Introduction to machine learning
Exercise 2

Fall 2024/25

Submission guidelines, **read and follow carefully**:

- The exercise **must** be submitted in pairs.
- Submit via Moodle.
- The submission should include two separate files:
  1. A PDF file that includes your answers to all the questions.
  2. The code files for the python question. You must submit a copy of the shell python file provided for this exercise in Moodle, with the required functions implemented by you. **Do not change the name of this file.** In addition, you can also submit other code files that are used by the shell file.
- Your python code should follow the course python guidelines. See the Moodle website for guidelines and python resources.
- Before you submit, **make sure that your code works in the course environment**, as explained in the guidelines. Specifically, **make sure that the test `simple_test` provided in the shell file works**.
- You may only use python modules that are explicitly allowed in the exercise or in the guidelines. If you are wondering whether you can use another module, ask a question in the exercise forum. No module containing machine learning algorithms will be allowed.
- For questions, use the exercise forum, or if they are not of public interest, send them to the course staff email intromlbgu25@gmail.com.
- Grading: Q.1 (python code): 10 points, Q.2: 20 points, Q.3: 18 points, Q.4: 18 points, Q.5: 16 points, Q.6: 18 points

Question 1. Implement the soft-SVM algorithm that we learned in class in python. The shell file "softsvm.py" is provided for this exercise in Moodle. It contains an empty implementation of the function required below. You should implement it and submit according to the submission instructions.

```
def softsvm(l, trainX, trainy)
```

The input parameters are:

- `l` - the parameter λ of the soft SVM algorithm.
- `trainX` - a 2-D matrix of size $m \times d$, where m is the sample size and d is the dimension of the examples. Row i in this matrix is a vector with d coordinates that describes an example x_i from the training sample.
- `trainy` - a column vector of length m. The i's number in this vector is the label $y_i \in \{-1, 1\}$ from the training sample.

The function returns the linear predictor `w` which is a column vector in $\mathbb{R}^d$.

- You may assume all the input parameters are legal.
- We will use the library `cvxopt` for our Quadratic Program solver.

Instructions for using `cvxopt`:

- First, you will need to define the matrices `H`, `u`, `A`, and `v` which correspond to the vectors and matrices with the same names in the quadratic programming problem you learned in class. Those matrices should be `cvxopt` matrices, check how to create `cvxopt` matrices or convert `numpy` arrays to `cvxopt` matrices here: `http://cvxopt.org/userguide/matrices.html`.
- In order to conserve memory, use sparse matrices when possible.
- Run `sol = cvxopt.solvers.qp(H, u, -A, -v)` to solve the quadratic programming problem. Here, we pass A and v with a minus sign, since this solver assumes the constraints are $Az \leq v$, while in class we assumed they were $Az \geq v$. The solution of the quadratic program is provided in `sol["x"]`.
- See the note at the end of the exercise regarding a possible error and how to solve it.

Question 2. In this question, you will run your soft SVM implementation on data from the MNIST dataset you saw in exercise 1. For this task, we took a subset of this dataset which includes the digits 3 and 7, and the goal of the predictor is to distinguish between the two digits. You can load the dataset, which is already divided to train and test, from the file `EX2q2_mnist.npz` on the course website.

Run two experiments on this data set. In the first experiment, use a sample size of 100. To generate this small sample, draw it randomly from the provided training sample. Repeat the "small sample" experiment 10 times, and when you report the results, average over these 10 experiments, and plot also error bars which show the maximum and minimum values you got over all experiments. Run your soft-SVM implementation with each of the following values of λ: $\lambda = 10^n$, for $n \in \{-1, 1, \dots, 9, 11\}$.

In the second experiment, use a sample size of 1000, which you should also draw randomly from the training set. Run your soft-SVM implementation with each of the following values of λ: $\lambda = 10^n$, for $n \in \{1, 3, 5, 8\}$. To make the running time feasible, you should run this experiment only once for each value of λ.

(a) Submit a plot of the training error and test error of the small sample size results as a function of λ (plot λ on a logarithmic scale), with one line for the train error and another line for the test error. Each line should show an average of the 10 experiments, and error bars which show the maximum and minimum values you got over all experiments.

(b) Add to the plot the points describing the training error and test error of the large sample size. For this part, don't draw lines between the points in this case, only show each point individually, since you tested values of λ which are quite far away from each other.

(c) Based on what we learned in class, what would you expect the results to look like? Do the results you got match your expectations? In your answer address the following issues:

- Which sample size should get a smaller training error? What about test error? Do the results match your expectations?
- What should be the trend in the *training error* as a function of λ (decreasing/increasing/other)? Why? Do the results (for the small sample size) match your expectations?
- What should be the trend in the *test error* as a function of λ (decreasing/increasing/other)? Why? Do the results (for the small sample size) match your expectations?

Question 3. Let the example domain be $\mathcal{X} = \mathbb{R}^d$ and the label domain be $\mathcal{Y} = \{-1, +1\}$. For a given training sample $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, consider the following **modified version** of the soft-SVM optimization problem:

$$\text{Minimize}_{w \in \mathbb{R}^d} \lambda \|w\|_2^2 + \sum_{i=1}^{m} \left[\ell_h(w, (x_i, y_i))\right]^2 ,$$

where $\ell_h(w, (x, y)) = \max\{0, 1 - y\langle w, x\rangle\}$ is the *hinge loss* defined in class.

Express the above optimization problem as a quadratic program in standard form, as we showed in class.

(a) Write a quadratic minimization problem with constraints that is equivalent to the problem above, using auxiliary variables similar to the ξ_i in the soft-SVM implementation.

(b) Write what H, u, A, v in the definition of a Quadratic Program should be set for solving the minimization problem you wrote above.

Question 4. Prove or refute the following claims:

(a) Recall the soft-SVM objective

$$\min_{w \in \mathbb{R}^d} \lambda \|w\|_2^2 + \frac{1}{m} \sum_{i=1}^{m} \ell_h(w, (x_i, y_i)).$$

For a separable sample $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, the soft-SVM objective with $\lambda = 0$ has infinitely many solutions.

(b) For a separable sample S, there always exists $\lambda > 0$ such that the soft-SVM solution w satisfies $\widehat{\text{err}}(h_w, S) = 0$.

(c) There exists a value $\lambda > 0$ such that for all sample S of $m > 1$ examples, which is separable by the class of homogeneous linear predictors, the soft-SVM (with parameter λ) and the hard-SVM learning rules return exactly the same weight vector.
Note: To prove you need to show that for a fixed m there exists λ such that for all the samples S of size m, hard-SVM and soft-SVM (with parameter λ) return exactly the same weight vector. To refute you need to show that for every λ and m, there exists a sample S of size m such that hard-SVM and soft-SVM (with parameter λ) return different weight vector.

(Hint: Consider a scenario where all m labels are identical).

Question 5. Consider an input space $\mathcal{X} = \{x \in \mathbb{R}^d \mid \|x\|_2^2 \leq d\}$ and a label space $\mathcal{Y} = \{-1, 1\}$. A separable sample $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ of m examples from $\mathcal{X} \times \mathcal{Y}$ is given.

The vector $\widetilde{w} \in \mathbb{R}^d$ has q components of value 1 and the rest are 0. The linear predictor defined by $\widetilde{w}$ separates the sample S. Moreover,

$$\min_{i \in \{1,\ldots,m\}} y_i \langle \widetilde{w}, x_i \rangle = q.$$

The linear predictor defined by $w^* \in \mathbb{R}^d$ has the largest sample margin among all the separators of the sample S. Specifically, the sample margin of w^* is larger than the sample margin of $\widetilde{w}$.

Let T_p be the number of iterations until the Perceptron algorithm stops for the given sample S.

Note that the Perceptron algorithm does **not** use w^* or $\widetilde{w}$. However, w^* and $\widetilde{w}$ can be used for formulating an upper bound on the number of Perceptron iterations.

Formulate an upper bound on the number of Perceptron iterations T_p. The formula should be based on the problem description in this question and explicitly use d and q.

The solution should include a detailed mathematical proof.

Question 6. Consider a prediction problem where the input space is $\mathcal{X} = \mathbb{R}^d$ and $d > 5$; and the label space is $\mathcal{Y} = \{-1, 1\}$.

For an integer $a \geq 2$, the function

$$Q_a(x, x') = 2^{x(1)+x'(1)} + \frac{1}{x(3)x'(3)} + a^{x(4)+x'(4)+x(5)+x'(5)}$$

is defined for $x, x' \in \mathbb{R}^d$.

(a) Prove that $Q_a(x, x')$ is a kernel function.
For the proof, formulate a function $\psi : \mathcal{X} \to \mathcal{F}$ which is a possible feature map (for feature space $\mathcal{F}$) that proves that $Q_a(x, x')$ is a kernel function.

(b) Let V be a validation set of n examples from $\mathcal{X} \times \mathcal{Y}$ that are statistically independent from the training sample S. Consider the selection of a for the kernel function Q_a; the set of examined values for a is $A = \{\widetilde{a}_1, \ldots, \widetilde{a}_r\}$, which includes r integers greater than 2.
For $a \in A$, denote $\widehat{h}_a : \mathcal{X} \to \mathcal{Y}$ as the predictor given by a learning algorithm that uses the kernel Q_a.
Formulate the maximal value of r for which it is guaranteed that

$$\mathbb{P}\left[\forall a \in A, |\widehat{\text{err}}(\widehat{h}_a, V) - \text{err}(\widehat{h}_a, \mathcal{D})| \leq \epsilon\right] \geq 1 - \delta$$

for $\epsilon, \delta \in (0, 1)$.

A note on the error:
`"ValueError:  Rank(A) < p or Rank([P; A; G]) < n"`

from cvxopt:

`cvxopt` expects the matrix H in a quadratic program to be positive definite, that is: to have only non-negative eigenvalues.

However, due to numerical inaccuracies in calculations, when some eigenvalues are very close to zero (though still positive), `cvxopt` might think they are negative (e.g., a tiny amount smaller than zero). You can check this by running `"numpy.linalg.eigvals(H)"` (make sure that H is a numpy array) and see the eigenvalues of the matrix H. If they are very close to zero, either positive or negative, this explains why you are getting this error.

To avoid this issue, if you have it, the solution is to add to the (main) diagonal of the matrix a **small** positive value, let's call it ϵ. If you add ϵ to the diagonal, all the eigenvalues grow by ϵ, so if one of the eigenvalues was too close to zero for python to work with, it will now be a little larger so that python is not confused thinking it's negative. However, adding anything to the matrix might change the result, and if you add a large number to the diagonal it might change the result too much. So the best strategy is to add the smallest value that works, so that on the one hand python doesn't get confused, and on the other hand the results don't change significantly.