverse diff concatenates $\partial a_i/\partial q$ vertically for each i. In other words, forward diff computes $V(JK^\top)$ while reverse diff computes $K(J^\top V^\top)$ or equivalently $(VJ)K^\top$. The complexity of forward diff is $O(n_{\text{ctx}}^2 d_k + n_{\text{ctx}} d_k d_v)$ while the complexity of reverse diff is $O(n_{\text{ctx}}^2 d_v + n_{\text{ctx}} d_k d_v)$. This means that forward diff is faster if $d_k < d_v$, otherwise reverse diff is faster.

— How could this computation be accelerated using a GPU instead of a CPU ?

Solution: As these are matrix-matrix products, this computation is highly parallelizable and hence will get a good speed up on a GPU.

Question 3 : Stable Diffusion for music generation

Consider the problem of music generation given a text prompt. For instance, given "classic jazz", the model should generate a music of style "classic jazz". The music is represented by a sequence $(y_k)_{k=1}^N$ of real numbers of fixed length N. These represent the intensity of the music sampled at a given fixed rate (for instance every millisecond).

— Draw the model and different components that you would use for this :

Solution: *(sketch)* Transformer encoder for the text prompt which is then given by a diffusion model with cross-attention. The diffusion model starts with a random vector of N numbers and then generates the music.

— Describe the datasets needed and the procedure to be used to train each component of the model.

Solution: *(sketch)* First train encoder and then diffusion model.

— Given a text prompt, describe the procedure to generate a new music.

Solution: *(sketch)* Classifier-Free Guidance.

— Explain how you can use an auto-encoder to prevent the model from learning imperceptible details of the music. How do you adapt the training and inference process with this new component ?

Solution: *(sketch)* Use Variational Auto-Encoder to first compress the music.

— Suppose that you want to generate a "classic jazz" music that, when reversed in time sounds like "classic rock". How would you use your already trained model to achieve this task without retraining ?

Solution: *(sketch)* Like project.