Introduction to learning and analysis of big data
Exercise 2

Prof. Sivan Sabato

Fall 2022/3

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 via the course requests system.
- Grading: Q1,Q3,Q6,Q8: 10 points each. Q2,Q4,Q5,Q7,Q9: 12 point each.

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 include and 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 $n \in \{1, \ldots, 10\}$.

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. Implement the soft-margin kernel SVM routine described in class, using `cvxopt` quadratic problem solver you used in Question 1. The algorithm should use the polynomial kernel. You will implement the function `softsvmpoly` in the shell file "softsvmpoly.py" which is provided for this exercise in Moodle. function details:

```
def softsvmpoly(l, k, trainX, trainY)
```

The input parameters are:

- `l` - the parameter λ of the soft SVM algorithm.
- `k` - the degree of the polynomial kernel.
- `trainX` - a 2-D matrix of size $m \times d$. Row i in this matrix is a vector with d coordinates that describes 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 from the training sample. You can assume that each label is either -1 or 1.

The function returns the column vector `alpha` $\in \mathbb{R}^m$, which contains the coefficients found by the algorithm.

Question 4. For this question, use the data file `EX2q4_data.npz` provided on the course website, which contains data points in the domain $\mathcal{X} = \mathbb{R}^2$ and labels in $\{-1, 1\}$, split into a training set and test set.

(a) We would like to use soft SVM to learn a predictor for this problem. Plot the points in the training set in $\mathbb{R}^2$, and color them by their label. Explain why it may be a better idea use kernel soft SVM and not the linear (non-kernel) soft SVM.

(b) Run your Polynomial soft SVM code on the training set. Perform 5-fold cross-validation to tune λ and k. Try the values $\lambda \in \{1, 10, 100\}$ and $k \in \{2, 5, 8\}$ — a total of 9 parameter pairs to try. Report the 9 average validation error values for each of the pairs (λ, k). Report which pair was selected by the cross validation, rerun the training using this pair on the entire training set, and report the test error of the resulting classifier.

Repeat the procedure above using the linear (non-kernel) soft SVM code from Q1 on the given training set. In this case, you only need to choose λ since there is no k parameter.

(c) Which approach (polynomial kernel or linear soft SVM) achieved a better validation error? Is it what you expected? Why?

(d) For a general classification problem, give one reason why a polynomial SVM might get a better validation error than linear soft SVM, and one reason why it might get a worse validation error.

(e) Set $\lambda = 100$ and consider $k \in \{3, 5, 8\}$. For these values, run the polynomial soft SVM on the training set, and plot the resulting predictor in $\mathbb{R}^2$ as follows: Define a fixed region (roughly the region in which the training data resides), divide it into a fine grid, and color the grid points red or blue, depending on the label predicted by the classifier for each point. You can use the the function `matplotlib.pyplot.imshow` to plot this.

(f) (10pts) Choose the best value of λ for $k = 5$ that you found in the previous question. Take the output α that you got from this pair, and use it to calculate the predictor w, which is a vector in the new feature space. Answer the following questions:

i. What formula did you use to convert α to w?

ii. List the coordinates of the vector w.

iii. Write down the multivariate polynomial in x that is generated by the inner product $\langle w, \psi(x) \rangle$.

iv. Plot the prediction of this w as follows: Plot all the train and test points on a 2-dimensional plane, and color them according to the label that w predicts for each of them.

Question 5. Let $\mathcal{X}$ be the set of all undirected graphs over n vertices numbered $1, \ldots, n$ with degree at most 7. Let $\mathcal{Y} = \{0, 1\}$. For a graph $x \in \mathcal{X}$, define the mapping $g : \mathcal{X} \to \mathbb{N}^n$, where coordinate i in the vector $g(x)$ is the degree of vertex i in the graph x. Let $\mathcal{H} = \{h_v : \mathcal{X} \to \mathcal{Y} \mid v \in \mathbb{N}^n, h_v \not\equiv 0\}$, where $h_v(x) := \mathbb{I}[g(x) = v]$.

Suppose that $\mathcal{D}$ is a distribution over $\mathcal{X} \times \mathcal{Y}$, and suppose that in this distribution, the label of a graph x is a deterministic function of $g(x)$.

(a) What is the smallest dependence on ϵ that you can get in the sample complexity using the PAC bounds that we learned in class? Why?

(b) Use a bound on the **size** of the hypothesis class and the PAC-learning upper bound that we showed in class to show that the sample complexity of learning $\mathcal{H}$ as a function of n is $O(n)$.

(c) What is the VC dimension of $\mathcal{H}$? Use this value to get a better upper bound for the sample complexity of learning $\mathcal{H}$ as a function of n and ϵ.

Question 6. **Perceptron.** Let the example domain be $\mathcal{X} = \mathbb{R}^d$ and the label domain be $\mathcal{Y} = \{-1, +1\}$. Write a pseudo-code for an algorithm that accepts as input a labeled sample S of labeled examples from $\mathcal{X} \times \mathcal{Y}$, and outputs an upper bound on the number of updates that the Perceptron algorithm would require if it was run on this sample. Follow the following instructions:

- The algorithm must terminate and must be efficient. This means that it cannot simply run the Perceptron on the sample.
- You are allowed to use other algorithms that we studied in class as black boxes.
- If the Perceptron algorithm would not terminate on S, then the algorithm should output -1.

Question 7. **Quadratic program.** 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:

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

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 to so as to solve the minimization problem you wrote above.

Question 8. **The representer theorem.**

(a) Prove that the conditions of the representer theorem **do not hold** for the following optimization objective:

$$\text{Minimize}_{w \in \mathbb{R}^d} \lambda \|w\|_1 + \sum_{i=1}^{m} \langle w, x_i \rangle.$$

Note that $\|w\|_1 := \sum_{i=1}^{d} |w(i)|$.

Hint: The representer theorem requires a function R of the Euclidean norm, $\|w\|_2$. Show that a function of $\|w\|_2$ cannot output the correct $\|w\|_1$ for all $w \in \mathbb{R}^d$.

(b) Can anything be inferred about the minimization problem above from the fact that the representer theorem does not hold? Explain.

Question 9. **Kernel functions.** Consider a space of examples $\mathcal{X} = \mathbb{R}^d$. Let $x, x' \in \mathcal{X}$.

(a) Prove that the following function *cannot be* a kernel function for any feature mapping:

$$K(x, x') := (x(7) + x(3)) \cdot x'(1).$$

Hint: what property of inner products does this function violate? How can you prove it?

(b) Prove that the following function *cannot be* a kernel function for any feature mapping:

$$K(x, x') := 3 - (x(1) - x(2))(x'(1) - x'(2)).$$

Hint: consider the case $x = x'$.

(c) Show that the following function f is a Kernel function, by providing a possible feature mapping Ψ such that $f(x, x') = \langle \Psi(x), \Psi(x') \rangle$:

$$f(x, x') = (x(1)x'(1))^4 + e^{x(3)+x(5)+x'(3)+x'(5)} + 1/(x(1)x(1')).$$

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