Machine Learning Algorithms: Lab Exercises

Haolin CHEN, Stefano BAVARO, Irene TORRIJOS ROBLES

February 20, 2023

1 Goal of the lab exercise

The goal of this lab exercise is to provide us with hands-on experience in implementing and experimenting with different optimization algorithms for training a binary SVM classifier. Through the lab exercise, we will code three different algorithms: sub-gradient descent on the primal, projected gradient ascent on the dual, and box constrained coordinate ascent on the dual.

Each of these algorithms has different strengths and weaknesses, and by implementing and experimenting with them, we can gain a deeper understanding of how each works and when to use them. For example, sub-gradient descent on the primal is more efficient for large-scale problems, while projected gradient ascent on the dual is a popular choice for SVM training due to its efficiency and effectiveness.

We will use a toy dataset generated via scikit learn, which allows us to focus on the implementation and understanding of the optimization algorithms rather than on the specifics of the dataset. By playing around with the dataset and hyperparameters, we can gain a deeper understanding of how the algorithms work in practice and how to fine-tune them for optimal performance.

Additionally, we will need to implement a hyperparameter to weight the regularization term, which requires us to redo the math to check how this update changes in the primal and dual problem optimization algorithms. This will help us develop a deeper understanding of the regularization term and its impact on the performance of the SVM classifier.

Finally, the lab exercise requires us to use tensor operations exclusively in our code, which will help us develop good coding habits and improve the computational efficiency of our code. By doing this, we will learn how to use libraries like NumPy to vectorize operations and accelerate computation, which is particularly important for large-scale machine learning problems. The lab exercise also provides us with the opportunity to experiment with the dataset and hyperparameters, which can help us gain a deeper understanding of the impact of these factors on the performance of the SVM classifier. By varying the hyperparameters and observing the resulting changes in the classifier's performance, we can gain insights into how to optimize the SVM for different datasets and applications. This is an important skill for machine learning students, as the ability to fine-tune models for optimal performance is critical in real-world scenarios.

2 Algorithm explanations and implementations

2.1 Training algorithm 1: sub-gradient descent on the primal problem

Subgradient descent is an iterative optimization algorithm used to minimize a convex function (e.g SVM primal problem) that is not necessarily differentiable. The subgradient descent algorithm is similar to the gradient descent algorithm, but instead of using the gradient of the function to update the parameters, it uses a subgradient, which is a generalized concept of the gradient for non-differentiable functions.

The subgradient of a convex function at a point is a vector that points in the direction of the steepest descent of the function at that point. If the function is differentiable, the subgradient reduces to the gradient.

The subgradient descent algorithm starts with an initial guess for the optimal solution and then iteratively updates the solution using the subgradient of the function at the current point. The step size is chosen to ensure convergence to a minimum.

The main difference between gradient descent and subgradient descent is that gradient descent requires the function to be differentiable, while subgradient descent can handle non-differentiable functions. Gradient descent can converge faster than subgradient descent, but it may not work for functions that are not differentiable.

To implement this algorithm, we have to compute the subgradient of the objective function:

$$L(a, X, y, c) = \left(\sum_{i}^{n} max(0, 1 - \langle X_i, a \rangle y_i) \right) + \frac{c}{2} \|a\|_2^2$$

where:

- $\mathbf{X} \in \mathbb{R}^{n \times d}$: matrix where each row consists of a training point, i.e. $X_{i,j} = x_j^{(i)}$;
- $\mathbf{y} \in \mathbb{R}^n$: vector with the labels for each datapoint;
- $a \in \mathbb{R}^d$: the vector containing the primal variables;
- $c \in \mathbb{R}, c > 0$: the weight of the regularization term.

It results that

$$\frac{\partial}{\partial a_k} L(a, X, y, c) = \sum_{i \in I} -X_{i,k} y_i + \frac{c}{2} 2 a_k$$

where

$$I = \{i = 1, ..., n | 1 - \langle X_i, a \rangle y_i > 0\}$$

the set of indexes for which

$$max(0, 1 - \langle X_i, a \rangle y_i) > 0$$

The subgradient is coded using a boolean mask, which is a vector of N elements, where $N = number\ of\ datapoints$, which has all values as True and they are False for the indexes for which $1 - \langle X_i, a \rangle y_i > 0$. This mask is then used to select, whithin the matrix (dataset) X and the vector y (labels), only the elements in the indexes contained in I, so that to be able to retrieve the subgradient as:

$$g = -X[mask]^T y[mask] + ca$$

The update, with ϵ being the step-size, will then be:

$$a^{t+1} = a^t - \epsilon g$$

2.2 Training algorithm 2: projected gradient ascent on the dual problem

The projected gradient ascent algorithm is a method used to maximize a function subject to a constraint. It works by iteratively updating the input of the function in the direction of the gradient of the function, and then projecting the result onto the feasible region of the constraint.

Suppose you have a function that you want to maximize subject to some constraint. The projected gradient ascent algorithm behaves just as a standard gradient ascent, iteratively updating the values in the direction of the gradient of the function, which indeed tells on which direction to move in order to increase the value of the function.

However, this update step may result in a value that violates the constraint. For example, the input may end up being negative when the constraint requires it to be non-negative. To address this, the algorithm then "projects" the updated value onto the feasible region of the constraint, meaning it finds the closest point to the updated value that satisfies the constraint.

This process of updating and projecting is repeated until the input converges to a value that maximizes the function subject to the constraint. The projection step ensures that the final value satisfies the constraint, even if the gradient update step may have led the input to a value outside of the feasible region.

In order to implement the projected gradient ascent on the dual problem in this case, two tasks needed to be performed differently than what we saw during the course.

First, it was needed to retrieve the dual objective when an hyperparameter to control the weight of the regularization term is added. Here we show the steps that we performed and the result we got for this case.

In this case, the SVM training problem is:

$$\min_a \sum_{i=1}^{n} \ell([\mathbf{YX}a]_i) \;+\; \frac{\alpha}{2}\|a\|_2^2$$

where:

- $\mathbf{X} \in \mathbb{R}^{n\times d}$: matrix where each row consists of a training point, i.e. $X_{i,j} = x_j^{(i)}$;
- $\mathbf{Y} \in \mathbb{R}^{n\times n}$: matrix with the labels for each datapoint on the diagonal;
- $a \in \mathbb{R}^d$: the vector containing the primal variables
- $\ell(v_i) = \max(0, 1 - v_i)$

To get the SVM Fenchel dual objective, we need to derive both the Fenchel conjugate of the hinge loss (already done during the course) and of the quadratic regularization with a weight α. The latter is derived as follows:

- First let's prove that given

  $$f : R \to R \;\cup \{\infty\},\; f(u) = \alpha h(u), \alpha > 0,$$

  it results that

  $$f^*(t) = \alpha h^*\left(\frac{t}{\alpha}\right)$$

  PROOF:

  $$f^*(t) = \sup_{u\in dom(f)} \langle u, t\rangle - \alpha h(u) = \alpha$$

  $$\sup_{u\in dom(f)} \frac{\langle u, t\rangle}{\alpha} - \frac{\alpha}{\alpha}h(u) = \alpha$$

  $$\sup_{u\in dom(f)} \langle u, \frac{t}{\alpha}\rangle - h(u) = \alpha h^*(\frac{t}{\alpha})$$

- Then, let's compute the Fenchel conjugate of the quadratic regularization term

  $$h(\mathbf{a}) = \frac{1}{2}\|\mathbf{a}\|_2^2$$

  $$h^*(t) = \sup_{a\in dom(h)} \langle a, t\rangle - \frac{1}{2}\|\mathbf{a}\|_2^2$$

To find the maximizer $\hat{a}$, we proceed:

$$\nabla_a h^*(t) = t - a = 0 \Leftrightarrow \hat{a} = t$$

Then, we substitute $\hat{a}$ in $h^*(t)$ and we get:

$$h^*(t) = \langle t, t \rangle - \frac{1}{2}\|t\|_2^2 = \|t\|_2^2 - \frac{1}{2}\|t\|_2^2 = \frac{1}{2}\|t\|_2^2 = h(t)$$

From these two results, we finally get that the hinge loss of the quadratic regularization function with a weight α is:

$$h^*(\mathbf{a}) = \frac{\alpha}{2}\|\frac{\mathbf{a}}{\alpha}\|_2^2 = \frac{\alpha}{2\alpha^2}\|\mathbf{a}\|_2^2 = \frac{1}{2\alpha}\|\mathbf{a}\|_2^2$$

This given, we finally have that the SVM Fenchel dual objective is :

$$f(\lambda) = -\sum_{i=1}^{n} \lambda_i - \frac{1}{2\alpha}\lambda^T Y X X^T Y \lambda$$

where the dual variables λ are constrained in the interval [-1,0]. To perform project gradient ascent, we now have to derive the gradient of the objective function. We have that:

$$\nabla_\lambda f(\lambda) = -\mathbf{1} - \frac{1}{2\alpha}(Y^T X X^T Y \ + \ (Y^T X X^T Y)^T)\lambda = -\mathbf{1} - \frac{1}{\alpha} Y^T X X^T Y \lambda$$

where $\mathbf{1}$ is a vector of length n with only ones. Having obtained the gradient, we can finally implement the projected gradient ascent algorithm by defining the step in which the dual variables are updated, which is:

$$\lambda^{t+1} = proj_{[-1,0]^n}(\lambda^t + \epsilon \nabla_\lambda f(\lambda))$$

Finally, we also retrieve the formula to recover optimal primal variables from dual variables. From the KKT's stationary conditions, for optimal primal and dual variables we have:

$$\nabla_a L(a, v, \lambda) = 0 \Leftrightarrow \nabla_a \left(\sum_{i=1}^{n} max(0, 1 - v_i) + \frac{\alpha}{2}\|a\|_2^2 \ + \ \lambda^T(Y X a - v) \right) = 0$$

$$\Leftrightarrow \alpha a \ + \ (\lambda^T Y X)^T \Leftrightarrow a = -\frac{(\lambda^T Y X)^T}{\alpha}$$

2.3 Training algorithm 3: box constrained coordinate ascent on the dual problem

The box constrained coordinate ascent algorithm is an iterative optimization algorithm used to solve optimization problems with box constraints on the decision variables. It is a variation of the coordinate ascent algorithm, where each coordinate update is subject to box constraints.

The algorithm starts with an initial guess for the optimal solution and iteratively updates the decision variables one at a time, while keeping the other variables fixed. The update is subject to the box constraints on the variable, which restrict the range of values it can take.

The box constraints on the decision variables can be expressed as a set of upper and lower bounds for each variable, and the updates must ensure that the new value of the decision variable remains within these bounds. The algorithm stops when a convergence criterion is met, such as a maximum number of iterations or a small enough change in the objective function.

To implement this algorithm, as the objective function is the same of the one used in the second algorithm, the only thing left to do is to compute the partial derivative of the objective function.

To do it, let's first rewrite the objective funtion (setting $Q = YXX^TY$) as:

$$f(\lambda) = -\frac{1}{2\alpha}\lambda^T Q\lambda - \sum_{i=1}^{n}\lambda_i = -\frac{1}{2\alpha}\sum_{i\neq j}\lambda_i\lambda_j Q_{j,i} - \frac{1}{2\alpha}\sum_{i}^{n}\lambda_i^2 Q_{i,i} - \sum_{i=1}^{n}\lambda_i$$

From this formulation it is easier to compute the partial derivative, which is:

$$\frac{\partial}{\partial\lambda_k}f(\lambda) = -\frac{1}{2\alpha}2\sum_{i\neq k}^{n}\lambda_i Q_{k,i} - \frac{1}{2a}2\lambda_k Q_{k,k} - 1 = -\frac{1}{\alpha}\sum_{i\neq k}^{n}\lambda_i Q_{k,i} - \frac{1}{\alpha}\lambda_k Q_{k,k} - 1$$

By solving for the derivative equal to zero, we have:

$$-\frac{1}{\alpha}\sum_{i\neq k}^{n}\lambda_i Q_{k,i} - \frac{1}{\alpha}\lambda_k Q_{k,k} - 1 = 0 \iff \lambda_k = -\frac{\alpha(1 + \frac{1}{\alpha}\sum_{i\neq k}^{n}\lambda_i Q_{k,i}}{Q_{k,k}}$$

Since the dual variables λ are constrained in $[-1, 0]$, the update of each of the coordinates is clipped in that interval.

To implement this procedure in code, and more specifically to code the $\sum_{i\neq k}^{n}\lambda_i Q_{k,i}$ term of the updated value of λ_k, we used a mask having all True values and a False value in position k, so that to remove the element in position k both from the λ $(dual_v ars)$ vector and from the k-th row of Q. The function to retrieve the primal variables from the dual variables is the same as described in algorithm 2.

3 Experiment results

Executing the algorithms described in the previous sections, we got some expected results. Indeed, the accuracy of the prediction, for each of the algorithm and considering different values of the regularization weight (either 1, 3 or 5), revealed to be high, always almost 1 (depending on how the data are randomly generated). One interesting aspect to notice is the shape of the loss function over the different epochs: indeed, for the first algorithm the values taken by the objective function are decreasing as, in fact, the weights are updated following the opposite direction of the gradient (direction of steepest reduction of the loss); for the other two algorithms, instead, the the values taken by the objective function are increasing, as we are performing two algorithms for which the direction of the gradient (steepest increase) is followed for the updates. Figure 1 show the shapes of the objective function for the three algorithms.

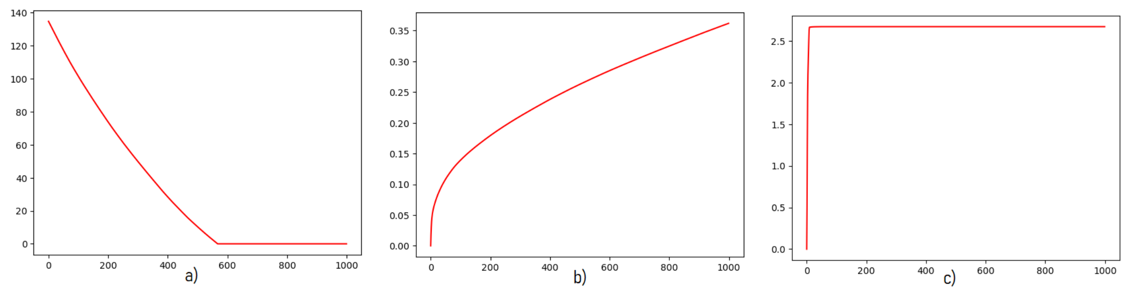
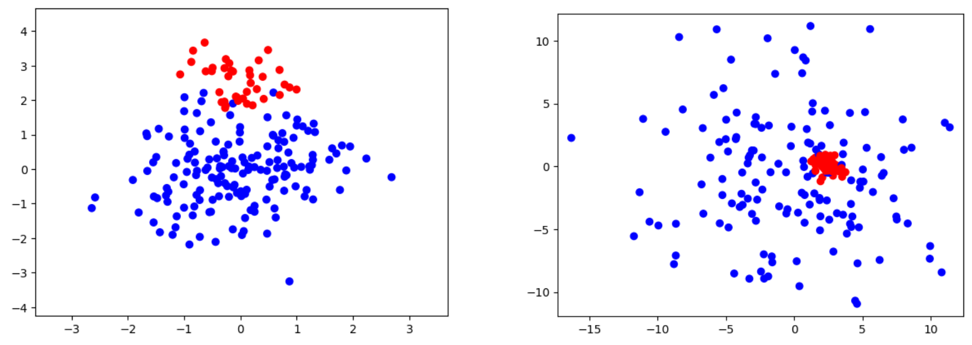


Figure 1: Objective functions of the three algorithms impemented (a. sub-gradient descent; b. projected gradient ascent; c. box constrained coordinate ascent)

We tried to execute the same algorithms on other two datasets (depicted in Figure 2): as the data were less separable than the ones given initially, the accuracy scores are smaller (especially in the last dataset used), but the characteristics noticed in the objective functions reappeared, as expected.

Another last observation that can be made is that the box constrained coordinate ascent algorithm on the dual problem takes more time to train: this observation is expected as, in this case, each coordinate



(a) (b) label 2

Figure 2: Additional datasets used to test the algorithms

is updated individually with a for loop, less efficiently than tensor operations, eventually increasing the time complexity of the algorithm.