Machine Learning

Homework #(**3**)
JARRY Guillaume

---

1 Q1

1) Here the loss function is a sum of function. Thus the subgradient of the sum will be the sum (in the sense of the sum of two set) of the subgradient for each function.

$$\partial l(w) = \{\lambda w\} + \frac{1}{n}\sum_{i=1}^{n} \begin{Bmatrix} \{0\} \text{ if } 1 - y_i w^T x_i < 0 \\ \{-y_i x_i\} \text{ if } 1 - y_i w^T x_i > 0 \\ \{\} \text{ if } if 1 - y_i w^T x_i = 0 \end{Bmatrix}$$

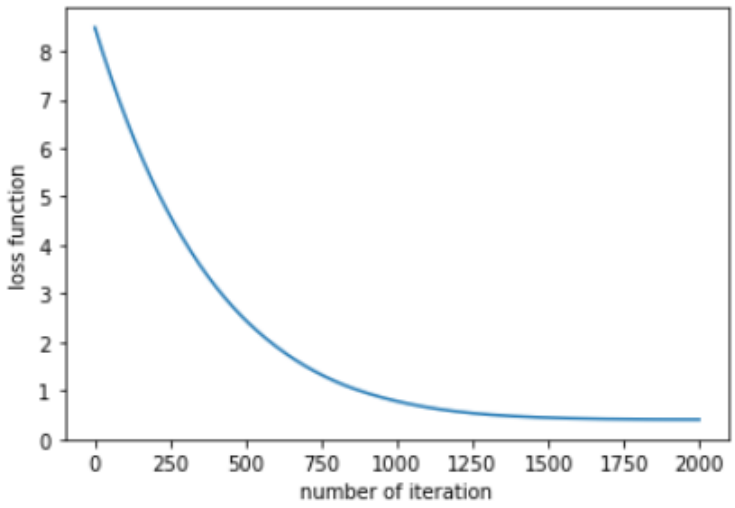
2) The following code produces the following plots.I took the liberty to increase the number of iteration as the algorithm was not converging for a number of iteraton of 100. Instead and thus we can observe the curves below.

```
lbda = 0.1
n = 1000
d = 100
np.random.seed(1337)
X = np.vstack([np.random.normal(0.1, 1, (n//2, d)),
np.random.normal(-0.1, 1, (n//2, d))])
y = np.hstack([np.ones(n//2), -1.*np.ones(n//2)])
w0 = np.random.normal(0, 1, d)
```

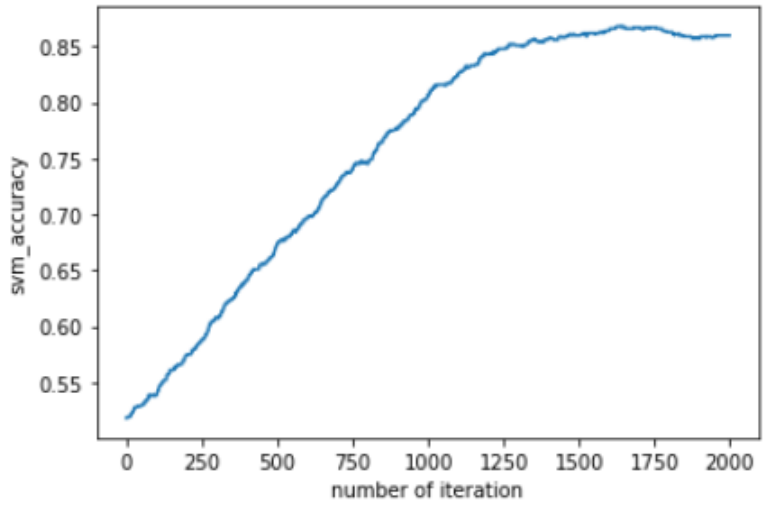
```
def f(w):
    return 1/n * sum([max(0, 1 - y[i]*w.dot(X[i])) for i in range(len(X))]) + lbda/2 * np.linalg.norm(w)**2

def grad_f(w):
    return lbda * w + 1/n * sum([-y[i]*X[i] if 1-y[i]*w.dot(X[i]) > 0 else 0 for i in range(n)])

def accuracy(w):
    accuracy = 0
    for i in range(n):
        y_pred = np.sign(w.dot(X[i]))
        if y_pred == y[i]:
            accuracy += 1

    return accuracy/n

def gradient_descent(f, grad_f, w, step_size=0.01, n_iter=1500):
    loss_values = [f(w0)]
    acc = []
    for _ in range(n_iter):
        w = w - step_size * grad_f(w)
        loss_values.append(f(w))
        acc.append(accuracy(w))

    return w, loss_values, acc
```

```
n_iter = [i for i in range(2000)]
w, loss_values, svm_accuracy = gradient_descent(f, grad_f, w0, n_iter = len(n_iter)-1)
```

```
1  plt.plot(n_iter, loss_values)
2  plt.xlabel("number of iteration")
3  plt.ylabel("loss function")
4  plt.show()
```




```
1  plt.plot(n_iter[:len(n_iter)-1], svm_accuracy)
2  plt.xlabel("number of iteration")
3  plt.ylabel("svm_accuracy")
4  plt.show()
```



Machine Learning

Homework #(3)
JARRY Guillaume

---

2 Q2

1) Using the equality constraint, we get:

$$\sum_{=1}^{n} y_i \alpha_i = 0$$

And thus we can eliminate variable α_1:

$$\alpha_1 = -\frac{1}{y_1} \sum_{i=2}^{n} \alpha_i y_i$$

In the equation (2), we then eliminate all the temrs of the sum independant of α_1, α_2 as those are constant and do not intervene in the maximization problem. What is left is then, by simetry of the kernell:

$$l(\alpha_1, \alpha_2) = -\sum_{i=1}^{2}\sum_{j=3}^{n} y_i y_j k(x_i, x_j)\alpha_i \alpha_j + \alpha_1 + \alpha_2 - \frac{1}{2}\sum_{i=1}^{2}\sum_{j=1}^{2} y_i y_j k(x_i, x_j)\alpha_i \alpha_j$$

The intermediate line is (using $y_i^2 = 1$ for all i):

$$l(\alpha_1, \alpha_2) = \alpha_1 + \alpha_2 - \left(\frac{\alpha_2^2}{2} k(x_2, x_2) + \frac{\alpha_1^2}{2} k(x_1, x_1) + y_2 \alpha_2 \sum_{i=3}^{n} y_i k(x_2, x_i) + y_1 \alpha_1 \sum_{i=3}^{n} y_i k(x_1, x_i)\alpha_i + y_1 y_2 \alpha_1 \alpha_2 k(x_1, x_2) \right)$$

And then, by injecting the constraint into this equation, we get (again discarding all the unecessary constant irrelevant to the maximization problem):

$$l(\alpha_2) = -\frac{\alpha_2^2}{2}\left(k(x_1, x_1) + k(x_2, x_2) - 2k(x_1, x_2)\right) - \alpha_2 \left(\frac{y_2}{y_1} - 1 - y_2 \sum_{i=3}^{n} y_i \alpha_i \left[k(x_2, x_i) - k(x_1, x_i) - k(x_1, x_2) + k(x_1, x_1)\right] \right)$$

Here we recognize a polynomial $l(\alpha_2) = \frac{\eta}{2}\alpha_2^2 + \gamma\alpha_2$, a result which will be important later in the optimization problem.

2) Using the inequality constraints this time, we can derive the value of L and U. We have both constraints:

$$0 \leq \alpha_2 \leq C \text{ and } 0 \leq -\frac{1}{y_1}\sum_{i=3}^{n} \alpha_i y_i - \alpha_2 \frac{y_2}{y_1} \leq C$$

Thus, we can summarize both into a single line.

$$max\left\{-\frac{Cy_1}{y_2} - \frac{1}{y_2}\sum_{i=3}^{n}\alpha_i y_i, 0\right\} \leq \alpha 2 \leq max\left\{-\frac{1}{y_2}\sum_{i=3}^{n}\alpha_i y_i, C\right\}$$

And then we can assign:

$$L = max\left\{-\frac{Cy_1}{y_2} - \frac{1}{y_2}\sum_{i=3}^{n}\alpha_i y_i, 0\right\} \text{ and } U = max\left\{-\frac{1}{y_2}\sum_{i=3}^{n}\alpha_i y_i, C\right\}$$

---

3) Calculating the second order derivative we find $\eta = 2k(x_1, x_2) - k(x_1, x_1) - k(x_2, x_2)$, and then we realize that the problem above is that of maximizing a decreasing (because $\eta < 0$ parabola on an interval. Thus the problem will have always have a unique solution on the interval.

By putting the polynomial $l(\alpha_2)$ under canonical form, $\frac{\eta}{2}\left(\alpha_2 + \frac{\gamma}{\eta}\right)^2 - \frac{\gamma^2}{2\eta}$ we find the optimal α_2^* on R to be:

$$\alpha_2^* = -\frac{\gamma}{\eta}$$

Where $\gamma = -\left(\frac{y_2}{y_1} - 1 + y_2 \sum_{i=3}^{n} y_i \alpha_i [k(x_1, x_i) + k(x_2, x_i) - k(x_1, x_2)]\right)$. Let us simplify γ using E_1 and E_2. Because in the algorithm, the value of E_1 and E_2 have not been computed yet, they store the previous value $\alpha_1^{(t)}$ and $\alpha_2^{(t)}$. Thus we get:

$$E_2 - E_1 = -y_2 \eta \alpha_2^{(t)} + \sum_{i=3}^{n} \alpha_i y_i \left[k(x_2, x_i) - k(x_1, x_i) - k(x_2, x_1) + k(x_1, x_1)\right] - y_2 + y_1)$$

$$\gamma = -\left(\frac{y_2}{y_1} - 1 + y_2(E_2 - E_1 + \eta \alpha_2^{(t)} - y_1 + y_2\right) = -y_2(E_2 - E_1 + \eta \alpha_2^{(t)})$$

But, we need need to check the constaint: $L \leq \alpha_2 \leq U$. Thus, the correct expression of α_2^* will be:

$$\alpha_2^* = \begin{cases} \alpha_2^{(t)} + y_2 \frac{(E_2 - E_1)}{\eta} \text{ if } L \leq -\frac{\gamma}{\eta} \leq U \\ L \text{ if } -\frac{\gamma}{\eta} \leq L \\ U \text{ if } U \leq -\frac{4\gamma}{\eta} \end{cases}$$

Then, simply using the expression for α_1 found in question 1), we get:

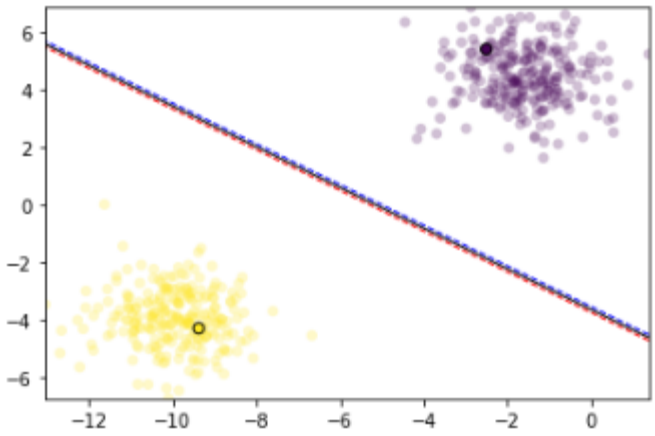
$$\alpha_1^* = -\frac{y_2}{y_1}\alpha_2^* - \frac{1}{y_1}\sum_{i=3}^{n} \alpha_i y_i$$

Which can be simplified into:

$$\alpha_1^* = \alpha_1^{(t)} + \frac{y_2}{y_1}(\alpha_2^{(t)} - \alpha_2^*)$$

4) You can see the implemented code in the q2 file.

5) I only had time to run the first experiment, as the second one was much more computationaly heavy. I apologize for not plotting the title of the graph in the python code.



Decision bounndary for experience 1.

Here, the red doted line and the blue doted line

3 Q3

1) Because of the whitening, let us remark that :

$$\forall (i,j) \in \{1,\dots,n\}^2, \quad X_{.i}^T X_{.j} = \delta_{ij}$$

Where δ_{ij} is the kronecker symbol. Thus we can simplify the expression:

$$J_\lambda(\beta) = \frac{1}{2}(y - \sum_{i=1}^{n} \beta_i X_i)^T (y - \sum_{i=1}^{n} \beta_i X_i) + \lambda \sum_{i=1}^{n} |\beta_i|$$

Which gives:

$$J_\lambda(\beta) = \|y\|^2 + \sum_{i=1}^{n} \left(\beta_i y^T X_{.i} + \lambda|\beta_i| + \beta_i^2\right)$$

And thus we can pose:

$$f(X_{.i}, \beta_i, \lambda, y) = \beta_i y^T X_{.i} + \lambda|\beta_i| + \beta_i^2$$

2) Then, we reognize that $J_\lambda(\beta)_i$ is (almost) polynomial in β_i. We can turn the minimization problem into that of minimizing the polynomial by specifying the sign of β_i.

If $\beta_i^* > 0$, then the polynomial to minimize is $\beta_i^2 + \beta_i(\lambda + y^T X_{.i}$, and the solution will be:

$$\beta_i^* = -\frac{(\lambda + y^T X_{.i})}{2}$$

3) If $\beta_i^* < 0$, then the polynomial to minimize is $\beta_i^2 + \beta_i(-\lambda + y^T X_{.i}$, and the solution will be:

$$\beta_i^* = -\frac{(y^T X_{.i} - \lambda)}{2}$$

---

4) From the result above, $\beta_i^* = 0$ will happen if $\lambda = y^T X_{.i}$ or $\lambda = -y^T X_{.i}$. We can interpret this condition as the column j of the data as the regularizer being equal to the scalar product of the output and a column of the input matrix.

5) In that case, then the expression of our polynomial simply becomes $\beta_j^2(1 + \frac{\lambda}{2}) + y^T X_{.j}\beta_{.i}$. And the minimizing condition becomes:

$$\beta_j^* = \frac{4y^T X_{.j}}{(2 + \lambda)}$$

Thus, here, β_j^* will be null if $y^T X_{.j} = 0$, ie, the output value is othogonal to a column in the matrix (the vector formed by the coordinate of all the training point). This would mean that this coordinate indicate the orientation of the output and that it is irrelevant.

β_j^* is more likely to be zero in the L_1 case because it is very rare that one column of the training data is of no use in the training.