Machine Learning — Prof. Dr. Bernt Schiele

Dr. Paul Swoboda & Dr. Gerard Pons-Moll

Exercise Sheet 5 - 19.11.2018 — due: 26.11.2018, 14:15

Exercise 13 - Dual Coordinate Ascent for Linear SVM

In this exercise we implement coordinate ascent method for solving the dual of the soft-margin SVM problem. For simplicity, we restrict to the case where the offset b is fixed at zero. The dual problem in this case is given by

$$\max_{\alpha \in \mathbb{R}^n} \ \Psi(\alpha) \tag{1}$$
$$\text{subject to: } 0 \leq \alpha_i \leq \frac{C}{n}, \quad \forall i = 1, \ldots, n$$

where $\Psi(\alpha) = \sum\limits_{i=1}^{n} \alpha_i - \frac{1}{2} \sum\limits_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle$ and $x_i \in \mathbb{R}^d$ is the i^{th} training example and $y_i \in \{-1, +1\}$ is its label.

a. (**Bonus 2 Points**) Derive the dual formulation given above for the soft-margin SVM problem when the offset $b = 0$.

b. Coordinate descent/ascent is an optimization method which can be applied when the objective is differentiable and the optimization variables are not coupled by the constraints. This method solves a sequence of smaller subproblems where the objective is optimized over a single coordinate while fixing the values for the other coordinates. The coordinates for the subproblems are typically chosen in a cyclic order.

The subproblem for the dual (1) over the coordinate r is given by

$$\alpha_r^{\text{new}} = \arg\max_{\alpha_r \in \mathbb{R}} \quad \Psi\Big(\alpha_1^{\text{old}}, \ldots, \alpha_{r-1}^{\text{old}}, \alpha_r, \alpha_{r+1}^{\text{old}} \ldots, \alpha_n^{\text{old}}\Big)$$
$$\text{subject to: } 0 \leq \alpha_r \leq \frac{C}{n}$$

where all the variables except for α_r are fixed at previous values. Note that this is a quadratic problem in one variable α_r with an interval constraint: $\alpha_r \in [0, \frac{C}{n}]$.

1. (**3 Points**) First derive the unconstrained solution $\bar{\alpha}_r$ of the above subproblem (i.e., ignoring the interval constraint). Next show that the solution in the presence of constraints is simply the projection of $\bar{\alpha}_r$ onto the interval $[0, \frac{C}{n}]$. That is

$$\alpha_r^{\text{new}} = \max\Big\{0, \min\Big\{\bar{\alpha}_r, \frac{C}{n}\Big\}\Big\}$$

2. (**1 Point**) Show that the KKT conditions given in the lecture can be rewritten as the following:

$$\forall i = 1, \ldots, n:$$
$$\alpha_i = 0 \Rightarrow y_i \langle w, x_i \rangle \geq 1, \quad 0 < \alpha_i < \frac{C}{n} \Rightarrow y_i \langle w, x_i \rangle = 1, \quad \alpha_i = \frac{C}{n} \Rightarrow y_i \langle w, x_i \rangle \leq 1$$

c. (**3 Points**) Complete the given Matlab function `CoordinateDescentSVM.m` which takes as arguments the training data `Xtrain` $\in \mathbb{R}^{n \times d}$, the class labels `ytrain` $\in \{-1, +1\}^n$, the error parameter C as well as the test data `Xtest`, `ytest` and returns the dual solution α, the primal solution w and training and test errors `TrainErrs`, `TestErrs` computed at each step of the coordinate descent method. Note that test data is only used to compute the test errors at intermediate steps! You have to fill-in the following blocks in the code labelled as `Fill-in`:

1. Compute the unconstrained solution $\bar{\alpha}_r$ and the constrained solution α_r^{new} of the subproblems
2. Compute the dual objective, training and test errors in each iteration from the current iterate $\alpha^k \in \mathbb{R}^n$
3. Implement stopping criteria: Check if the KKT conditions given above are satisfied by the current iterate α^k upto the given tolerance `EPS`.
4. Compute the primal solution w from the dual solution α

d. (**3 Points**) Train the SVM classifiers for different choices of the error parameter $C = \{10, 100, 200, 500\}$ on the given USPS digit dataset `DIGITS01`. This is a dataset of handwritten digits containing the digits 0 and 1. Load the file `DIGITS01`. The variables `Xtrain` and `Xtest` contain respectively the training and test digit data (each digit is an image of 16×16, so we have 256 gray values) and the variables `ytrain` and `ytest` contains the class labels (-1 for the digit 0 and 1 for the digit 1) for the training and test examples. Plot the training and test errors that you obtained in each iteration of the coordinate descent method and save them as `DIGITS01TrainErrs_C.png` and `DIGITS01TestErrs_C.png` for each choice of C.

Hints:

a. Avoid `for` loops in your implementation. All the computations that you need to fill-in can be expressed as matrix-vector multiplication. The operator `.*` (see `MATLAB` help for `times`) that computes element-by-element multiplication of vectors might be helpful here.

b. Note that the dual variable α and the primal variable w are related by

$$w = \sum_{i=1}^{n} y_i \alpha_i x_i,$$

where x_i is the i^{th} row of X. Recall that the prediction at a point x is given by $\text{sign}(\langle w, x \rangle)$ (since the offset $b = 0$).

Submission:

- Create **one** zip/rar/tar.gz/tgz-file containing the m-file (`CoordinateDescentSVM.m`), your plots as .png files and the matlab data file `Solution.mat` containing `TrainErrs` and the `TestError` and send the file to: srangapu@mpi-inf.mpg.de. The filename has to follow the following convention:
`[group:A,B,C]_[matrikel numbers separated by underscore]_ex[nr].[extension]`
e.g. if you are in group B and your team members have matrikelnumbers 3503239, 3028258 and the current exercise number is 10 then the filename reads: `B_3503239_3028258_ex10.zip`.

Solution:

a. The Lagrangian function for the soft-margin SVM problem (without offset) is given by

$$L(w, \xi, \alpha, \beta) = \frac{1}{2} \|w\|^2 + \frac{C}{n} \sum_{i=1}^{n} \xi_i + \sum_{i=1}^{n} \alpha_i \big[1 - \xi_i - y_i \langle w, x_i \rangle \big] - \sum_{i=1}^{n} \beta_i \xi_i,$$

where $\alpha_i \geq 0$ and $\beta_i \geq 0$, $i = 1, \ldots, n$.

The Lagrangian dual function is given by

$$q(\alpha, \beta) = \min_{w, \xi} L(w, \xi, \alpha, \beta)$$

Note that the Lagrangian L is convex over (w, ξ) and hence the necessary and sufficient condition for a minimizer (w, ξ) are

$$\nabla_w L = 0, \quad \nabla_\xi L = 0$$

Thus any minimizer (w, ξ) should satisfy the conditions

$$w + \sum_{i=1}^{n} \alpha_i(-y_i x_i) = 0 \implies w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

and

$$\frac{C}{n}\mathbf{1} - \alpha - \beta = 0 \implies \beta = \frac{C}{n}\mathbf{1} - \alpha$$

Note here that the first condition gives the relationship between any primal and dual feasible points. The second condition relates the dual variables α and β and hence this would be one of the constraints of the dual problem.

Hence the dual function is given by (see Exercise 12 for a similar derivation with more explanation)

$$q(\alpha, \beta) = \min_{w,\xi} L(w, \xi, \alpha, \beta) =$$

$$\frac{1}{2}\left\langle \sum_{i=1}^{n} \alpha_i y_i x_i, \sum_{i=1}^{n} \alpha_i y_i x_i \right\rangle + \frac{C}{n}\sum_{i=1}^{n} \xi_i + \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \alpha_i \xi_i - \sum_{i=1}^{n} \alpha_i y_i \left\langle \sum_{j=1}^{n} \alpha_j y_j x_j, x_i \right\rangle - \sum_{i=1}^{n} \beta_i \xi_i$$

Using the condition $\beta = \frac{C}{n}\mathbf{1} - \alpha$, we have

$$q(\alpha, \beta) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle$$

Finally, the constraints of the dual problem are given by

$$\forall i, \ \alpha_i \geq 0, \quad \beta_i \geq 0, \quad \beta = \frac{C}{n}\mathbf{1} - \alpha$$

We can eliminate the dual variable β (it does not appear in the dual function) using the last equation. Since $\beta_i \geq 0$, we have $\alpha_i \leq \frac{C}{n}$. This yields the dual problem given in the exercise.

b. 1. The subproblem for coordinate r is given by

$$\begin{aligned} &\max_{0 \leq \alpha_r \leq \frac{C}{n}} \quad \alpha_r - \frac{1}{2}\alpha_r^2 y_r^2 \langle x_r, x_r \rangle - \frac{1}{2}\sum_{j \neq r} \alpha_r \alpha_j^{\text{old}} y_r y_j \langle x_r, x_j \rangle - \frac{1}{2}\sum_{i \neq r} \alpha_i^{\text{old}} \alpha_r y_i y_r \langle x_i, x_r \rangle \\ = &\max_{0 \leq \alpha_r \leq \frac{C}{n}} \quad \alpha_r - \frac{1}{2}\alpha_r^2 y_r^2 \langle x_r, x_r \rangle - \sum_{i \neq r} \alpha_i^{\text{old}} \alpha_r y_i y_r \langle x_i, x_r \rangle \end{aligned} \tag{2}$$

Note that the objective is a function of a single variable α_r and its first and second derivatives are given by

$$1 - \alpha_r y_r^2 \langle x_r, x_r \rangle - \sum_{i \neq r} \alpha_i^{\text{old}} y_i y_r \langle x_i, x_r \rangle$$

and

$$-y_r^2 \langle x_r, x_r \rangle = -\|x_r\|^2$$

Here we used the fact that $y_r^2 = 1$, for both positive and negative label y_r. Note that the second derivative is non-positive everywhere and hence the function is concave. Thus the necessary and sufficient condition for any (unconstrained) maximizer is that the first derivative vanishes:

$$1 - \alpha_r y_r^2 \langle x_r, x_r \rangle - \sum_{i \neq r} \alpha_i^{\text{old}} y_i y_r \langle x_i, x_r \rangle = 0$$

Adding and subtracting $\alpha_r^{\text{old}} y_r^2 \langle x_r, x_r \rangle$ yields

$$\underbrace{1}_{y_r^2} - \alpha_r y_r^2 \langle x_r, x_r \rangle + \alpha_r^{\text{old}} y_r^2 \langle x_r, x_r \rangle - \underbrace{\sum_{i=1}^{n} \alpha_i^{\text{old}} y_i y_r \langle x_i, x_r \rangle}_{y_r \langle w^{\text{old}}, x_r \rangle} = 0$$

This simplifies to

$$y_r \big(y_r - \langle w^{\text{old}}, x_r \rangle \big) + \alpha_r^{\text{old}} \langle x_r, x_r \rangle = \alpha_r \langle x_r, x_r \rangle$$
$$\implies \alpha_r = \alpha_r^{\text{old}} - \frac{y_r \big(\langle w^{\text{old}}, x_r \rangle - y_r \big)}{\langle x_r, x_r \rangle}$$

We now derive the optimizer for a general convex optimization problem in the presence of interval constraints. Note that (2) is a convex optimization problem (maximizing a concave function over convex set). In general, we always have for any function g

$$\max_{x \in C} g(x) = -\min_{x \in C} -g(x), \quad \arg\max_{x \in C} g(x) = \arg\min_{x \in C} -g(x)$$

So we derive the constrained minimizer for the convex optimization problem expressed in the following form:

$$\min_{l \leq x \leq u} f(x), \text{ where } f(x) \text{ is a convex function.}$$

Let x_0^* be an unconstrained minimizer of this problem and assume that $x_0^* \notin [l, u]$ (otherwise we are done). In the first case, assume that $l \leq u < x_0^*$. Then by convexity of f we have for any $x \in [l, x_0^*]$ and $\theta \in (0, 1)$,

$$f(\theta x + (1-\theta) x_0^*) \leq \theta f(x) + (1-\theta) f(x_0^*) < \theta f(x) + (1-\theta) f(x) = f(x)$$

This shows that for any chosen $x \in [l, x_0^*]$, the value of the function in the interval (x, x_0^*) is strictly smaller than $f(x)$. Thus the function is decreasing in the interval (x, x_0^*). Thus u has the smallest value of f among all the feasible x.
In the second case, assuming $x_0^* < l \leq u$, we see that the constrained minimizer is l. Thus, overall the constrained minimizer is given by the closest point of x_0^* in the interval $[l, u]$. Note that this includes the case when $x_0^* \in [l, u]$.

2. Let (w, ξ) and (α, β) be an optimal primal-dual pair. Then the KKT conditions are given by

Complementary-Slackness:

$$\alpha_i \big[1 - \xi_i - y_i \langle w, x_i \rangle \big] = 0, \quad \beta_i \xi_i = 0, \quad \forall i = 1, \ldots n$$

Primal feasibility:

$$\xi_i \geq 0, \quad y_i \langle w, x_i \rangle \geq 1 - \xi_i, \quad \forall i = 1, \ldots n$$

Dual feasibility:

$$\beta_i \geq 0, \quad 0 \leq \alpha_i \leq \frac{C}{n}, \quad \forall i = 1, \ldots n$$

Stationary Point:

$$\nabla_w L(w, \xi, \alpha, \beta) = 0 \implies w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

$$\nabla_\xi L(w,\xi,\alpha,\beta) = 0 \implies \beta = \frac{C}{n}\mathbf{1} - \alpha$$

Note that the stationary point conditions are satisfied by any primal-dual feasible pair $(\tilde{w},\tilde{\xi})$ and $(\tilde{\alpha},\tilde{\beta})$ not just the optimal pair (w,ξ) and (α,β).

Now we derive the conditions given the exercise from these KKT conditions. First note that

$$\alpha_i = 0 \implies \beta_i = \frac{C}{n} - \alpha_i = \frac{C}{n}$$

From the (second set of) complementary slackness conditions, we have $\xi_i = 0,\ \forall i = 1,\dots n$. Primal feasibility then implies

$$y_i \langle w, x_i\rangle \geq 1 \tag{3}$$

Next, if $0 < \alpha_i < \frac{C}{n}$, then $\beta_i = \frac{C}{n} - \alpha_i > 0$. Then the second set of complementary slackness conditions imply $\xi_i = 0,\ \forall i = 1,\dots n$. Moreover since α_i is non-zero, the first set of complementary slackness conditions imply $1 - \xi_i - y_i \langle w, x_i\rangle = 0,\ \forall i = 1,\dots n$. Thus we have

$$0 < \alpha_i < \frac{C}{n} \implies y_i \langle w, x_i\rangle = 1, \quad \forall i = 1,\dots n \tag{4}$$

Finally, if $\alpha_i = \frac{C}{n}$ we still have by the first set of complementary slackness conditions $1 - \xi_i - y_i \langle w, x_i\rangle = 0,\ \forall i = 1,\dots n$. Moreover, the primal feasibility of ξ_i implies that

$$\alpha_i = \frac{C}{n} \implies y_i \langle w, x_i\rangle = 1 - \xi_i \leq 1, \quad \forall i = 1,\dots n \tag{5}$$

Note that from the conditions (3), (4) and (5) and the stationary point conditions, which are satisfied by any primal-dual feasible pair we can similarly deduce all the KKT conditions. Hence these three conditions are equivalent to KKT conditions and can be used as the stopping criteria of the optimization method.

c. Note that the update of α^{new} in each iteration of the coordinate descent is straightforward. On the other hand, one needs to compute in each iteration the quantity $\langle w^{\text{new}}, x_i\rangle$ in order to test the optimality conditions. One could either update the primal variable w^{new} in each iteration efficiently or directly compute this quantity without actually updating w^{new}. Here we provide both versions.

Version 1 - Updating the primal variable w^{new}: Let $X \in \mathbb{R}^{n\times d}$ be the design matrix where row i of X contains the training example x_i. Then $w^{\text{new}} = X'(y.*\alpha^{\text{new}})$, where $.*$ is the component-wise multiplication.

Note that in each iteration, only one component of α^{new} changes while the rest remain the same. Thus, when the coordinate r is chosen, we have

$$\begin{aligned} w^{\text{new}} = X'(y.*\alpha^{\text{new}}) &= X'(y.*\alpha^{\text{old}}) + X'(y.*(\alpha^{\text{new}} - \alpha^{\text{old}})) \\ &= X'(y.*\alpha^{\text{old}}) + X(:,r)'y_r(\alpha_r^{\text{new}} - \alpha_r^{\text{old}}) \end{aligned}$$

Thus the primal variable w can be updated efficiently by just adding a vector; however one needs to do a matrix-vector multiplication $(X'w^{\text{new}})$ in each iteration to test the optimality conditions.

```
    function [alpha, w, TrainErrs, TestErrs] = ...
        CoordinateDescent_SVM_SampleSolution(Xtrain, ytrain, C, Xtest, ytest)

    assert(sum(unique(ytrain) == [-1; 1]) == 2); % assert the class labels are ...
        -1 and 1
    n = size(Xtrain, 1);
    alpha = sparse(n, 1);

    w = Xtrain'*(alpha.*ytrain); % the primal variable
    TrainErrs = []; TestErrs = [];

    counter = 0; iter = 0; EPS = 1e-3;
    CONVERGENCE = false;
    while ¬CONVERGENCE

        r = counter+1;

        % Solve the subproblem for coordinate r without any constraints:
        Er = Xtrain(r,:)*w - ytrain(r);
        alpha_prime = alpha(r) - ytrain(r)*Er/(Xtrain(r, :)*Xtrain(r, :)');

        % Project the solution to the interval [0, C/n]
        alpha_new = max(0, min(alpha_prime, C/n));
        Δ_alpha = alpha_new - alpha(r);
        alpha(r) = alpha_new;

        iter = iter+1;
        w = w + Xtrain(r,:)'*Δ_alpha*ytrain(r); %w = Xtrain'*(alpha.*ytrain);
        Xtrainw = Xtrain*w;
        TrainErrs(iter) = sum(sign(Xtrainw) ≠ ytrain)/length(ytrain);
        TestErrs(iter) = sum(sign(Xtest*w) ≠ ytest)/length(ytest);

        if rem(iter, 100) == 0
            % Compute the Dual objective
            dualObj = sum(alpha) - 0.5* norm(w)^2;
            display(['iter: ', num2str(iter), ' Dual Objective: ', ...
                num2str(dualObj)]);
        end

        % Check the KKT conditions for convergence
        ix_0 = alpha == 0; ix_inside = alpha > 0 & alpha < C/n; ix_Cn = alpha ...
            == C/n;
        dummy = ytrain.*(Xtrainw);
        residuals_0 = 1 - dummy(ix_0);
        residuals_inside = abs(dummy(ix_inside) - 1);
        residuals_Cn = dummy(ix_Cn) - 1;

        if rem(iter, 100) == 0
            display(['Residuals: ', num2str([sum(residuals_0>EPS) ...
                sum(residuals_inside>EPS) sum(residuals_Cn>EPS)]), ' TrainErr: ...
                ', num2str(TrainErrs(iter)), ' TestErr: ', ...
                num2str(TestErrs(iter))]);
        end
        if ¬sum(residuals_0 > EPS) && ¬sum(residuals_inside > EPS) && ¬...
            sum(residuals_Cn > EPS)
            CONVERGENCE = true;
        end

        counter = rem(counter+1, n);

    end
    dualObj = sum(alpha) - 0.5* norm(w)^2;
    display(['Final iter: ', num2str(iter), ' Dual Objective: ', ...
        num2str(dualObj)]);
    display(['Final Residuals: ', num2str([sum(residuals_0>EPS) ...
        sum(residuals_inside>EPS) sum(residuals_Cn>EPS)])]);
     % This is redundant as w is already updated inside the loop
    w = Xtrain'*(alpha.*ytrain); % the primal variable
end
```

Version 2 - Efficient solution without updating w^{new}: Now we show how to update the quantity $\langle w^{\text{new}}, x_i \rangle$ without computing w^{new}. When the coordinate r is being updated, we have

$$\begin{aligned} X'w^{\text{new}} = X'X(\alpha^{\text{new}}.*y) &= X'X(\alpha^{\text{old}}.*y) + X'X((\alpha^{\text{new}} - \alpha^{\text{old}}).*y) \\ &= X'X(\alpha^{\text{old}}.*y) + (X'X)(:,r)y_r(\alpha_r^{\text{new}} - \alpha_r^{\text{old}}) \\ &= K(\alpha^{\text{old}}.*y) + K(:,r)y_r(\alpha_r^{\text{new}} - \alpha_r^{\text{old}}). \end{aligned}$$

Here $K = XX'$ is the matrix where the $(i,j)^{th}$ element is given by $\langle x_i, x_j \rangle$. This is in general the kernel matrix specifying the similarity between all pairs of training examples.

Again we see that once we compute the matrix K and the matrix-vector multiplication $K(\alpha^{\text{old}}.*y)$, then updating $X'w^{\text{new}}$ is cheaply done by just adding a vector $K(:,r)y_r(\alpha_r^{\text{new}} - \alpha_r^{\text{old}})$.

Furthermore, the dual function can be computed by just knowing $X'w$:

$$\Psi(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n} y_i\alpha_i \langle w, x_i \rangle = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}(y.*\alpha)'(X'w)$$

```
function [alpha, w, TrainErrs, TestErrs] = ...
    CoordinateDescent_SVM_SampleSolution_Fast(Xtrain, ytrain, C, Xtest, ytest)

assert(sum(unique(ytrain) == [-1; 1]) == 2); % assert the class labels are ...
    -1 and 1
n = size(Xtrain, 1);
ntest = size(Xtest, 1);
alpha = sparse(n, 1);

%w = Xtrain'*(alpha.*ytrain); % the primal variable
TrainErrs = []; TestErrs = [];

Ktrain = Xtrain*Xtrain'; % the similarity matrix with inner products
Ktest = Xtest*Xtrain'; % For prediction on the test points
coeff = ytrain.*alpha;
yhat_train = Ktrain*coeff;
yhat_test = Ktest*coeff;

counter = 0; iter = 0; EPS = 1e-3;
CONVERGENCE = false;
while ¬CONVERGENCE

    r = counter+1;

    % Solve the subproblem for coordinate r without any constraints:
    Er = yhat_train(r) - ytrain(r);
    alpha_prime = alpha(r) - ytrain(r)*Er/Ktrain(r,r);

    % Project the solution to the interval [0, C/n]
    alpha_new = max(0, min(alpha_prime, C/n));
    Δ_alpha = alpha_new - alpha(r);
    alpha(r) = alpha_new;
    coeff(r) = ytrain(r)*alpha(r);

    iter = iter+1;
    yhat_train = yhat_train + Ktrain(:,r)*ytrain(r)*Δ_alpha;
    yhat_test = yhat_test + Ktest(:,r)*ytrain(r)*Δ_alpha;
    TrainErrs(iter) = sum(sign(yhat_train) ≠ ytrain)/n;
    TestErrs(iter) = sum(sign(yhat_test) ≠ ytest)/ntest;

    if rem(iter, 100) == 0
        % Compute the Dual objective
        dualObj = sum(alpha) - 0.5*(coeff'*yhat_train);
```

```
            display(['iter: ', num2str(iter), ' Dual Objective: ', ...
                num2str(dualObj)]);
        end

        % Check the KKT conditions for convergence
        ix_0 = alpha == 0; ix_inside = alpha > 0 & alpha < C/n; ix_Cn = alpha ...
            == C/n;
        dummy = ytrain.*(yhat_train);
        residuals_0 = 1 - dummy(ix_0);
        residuals_inside = abs(dummy(ix_inside) - 1);
        residuals_Cn = dummy(ix_Cn) - 1;

        if rem(iter, 100) == 0
            display(['Residuals: ', num2str([sum(residuals_0>EPS) ...
                sum(residuals_inside>EPS) sum(residuals_Cn>EPS)]), ' TrainErr: ...
                ', num2str(TrainErrs(iter)), ' TestErr: ', ...
                num2str(TestErrs(iter))]);
        end
        if ¬sum(residuals_0 > EPS) && ¬sum(residuals_inside > EPS) && ¬...
            sum(residuals_Cn > EPS)
            CONVERGENCE = true;
        end

        counter = rem(counter+1, n);

    end

    dualObj = sum(alpha) - 0.5*(coeff'*yhat_train);
    display(['Final iter: ', num2str(iter), ' Dual Objective: ', ...
        num2str(dualObj)]);
    display(['Final Residuals: ', num2str([sum(residuals_0>EPS) ...
        sum(residuals_inside>EPS) sum(residuals_Cn>EPS)])]);
    %z = sign(Xtrain*w); z(alpha≠0) = 2; GD_PlotLabels(Xtrain, z, 'a'); hold ...
        on; x1 = -2:0.1:2; x2 = -w(1)*x1/w(2); plot(x1, x2);

    w = Xtrain'*(alpha.*ytrain); % the primal variable
end
```

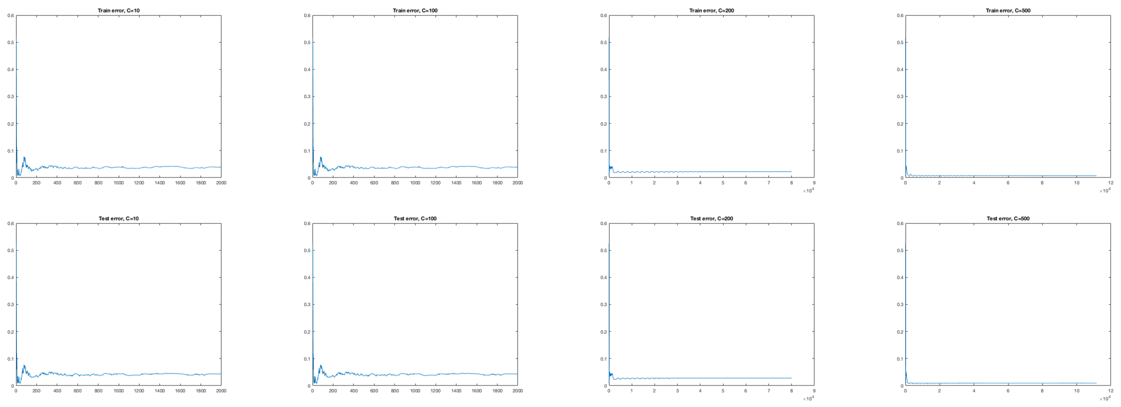

Figure 1: Training (top row) and test (bottom row) errors after each iteration of the coordinate descent method. Note that in this case, solving the optimization problem to full accuracy is not necessary, because the errors do not decrease much after the first few thousand iterations.