EE4530 APPLIED CONVEX OPTIMIZATION ASSIGNMENT

Erik Hagenaars
4272404

Husain Kapadia
4707915

***Abstract*—A Linear Support Vector Machine was created to classify vectors to either one of the two classes. The vectors have two dimensional coordinates making it possible to create clear visual representations of the data. In order to create this SVM, first a convex optimization problem was formulated. The SVM was modeled with a favourite off-the-shelf solver and evaluated. Also an own-made algorithm based on the (sub)gradient descent algorithm has been implemented to solve this convex problem. These two implementations are evaluated on performance in respect to number of iterations, CPU time, and convergence of the algorithms.**

***Keywords*—SVM, Convex optimization, (sub)Gradient descent**

I. Introduction

Support Vector Machines (SVM) are widely used for classification problems. The SVM is a type of supervised machine learning where a machine will be trained with a training set. After this training, a model or machine is created. This model should be able to classify the training set without error and classify new sets of data. This assignment will be taking a closer look on the binary classification where only 2 classes are possible as an output of a set.

This assignment is given by the supervisors of course EE4530 [1] and consists of two parts: The first part is to formulate the SVM problem to a suitable convex optimization problem where the definition and possible relaxations are well defined and motivated. The second part is to implement this binary classifier. Given with the assignment is a dataset containing a training set and a test set. Where only the training is allows to train the classifier with. Two approaches for this implementation were used. A dedicated piece of software for convex optimization problems and a own-made algorithm based on the (sub)gradient descent algorithm.

This essay will be divided into multiple parts. First, a description of the problem is given and the sets are evaluated including the classification in Section II. Secondly the SVM problem is formulated into a convex optimization problem in Section III. Thirdly, dedicated software is used for the implementation of the problem and the result are given in Section IV. After this, the problem is implemented using own-made algorithms and the result are given in Section V. Lastly, an evaluation is conducted among the results. This results and revaluation can be found in Section VI.

II. Problem description

The objective is to classify a two dimensional vector to either class 1 or class -1. This problem can be defined as a binary classification problem. Given along with the assignment were two datasets: One training set consisting of 100 vectors ($100x2$) and secondly, a test set consisting of 900 vectors. For both sets, the classification is already known using a labels vector ($Nx1$). These two labels vector have a different use: The labels vector for the training set is supposed to be known and used for training the SVM and the labels for the test set is used to check the success rate of the SVM.

Since both of the sets have known labels, the coordinates can be scattered in a figure. Fig. 1 shows the scattering of the training set and Fig. 2 shows the scattering of the test set.

It can be seen that a distinct line could be drawn to divide the different classes. It is therefore expected that a linear SVM should suffice for classifying the vectors. This eases the problem.

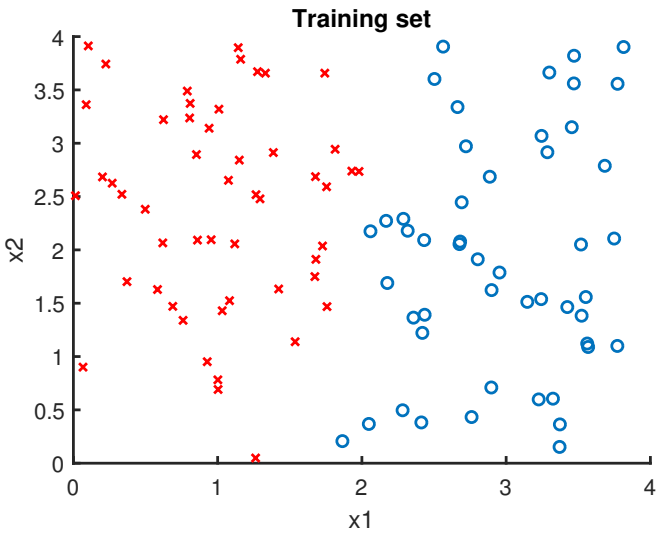

Fig. 1. The trainingt set with the corresponding labels.

III. Convex Optimization of a SVM

In order to convert the SVM problem into a convex problem, two parts are needed:

- A cost function to minimize.
- A constraint to subject the minimization to.

The hyperplane of a SVM is modeled in Eq. 1. Where $\mathbf{w}$ and $\mathbf{w_0}$ are the hyperplane's parameters and x the coordinates of an input.

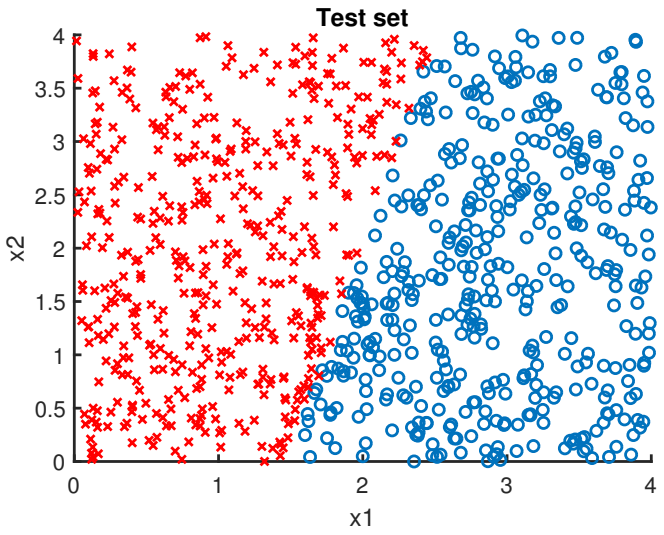

Fig. 2. The test set with the corresponding labels.

$$g(x) = \mathbf{w^T x} + w_0 = 0; \tag{1}$$

In the book Pattern Recognition[2], the distance between a vector x and a hyperplane is defined as in Eq. 2.

$$z = \frac{|g(\mathbf{x}))|}{\|\mathbf{w}\|} \tag{2}$$

The objective is to create a maximum margin, this margin should be maximized on either sides of the planes. Resulting in maximizing $\frac{2*|g(\mathbf{x}))|}{\|\mathbf{w}\|}$. Which can be converted into a minimization problem when inverted.

The hyperplane will split the classes in two. For all the coordinates of class 1, the equation $\mathbf{w^T x} + w_0$ should be greater or equal to 1 and for class -1 less than or equal to -1. This leads to the constraints shown in Eq. 5.

$$\mathbf{w}^T\mathbf{x} + w_0 \geq 1 \tag{3}$$
$$\mathbf{w}^T\mathbf{x} + w_0 \leq -1 \tag{4}$$
$$y_i(\mathbf{w}^T\mathbf{x} + w_0) \leq 1 \tag{5}$$

Combining Eq. 5 and the cost function. The convex optimization problem is formulates in Eq. 6.

$$\begin{aligned} \underset{\mathbf{w}}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x} + w_0) \leq 1,\ i = 1, \ldots, N. \end{aligned} \tag{6}$$

We like to call this the 'Basic' method. In this method, minimizing this cost function creates a maximum margin since the denominator of the distance to a point is minimized. The inequality constraint had the function of actually fitting every input to the correct class. If the data would not be linearly separable, the convex problem could not be solved since the inequality constraint can never be satisfied. To prevent this problem, slack variables can be created to allow a small error of the classification.

Two errors can be distinguished: Inputs which are classified correctly but don't satisfy the maximum margin constraint. And inputs which are miss-classified. Of course, the second error should be weighted more heavily, or even prohibited. The slack variable is denoted as ξ_i. Also a cost weight variable C is introduced. The resulting convex optimization problem is formulated in Eq. 7[3]. This method is named as the 'Slack' method.

$$\begin{aligned} \underset{\mathbf{w},\xi}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i^N \xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x} + w_0) \leq 1 - \xi_i,\ i = 1, \ldots, N. \\ & \xi_i \geq 0 \end{aligned} \tag{7}$$

Every error is added to the cost function weighted by C. In the inequality constraint, the small errors are allowed by this ξ_i which contributes to the cost function. If C is small, the error would occur more, while a large C will result in the same solutions as the convex optimization without slack variables. We determined the value of C by trial and error, by observing the changes in accuracy. We choose $C = 5$ for this experiment since it yields a maximum accuracy of 97.44%.

For future reasons, this latest convex optimization problem could be converted into a unconstrained optimization problem shown in Eq. 8[3]. This form will allow for an easy steepest descend algorithm, which will be explained in Section V. When $y_i(\mathbf{w}^T\mathbf{x}_i + w_0)$ is smaller than one, it will contribute to the loss or-else the value 0 will be taken. This method is called the 'Loss' method. We use the same value of C as above.

$$\underset{\mathbf{w}}{\text{minimize}} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i^N max[0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + w_0)] \tag{8}$$

IV. Software Implementation

To solve the convex optimization problem, dedicated software can be used. In this case, the current convex optimization problem will be solved by CVX, a package for specifying and solving convex programs[4][5]. The three equations from Section III will be implemented using this tool. The corresponding codes are shown in the the Appendix.

The tool begins similar as a `for` loop with `cvx_begin` and `cvx_end`. Within the statements, information about the convex problem can be defined starting with the variables which can be altered. In this case $\mathbf{w}$ and b and in one case ξ are variables. This is done using the command `variables w(p) b e(n)`. The minimization function can be defined using `minimize f(x)` where $f(x)$ is the minimization function. Lastly, constraints can be added with the command `subject to` followed by the constraints.

V. Algorithm Implementation

In order to create a (sub)gradient descent algorithm for the convex optimization problem, the Pegasus algorithm will be implemented, which was defined by Sontag[6]. The Pegasus algorithm uses the unconstrained problem formulation of Eq. 8. This form shows two terms; a regularization term $\frac{1}{2}\|\mathbf{w}\|^2$ and the empirical loss $C\sum_i^N max[0, 1-y_i(\mathbf{w}^T, \mathbf{x}_i)]$. When scaling this function with $C = \frac{1}{N\lambda}$, the function can be converted as shown in Eq. 9.

$$f(\mathbf{w}) := \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_i^N max[0, 1 - y_i(\mathbf{w}^T, \mathbf{x}_i)] \quad (9)$$

The first step of the (sub)gradient algorithm is taking the (sub)gradient of $f(\mathbf{w})$ as shown in Eq. 10. This last part is only a linear function when $y_i\mathbf{w} * x_i < 1$ with slope $-y_i x_i$.

$$\frac{df(\mathbf{w})}{d\mathbf{w}} = \lambda\mathbf{w} + \frac{d}{d\mathbf{w}} max[0, 1 - y_i(\mathbf{w}^T, \mathbf{x}_i)] \quad (10)$$

The update function for the variable $\mathbf{w}$ will then be given as in Eq. 11. Here the λ is the learning rate with $\eta = 1/(t\lambda)$ for scaling reasons. The stopping criteria will be a number of iterations. The implementation can be found in the Appendix. λ has been found using trial and error methods and comparing the resulting accuracy.

$$\begin{aligned} \mathbf{w}_{t+1} &= \mathbf{w}_t - \eta_t \bigtriangledown_{\mathbf{w}} f(\mathbf{w}) \\ &= (1-\eta\lambda)\mathbf{w}_t + \eta_t y_i xi, \quad y_i\mathbf{w}_t * x_i < 1 \\ &= (1-\eta\lambda)\mathbf{w}_t, \qquad\qquad Else \end{aligned} \quad (11)$$

VI. Evaluation

A. Results

1) CVX: Table I shows the performance of different methods using the CVX solver based on the number of iterations, CPU time (averaged over 10 measurements) and the accuracy over test data. CVX uses the SDPT3 algorithm[7] to solve the optimization problem, which is based on the semi-definite quadratic linear programming technique. We observe that the method with the 'Loss function' yields best results as compared to other methods in terms of accuracy with the time and the number of iterations as much as the 'Basic' method. This reason is one of the motives to choose the 'Loss' method over others and continue implementing it for the sub-gradient solver.

TABLE I. CVX Results for different Methods

Method	Iter	CPU time [s]	Accuracy [%]	Algorithm
Basic	11	0.124	97.67	SDPT3
Slack	18	0.225	97.44	SDPT3
Loss	12	0.157	97.78	SDPT3

2) (sub)gradient descent: As seen in Table II, the accuracy and run time increase by the number of iterations. This accuracy is calculated based on the test results. Best results were seen near 100 iterations where the accuracy oscillates around 99.5%. The run time was linear over the number of iterations.

TABLE II. Result (sub)gradient descent

Iterations	Run time [s]	Accuracy [%]
1	0.0005	66.2222
5	0.0010	87
20	0.0026	92.5556
50	0.0068	96.8889
100	0.0123	99.5556

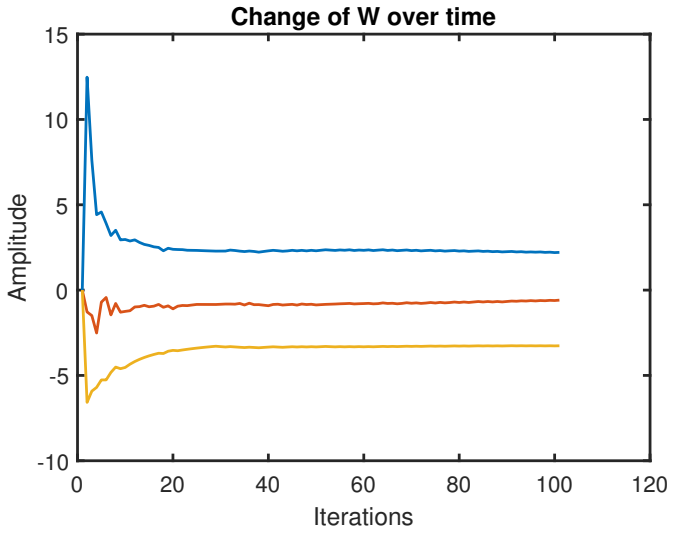

Fig. 3. Convergence of the w and b parameter. Blue and yellow for $\mathbf{w}$. Red for b

B. Conclusion

Based on the results we gathered and the observations we made for this experiment, we conclude that the 'Loss' method is a very general, suitable and efficient technique to solve a Support Vector Machine problem. This is mainly because the objective function for this problem does not involve any constraints and hence solving it is faster using convex optimization problem solvers like CVX and also by the (sub)gradient method that was described earlier. Along with it being fast, this method also yields accurate results.

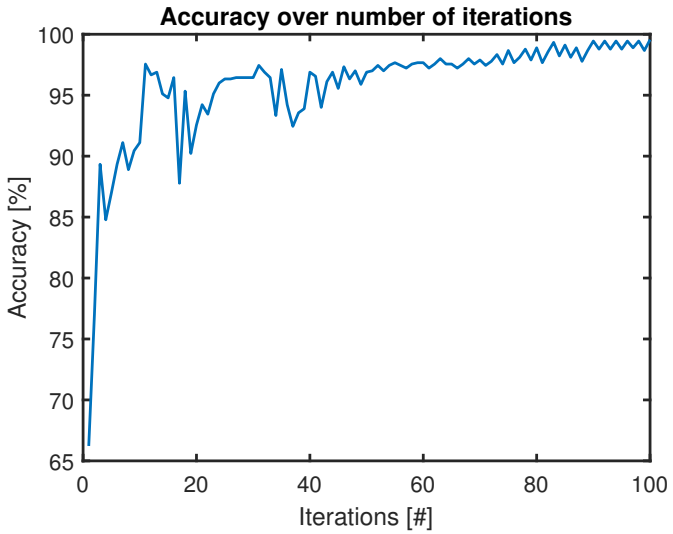

Fig. 4. Convergence of the accuracy

After finalizing on the objective function, we proceed to compare the CVX solver with the (sub)gradient algorithm in terms of CPU time required and accuracy over test data. We observe that the sub-gradient method requires more iterations to reach the optimal solution as compared the the CVX solver, however the time required by the sub-gradient algorithm is way low than the CVX solver. This is due to the fact that the CVX is a general and complex solver and before it starts optimizing, it needs to convert the objective function to a Lagrangian based on the defined constraints. The lesser the constraints, the faster the CVX solves the problem. We eliminate this step of converting the problem to it's dual which saves us the time. In terms of accuracy, the sub-gradient method beats the CVX solver.

Based on our experiment, we believe that the learning rate (step size) that we have defined for our problem decreases as the number of iterations increase. This means that we take infinitesimally small steps as we get closer to the optimal solution. This justifies why the (sub)gradient method requires more iterations. Moreover, the choice of λ was made based on the highest accuracy we obtained by iterating over a set of different values.

Hence, we can conclude that the sub-gradient method outperforms the CVX solver for our experiment mainly because the sub-gradient algorithm was designed to be more specific for this problem.

REFERENCES

[1] S. Chepuri, "Et4350 applied convex optimization assignment: Linear support vector machines," a PDF file containing the assignment related to this essay. [Online]. Available: http://ens.ewi.tudelft.nl/Education/courses/ee4530/ projects/Linear_SVM.pdf

[2] S. Theodoridis and K. Koutroumbas, *Pattern Recognition, Fourth Edition*, 4th ed. Academic Press, 2008.

[3] A. Zisserman, "Lecture 2: The svm classifier," a PowerPoint Presentation from the Oxfort University. [Online]. Available: http://www.robots.ox.ac.uk/ az/lectures/ml/lect2.pdf

[4] I. CVX Research, "CVX: Matlab software for disciplined convex programming, version 2.1," http://cvxr.com/cvx, Aug. 2012.

[5] M. Grant and S. Boyd, "Graph implementations for nonsmooth convex programs," in *Recent Advances in Learning and Control*, ser. Lecture Notes in Control and Information Sciences, V. Blondel, S. Boyd, and H. Kimura, Eds. Springer-Verlag Limited, 2008, pp. 95–110, http://stanford.edu/ boyd/graph_dcp.html.

[6] D. Sontag, "Support vector machines (svms)," a PowerPoint Presentation from the New York University. [Online]. Available: https://people.csail.mit.edu/dsontag/courses/ml16/slides/lecture5.pdf

[7] M. T. K.C. Toh and R. Tutuncu, "Sdpt3 — a matlab software package for semidefinite programming, optimization methods and software, 11," pp. 545–581, 1999.

APPENDIX A
MATLAB CODE

A. Plotting the sets

plot_labels.m

```
clear
clc
close all

%% Loading testsets
load('linear_svm.mat');

%% Plot training set
x = X_train(:,1); y = X_train(:,2);
k1 = find(labels_train == 1); k2 = find(labels_train == -1);
x1 = x(k1); y1 = y(k1);
x2 = x(k2); y2 = y(k2);

figure; hold on;
scatter(x1, y1);
scatter(x2, y2, 'rx');
title('Training set'); xlabel('x1'); ylabel('x2');

%% plot test set
x = X_test(:,1); y = X_test(:,2);
k1 = find(labels_test == 1); k2 = find(labels_test == -1);
x1 = x(k1); y1 = y(k1);
x2 = x(k2); y2 = y(k2);

figure; hold on;
scatter(x1, y1);
scatter(x2, y2, 'rx');
title('Test set'); xlabel('x1'); ylabel('x2');
```

B. CVX solver

cvx_basic.m

```
clc;
clear;
close all;

%% Load sets
load('linear_svm.mat');
[n, p] = size(X_train);

%% Plot training data
x_c1 = X_train(labels_train == 1, :);
x_c2 = X_train(labels_train == -1, :);

figure; hold on;
scatter(x_c1(:,1), x_c1(:,2), 'ro');
scatter(x_c2(:,1), x_c2(:,2), 'b+');

%% CVX software
cvx_begin
    variables w(p) b
    minimize(0.5 * norm(w));
    subject to
        labels_train.*(X_train*w + b*ones(n,1)) >= ones(n,1);
```

```
cvx_end;

%% Plot solution on training set
x2 = axis;
x1 = -(x2*w(2)+ b)/w(1);

plot(x1,x2,'k-','LineWidth',1)
title('Training Set'); xlabel('x1'); ylabel('x2');

%% Plot solution on test set
x_c1 = X_test(labels_test == 1, :);
x_c2 = X_test(labels_test == -1, :);

figure; hold on
scatter(x_c1(:,1), x_c1(:,2), 'ro');
scatter(x_c2(:,1), x_c2(:,2), 'b+');
plot(x1,x2,'k-','LineWidth',1)
title('Test Set'); xlabel('x1'); ylabel('x2');

%% Check accuracy
a = X_test*w + b.*ones(length(X_test),1);
a(a > 0) = 1;
a(a < 0) = -1;

c = (a == labels_test);
accuracy = sum(c)/length(c)*100
```

---

cvx_slack.m

```
clc;
clear;
close all;

%% Load sets
load('linear_svm.mat');
[n, p] = size(X_train);

%% Plot training data
x_c1 = X_train(labels_train == 1, :);
x_c2 = X_train(labels_train == -1, :);

figure; hold on;
scatter(x_c1(:,1), x_c1(:,2), 'ro');
scatter(x_c2(:,1), x_c2(:,2), 'b+');

%% CVX software
C = 0.5;
cvx_begin
    variables w(p) b e(n)

    minimize(norm(w) + C*sum(e));

    subject to
        labels_train.*(X_train*w + b*ones(n,1)) >= ones(n,1) - e;
        e >= 0;
        e <= 1;
cvx_end;

%% Plot solution on training set
x2 = axis;
x1 = -(x2*w(2)+ b)/w(1);
```

```
33
34 plot(x1,x2,'k-','LineWidth',1)
35 title('Training Set'); xlabel('x1'); ylabel('x2');
36
37 %% Plot solution on test set
38 x_c1 = X_test(labels_test == 1, :);
39 x_c2 = X_test(labels_test == -1, :);
40
41 figure; hold on
42 scatter(x_c1(:,1), x_c1(:,2), 'ro');
43 scatter(x_c2(:,1), x_c2(:,2), 'b+');
44 plot(x1,x2,'k-','LineWidth',1)
45 title('Test Set'); xlabel('x1'); ylabel('x2');
46
47 %% Check accuracy
48 a = X_test*w + b.*ones(length(X_test),1);
49 a(a > 0) = 1;
50 a(a < 0) = -1;
51
52 c = (a == labels_test);
53 accuracy = sum(c)/length(c)*100
```

---

cvx_loss.m

```
1  clc;
2  clear;
3  close all;
4
5  %% Load sets
6  load('linear_svm.mat');
7  [n, p] = size(X_train);
8
9  %% Plot training data
10 x_c1 = X_train(labels_train == 1, :);
11 x_c2 = X_train(labels_train == -1, :);
12
13 figure; hold on;
14 scatter(x_c1(:,1), x_c1(:,2), 'ro');
15 scatter(x_c2(:,1), x_c2(:,2), 'b+');
16
17 %% CVX software
18 C = 1;
19 cvx_begin
20     variables w(p) b
21
22     minimize(norm(w) + C*sum(max([zeros(n,1); 1-(labels_train.*(X_train*w + b*ones(n,1)
       ))])));
23 cvx_end;
24
25 %% Plot solution on training set
26 x2 = axis;
27 x1 = -(x2*w(2)+ b)/w(1);
28
29 plot(x1,x2,'k-','LineWidth',1)
30 title('Training Set'); xlabel('x1'); ylabel('x2');
31
32 %% Plot solution on test set
33 x_c1 = X_test(labels_test == 1, :);
34 x_c2 = X_test(labels_test == -1, :);
35
36 figure; hold on
```

```
scatter(x_c1(:,1), x_c1(:,2), 'ro');
scatter(x_c2(:,1), x_c2(:,2), 'b+');
plot(x1,x2,'k-','LineWidth',1)
title('Test Set'); xlabel('x1'); ylabel('x2');

%% Check accuracy
a = X_test*w + b.*ones(length(X_test),1);
a(a > 0) = 1;
a(a < 0) = -1;

c = (a == labels_test);
accuracy = sum(c)/length(c)*100
```

C. Trial and Error

steepest_descent.m

```
clc;
clear;
close all;

%% Load sets
load('linear_svm.mat');
[n, p] = size(X_train);

%% Plot training data
x_c1 = X_train(labels_train == 1, :);
x_c2 = X_train(labels_train == -1, :);

figure; hold on;
scatter(x_c1(:,1), x_c1(:,2), 'ro');
scatter(x_c2(:,1), x_c2(:,2), 'b+');

%% Iterate to find best lambda
accuracy_best = 0;
%for labda = 0.001 : 0.0001 : 0.1    first run
for labda = 0.007 : 0.00001 : 0.008 % after debugging
w_ = [0; 0; 0];
t = 0;
x_ = [X_train, ones(100,1)];

it = 100;
for i = 1 : it
    for j = 1 : length(x_)
        t = t + 1;
        eta = 1/(t*labda);
        if labels_train(j)*(x_(j,:)*w_) < 1
            w_ = (1 - eta*labda)*w_ + eta*labels_train(j)*x_(j,:)';
        else
            w_ = (1 - eta*labda)*w_;
        end
    end
end
w = w_(1:2);
b = w_(3);

%% Check accuracy
a = X_test*w + b.*ones(length(X_test),1);
a(a > 0) = 1;
a(a < 0) = -1;

```

```
c = (a == labels_test);
accuracy = sum(c)/length(c)*100;

if accuracy_best < accuracy
    accuracy_best = accuracy;
    w_best = w;
    b_best = b;
    labda_best = labda;
end
end

%% print best results
accuracy_best
w_best
b_best
labda_best
```

D. Trial and Error

plot_iterations.m

```
clc;
clear;
close all;

%% Load sets
load('linear_svm.mat');
[n, p] = size(X_train);

%% Iterations
labda = 0.0076; % best found in steepest_descent.m
x_ = [X_train, ones(100,1)];
acc = [];
runtime = [];

for it = 1:100  % for different number of iterations
w_ = [0; 0; 0];
t = 0;
w_plot = w_;

tic
for i = 1 : it  %steepest descent algorithm
    for j = 1 : length(x_)
        t = t + 1;
        eta = 1/(t*labda);
        if labels_train(j)*(x_(j,:)*w_) < 1
            w_ = (1 - eta*labda)*w_ + eta*labels_train(j)*x_(j,:)';
        else
            w_ = (1 - eta*labda)*w_;
        end
    end
    w_plot = [w_plot w_];
end
r = toc;
runtime = [runtime r];

%% Check accuracy
a = X_test*w_(1:2) + w_(3).*ones(length(X_test),1);
a(a > 0) = 1;
a(a < 0) = -1;

```

```
c = (a == labels_test);
accuracy = sum(c)/length(c)*100;
acc = [acc accuracy];
end

%% plotting
figure
plot(acc);
title('Accuracy over number of iterations');
xlabel('Iterations [#]');
ylabel('Accuracy [%]');

figure
plot(w_plot');
title('Change of W over time');
xlabel('Iterations');
ylabel('Amplitude');
```

E. Final Algorithm

final.m

```
clc;
clear;
close all;

%% Load sets
load('linear_svm.mat');
[n, p] = size(X_train);
x_ = [X_train, ones(100,1)];

%% initialize variables
labda = 0.0076;
it = 100;
w_ = [0; 0; 0];
t = 0;

w_plot = [w_];
tic
%% start algorithm
for i = 1 : it % number of iterations
    for j = 1 : length(x_) % for every sample
        t = t + 1;
        eta = 1/(t*labda);  % calculate eta for every iteration
        if labels_train(j)*(x_(j,:)*w_) < 1     % if misqualification
            w_ = (1 - eta*labda)*w_ + eta*labels_train(j)*x_(j,:)';
        else
            w_ = (1 - eta*labda)*w_;
        end
    end
    w_plot = [w_plot w_];   % record w movement
end
toc
w = w_(1:2);     % extract w
b = w_(3);       % extract b

%% Check accuracy
a = X_test*w + b.*ones(length(X_test),1);
a(a > 0) = 1;
a(a < 0) = -1;

```

```
c = (a == labels_test);
accuracy = sum(c)/length(c)*100;

%% plots
x_c1 = X_train(labels_train == 1, :);
x_c2 = X_train(labels_train == -1, :);

figure; hold on
scatter(x_c1(:,1), x_c1(:,2), 'ro');
scatter(x_c2(:,1), x_c2(:,2), 'b+');

%% animate movement of algorithm
for i = 1 : length(w_plot)
x2 = axis;
x1 = -(x2*w_plot(2,i)+ w_plot(3,i))/w_plot(1,i);

plot(x1,x2)
pause(0.1)
end
```