Machine Learning Algorithms: Lab Exercises

Racha Hachem Jacobo Ruiz Ocampo Alexis Delplace

Goal of the lab exercise

In this short report, we will explain the steps of our implementation of the theoretical foundations of the model.

First step

After testing the preliminary code and generating the data, we move to the first step to write the "predict" and the "accuracy" functions.
In this part, we should write the predict function for the binary SVM classifier: We have the set of input values X, and the model a, we should return a vector y of size X.shape[0], which will contain the predicted labels for each of the rows in X. The formula of the prediction for SVM binary classification is the following:

$$\hat{y} = sign(\mathbf{a} \circ \mathbf{x}^T)$$

As for the accuracy, we can simply get it from getting the average of the correctly predicted labels (we do that by checking if each of the predicted label is equal to the correct label)

$$accuracy(gold, predicted) = \frac{\sum_{i=1}^{|gold|} [gold_i = predicted_i]}{|gold|}$$

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

In this part, we want to implement the first algorithm to train the binary SVM classifier using the sub-gradient descent on the primal problem. To do that, we look first at the objective function:

$$\sum_i \max(0, 1 - X_i a y_i) + \frac{c}{2}||a||_2^2$$

To be able to write it with python code, we need to define a variable for the right side of max function:

$$right = 1 - X_i a y_i$$

Then we create a mask such that, if the right is greater than 0, the mask value will be 1 and 0 otherwise:

$$mask = \begin{cases} 1, & if\ right > 0 \\ 0, & otherwise \end{cases}$$

That way, we can calculate the max, by multiplying the mask term by the right term, and the objective function would look like the following:

$$\sum_{i=1}^{n} [right_i \cdot mask_i] + \frac{c}{2}||a||_2^2$$

Based on the simplification that we did on the objective function and the max, we can now easily compute the sub-gradient of the objective function inrespect to a by applying the following logic:

1. Starting off by the first term (the max), we can right something like that:

$$\frac{\partial}{\partial a} \max(0, right) = \begin{cases} 0, & right < 0 \\ -X_i y_i, & otherwise \end{cases}$$
$$= -\text{mask}_i X_i y_i$$

2. Then we compute the sub-gradient of the regularization term which is simply:

$$\frac{\partial}{\partial a} \frac{c}{2} |a|_2^2 = c \cdot a$$

3. Combining the 2 terms together, we get the final answer:

$$\frac{\partial}{\partial a} \left(\max(0, 1 - Xiay_i) + \frac{c}{2}|a|_2^2 \right) = \sum_i -\text{mask}_i X_i y_i + c \cdot a$$

Finally to update the update step, we simply subtract from it the product of the step-size with the sub-gradient

$$a \leftarrow a - step\_size \cdot sub\_grad$$

Training algorithm 2: sub-gradient descent on the dual problem

1. Primal problem of the SVM with a regularization weight

Primal SVM objective function:

$$\sum_i \max(0, 1 - X_i a y_i) + \frac{c}{2}\|a\|_2^2$$

Now we introduce a change in notation and end up with :

$$\sum_i l([YXa]_i) + \frac{c}{2}\|a\|_2^2 \tag{1}$$

2. Computing the fenchel dual problem of the SVM with a regularization weight

Let:

$$f : R^m \to R, h : R^n \to R$$

$$U \in R^{m*n}, v \in R^n, \lambda \in R^m$$

Then, the Fenchel dual problem allows us to write the following inequality:

$$\min_v f(Uv) + h(v) > \max_\lambda -f^*(\lambda) - h^*(-U^T\lambda) \tag{2}$$

Therefore for the SVM problem in equation (1), we have:

$$f(x) = \sum_i l([x]_i) \tag{3}$$

$$h(x) = \frac{c}{2}\|x\|_2^2 \tag{4}$$

$$U = YX, v = a$$

Hence we can rewrite (2) as the inequality between the dual and primal problem of the SVM as follows :

$$\min_v f(YXa) + h(a) > \max_\lambda -f^*(\lambda) - h^*(-(YX)^T\lambda)$$

In comparison to the SVM problem without regularization, the only function affected by the regularization term is $h(x)$. Hence we have the unchanged conjugate for $f(x)$:

$$f^*(x) = x + \delta_{[-1,0]}(x) \tag{5}$$

Now let's compute $h^*(x)$:

In the lectures we proved that for any two functions h and g:

If $h(x) = \alpha g(x)$ then $h^*(x) = \alpha g^*(\frac{x}{\alpha})$

Here, for $g(x) = \frac{1}{2}\|x\|_2^2$ and $\alpha = c$, then (4) can be written as :

$$h(x) = \alpha g(x)$$

Then we have that:

$$h^*(x) = \alpha g^*(\frac{x}{\alpha})$$

$$h^*(x) = \frac{c}{2}\left\|\frac{x}{c}\right\|_2^2$$

$$h^*(x) = \frac{c}{2}\left\|\frac{x}{c}\right\|_2^2$$

We know that for any scalar k and a vector a:

$$\|ka\| = |k|\,\|a\|$$

Therefore:

$$h^*(x) = \frac{c}{2}\left\|\frac{1}{c}\right\|_2^2 \|x\|_2^2$$

$$h^*(x) = \frac{c}{2}\frac{1}{c^2}\|x\|_2^2$$

Finally we have:

$$h^*(x) = \frac{1}{2c}\|x\|_2^2 \tag{6}$$

Now from equation (5) and (6), we can write the fenchel dual problem for the SVM:

$$\max_\lambda -f^*(\lambda) - h^*(-(YX)^T\lambda)$$

From now on we have the following constraints because of (5) : $\quad -1 \le \lambda_i \le 0, \qquad 1 \le i \le n$

$$\max_\lambda -\sum_i \lambda_i - \frac{1}{2c}\left\|-(YX)^T\lambda\right\|_2^2$$

$$\max_\lambda -\sum_i \lambda_i - \frac{1}{2c}(-(YX)^T\lambda)^T(-(YX)^T\lambda)$$

$$\max_\lambda -\sum_i \lambda_i - \frac{1}{2c}(\lambda^T YX)((YX)^T\lambda)$$

$$\max_\lambda -\sum_i \lambda_i - \frac{1}{2c}\lambda^T YXX^T Y^T\lambda$$

Because the Y matrix is diagonal, we can write the final form of the dual problem of the SVM as follows:

$$\max_\lambda -\sum_i \lambda_i - \frac{1}{2c}\lambda^T YXX^T Y\lambda \tag{7}$$

$$s.t \quad -1 \le \lambda_i \le 0, \quad 1 \le i \le n$$

3. Retrieving primal variables from dual variables

From the KKT's stationary conditions for optimal primal and dual variables we have:

$$\nabla_a L(a, v, \lambda) = 0$$

$$\nabla_a(\sum_i \max(0, 1 - v_i) + \frac{c}{2}\|a\|_2^2 + \lambda^T(YXa - v)) = 0$$

$$ca + (\lambda^T YX)^T = 0$$

$$ca = -X^T Y^T\lambda$$

Finally we have:

$$a = -\frac{X^T Y^T\lambda}{c} \tag{8}$$

4. Justifying the use of the projected gradient ascent algorithm

In this section we will justify why the projected gradient ascent algorithm can be used for solving the fenchel dual SVM maximization problem.

The problem in equation (7) can be generalized to the following form:

$$\max_{u \in S} f(u) \tag{9}$$

For $-f$ a convex, differentiable function and S a convex set.

Justification:

- We have that the fenchel dual is a concave function, as it is composed of the addition of the negation of two fenchel conjugates, see equation (2).
- Let's now prove that the set $C = [-1, 0]$ is convex. C is convex if:

  $$\forall x, y \in C : \quad \epsilon x + (1-\epsilon)y \in C$$

  Let $x, y \in C$ such that: $-1 \leq x \leq y \leq 0$
  Therefore we have:

  $$-1 \leq \epsilon x + (1-\epsilon)x \leq \epsilon x + (1-\epsilon)y \leq \epsilon y + (1-\epsilon)y \leq 0$$

  $$-1 \leq \epsilon x + x - \epsilon x \leq \epsilon x + (1-\epsilon)y \leq \epsilon y + y - \epsilon y \leq 0$$

  $$-1 \leq x \leq \epsilon x + (1-\epsilon)y \leq y \leq 0$$

  $$-1 \leq \epsilon x + (1-\epsilon)y \leq 0$$

  Therefore:

  $$\forall x, y \in C, \epsilon x + (1-\epsilon)y \in C$$

We know that a problem written like the one in the equation (9) can be solved using the projected gradient ascent algorithm. Therefore this algorithm can be used to solve the problem described in (7).

5. The projected gradient ascent on the dual problem of the SVM

This algorithm is adapted to optimization problems where the parameter is contrained to a convex set. This is the only particularity, therefore it behaves exactly as the gradient descent except that it clips each computed updated weight to force it to belong to the convex set.

This is the update step of the algorithm that optimizes the problem defined in (9) :

$$u^{(t+1)} = proj_S(u^{(t)} + \epsilon^{(t)} \nabla f(u^{(t)}))$$

Therefore we need the projection function $proj_{[0,1]}$ and the gradient of f.

- In class we defined $proj_{[-1,0]}$ this as follows:

  $$proj_{[-1,0]}(w) = Clip_{[-1,0]}(w) = \begin{cases} -1 & : w \leq -1 \\ 0 & : w \geq 0 \\ w & : -1 < w < 0 \end{cases}$$

- Let's compute the gradient for λ at step t (usually noted $\lambda^{(t)}$, but noted just λ here for readability):

  $$\nabla f(\lambda) = \nabla_\lambda(-\sum_i \lambda_i - \frac{1}{2c}\lambda^T Y X X^T Y \lambda)$$

  $$= -1 - \frac{1}{c}\nabla_\lambda(\frac{1}{2}\lambda^T Y X X^T Y \lambda)$$

  As we saw in class:

  $$\nabla_\lambda(\frac{1}{2}\lambda^T Y X X^T Y \lambda) = Y^T X X^T Y \lambda$$

  Therefore:

  $$\nabla f(\lambda^{(t)}) = -1 - \frac{1}{c} Y^T X X^T Y \lambda^{(t)} \tag{10}$$

We can finally write the update step of the projected gradient ascent for the SVM dual problem as follows:

$$\lambda^{(t+1)} = Clip_{[-1,0]}(\lambda^{(t)} + \epsilon^{(t)}(-1 - \frac{1}{c} Y^T X X^T Y \lambda^{(t)})) \tag{11}$$

6. Comments on the implementation

To implement this algorithm we had to implement:

- (8) $a = -\frac{X^T Y^T \lambda}{c}$:

  We use the primal variables extracted from dual variables to intialize our weights. The implementation is very straight forward, we compute the matrix multiplications first and then we divide the resulting scalar by c.

- (7) the objective function of the dual:
  Same comment as before, except that here we will be dividing by 2c.

- (10), gradient of the objective:
  Same comment as for the primal variables.

- (11) Update step: Same comment as for the primal variables. Here a particularity is that we need to use np.clip to clip the values after they are calculated.

Training algorithm 3: box constrained coordinate o on the primal problem

1. Computing the box constrained quadratic problem

We have from equation (7) that the dual problem is:

$$\max_\lambda -\sum_i \lambda_i - \frac{1}{2c}\lambda^T Y X X^T Y \lambda$$

$$s.t \quad -1 \leq \lambda_i \leq 0, \quad 0 \leq i \leq n$$

For $Q \in R^{n*n}$, $b \in R^n$, we know that the box constrained quadratic problem can be defined as:

$$\max_\lambda \frac{1}{2}\lambda^T Q \lambda + b^T \lambda \tag{12}$$

$$s.t \quad I \leq \lambda \leq u$$

For the SVM dual problem we have:

- $Q = -\frac{1}{c} Y X X^T$
- $b = -1$
- $I = -1$
- $u = 0$

2. Justifying the use of the coordinate ascent algorithm

As we have proven in the lectures, the box constrained quadratic problem:

- has a non smooth objective that could be written as:

$$\max_{u \in R^d} f(u) + h(u)$$

with $f : R^d \to R \cup (\infty)$, $h : R^d \to R \cup (\infty)$ and $h(u)$ additively separable.

- Because of the previous properties each coordinate of the vector λ can be optimized independently of the other coordinates. Therefore this problem can be solved via the coordinate ascent algorithm.

3. Coordinate ascent algorithm

The main idea of this algorithm is to iteratively solve the problem for one single coordinate at a time.
We want to solve at a given iteration k, $\frac{\partial f}{\partial \lambda_k} f(\lambda) = 0$.
If the solution found does not satisfy the constraints then we clip it.
It is important that when we solve for one k coordinate, we immediately use the solution found for the update step of the next coordinate.

From the lectures we know that for the problem (12) we have computed that the update step for coordinate k of lambda is:

$$\lambda_k = Clip_{[I,u]}\left(\frac{-b_k - \sum_{i \neq k} \lambda_i Q_{k,i}}{Q_{k,k}}\right) \qquad (13)$$

4. Implementation

To implement this algorithm we initialized our variables as follows:

- $Q = -\frac{1}{c} Y X X^T$
- $b = -1$
- $I = -1$
- $u = 0$

We defined the objective function as (12).

Finally, we updated one coordinate at a time using (13) while immediatly using the previously computed solution by updating λ. We used a boolean mask on the sum in order to comply with the constrains . We chose to update the coordinates in order.

Experiment results: hyperparameters, learning curves, analysis, model/dataset comparison

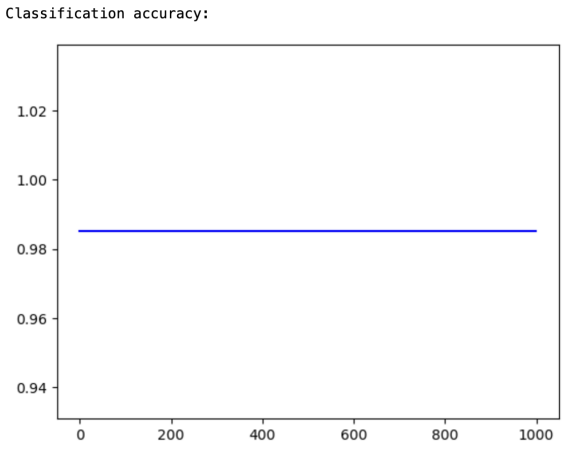

Figure 1: Accuracy of the 2^{nd} algorithm

The classification accuracy seems a bit odd since we have almost an accuracy of 1.0 and this result is obtained at the first epoch. In order to ensure that our algoritm is working properly, we tested it on a larger dataset which was less separable.

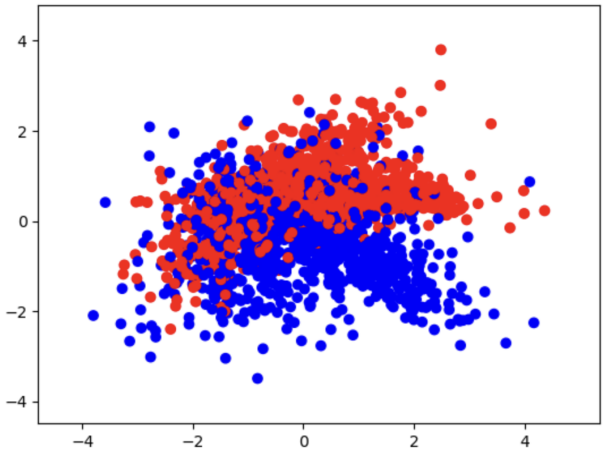

Figure 2: The larger dataset, 2000 points

On this new dataset, the obtained accuracy curve has a shape that is way more usal than the previous one, the algorithm seems to be working well.

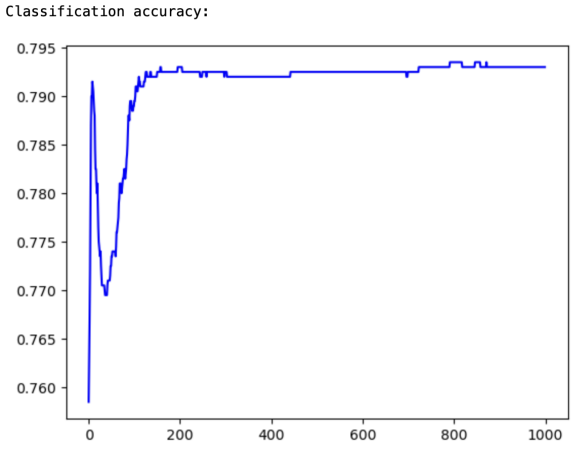

Figure 3: Accuracy of the 2^{nd} algorithm on the larger dataset