Aaron Dharna

Convex Optimization Final Project

Introduction

Typically in machine learning, learning-algorithm optimization is powered by a loss function. Loss functions measure how well or poorly the current algorithm parameterization is classifying (or regressing) a task and provides gradient feedback by taking the derivative of the loss with respect to the weights/tunable-parameters of the algorithm. One standard loss function is that of mean-squared-error:

$$\mathcal{L}(x, y) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2$$

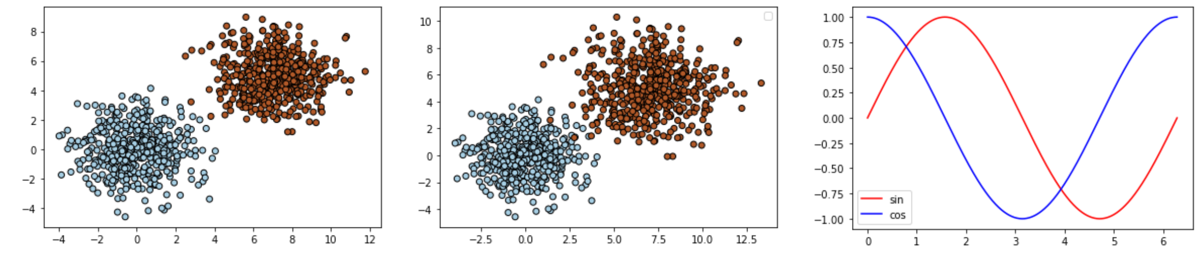
Then, to determine how the algorithm parameters should change, one simply takes the derivative of $\mathcal{L}$ with repsect to the θ. This approach is commonly used in algorithms ranging from logistic regression, neural networks, and many others. This work introduces Support Vector Machines (SVMs) and their motivation, and in particular will focus on how SVMs are actually just a convex optimization problem. The primal formualation is discussed and then from that the Dual formulation is derived. Datasets and Matlab CVX code are provided for each case as well as python scikit-learn code showing a population "standard" library.

Support Vector Machines

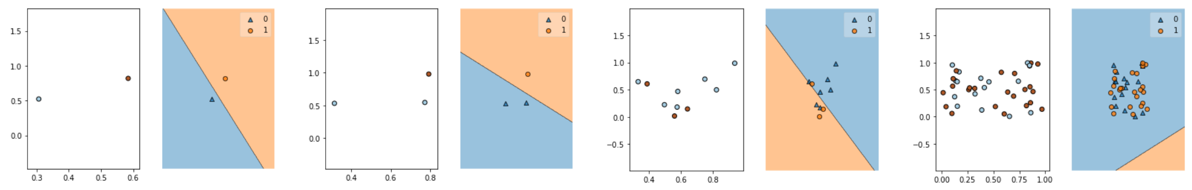
Motivation

If I were to sum-up all of supervised machine learning (i.e. learning to discriminate classes) in one idea it would: The science of finding decision boundaries when given a representative support of the desired distributions. There are bayesian ways to do this that use probabilities and densities, but given that density estimation can be an ill-posed problem, another method is to directly with the decision function/boundary. To do this one creates a patermeterized family of functions that one expects will partition the data-manifold. Then the problem becomes a search through function-space for the "best" function (e.g. makes the fewest errors) using a loss function to determine success for any given candidate-function. Specifically, these decision boundaries are hyperplanes that partition the data, therefore the simplest case would be that of linear classifiers.

Given a bunch of data in $\mathcal{R}^2$, is it possible to find a linear seperating line between the data classes? In the vast majority of cases, the answer is no as much data simple doesn't cluster together, or has cyclic properties to. However, that is not always true. For the sake of this, we're going to introduce an example problem here and track it through the rest of the paper. In particular, this work will follow the linearly seperable gaussian blobs denoted below.

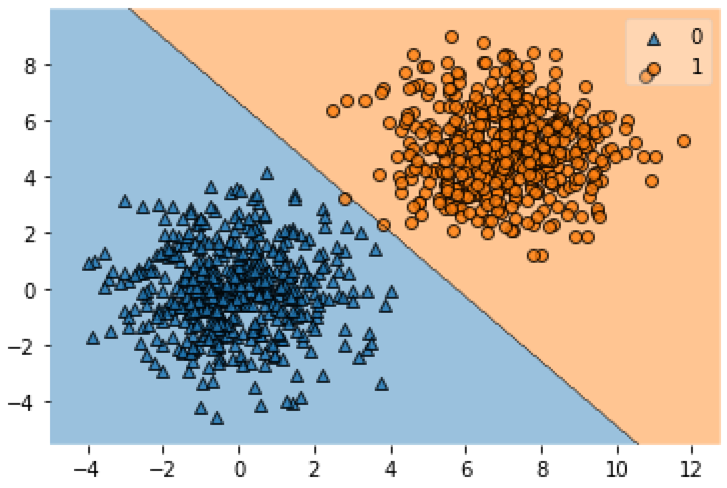


Note, that when points are drawn from a uniform distribution, although they might be initial linearlly seperable, they will quickly become inseperable by a linear classifier. Shattering is finding a hyperplane that places all data-points of a respective class on the same side of the plane. The minimum number of points that can be correctly classified is called the VC-Dimension of a classifier and this area of ML has a rich theorical history. When you pass more than three points in $\mathbb{R}^2$, you are not guaranteed to be able to find a linear plane the can shatter a set.

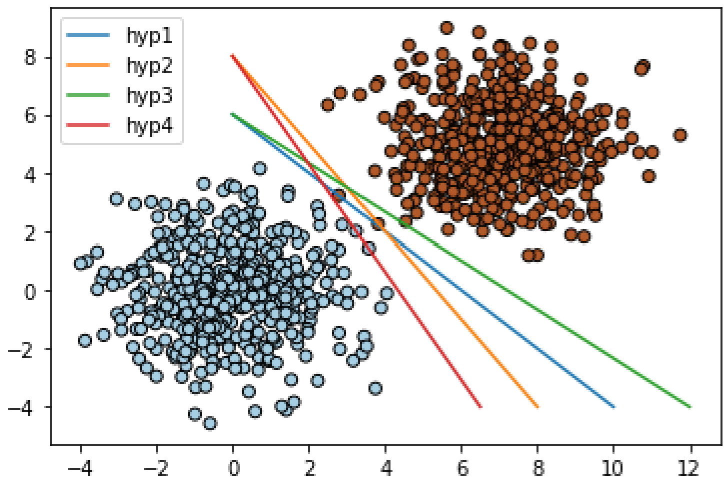


Now it might be reasonable to think that as the number of points sampled goes to infinity, that problems would move towards non-seperable. And that certainly seems true when looking at the uniform distribution. However, if there is some structure inherent in the data, then that is no longer the case. Take for instance that the two data classes are drawn from gaussians. If the gaussians are sufficently far apart (in mean and standard deviation), then we should be able to find a hyerplane seperating them.

Therefore, the question becomes how do we determine a seperating hyperplane for the data, assuming that one exists? Assuming that the data is equally dense between each class, one might conceive of finding 2 centroids of the data to represent the two classes, and then taking the mean of the centroids to get a point around which to rotate a hyperplane. Or one could think about taking linear combinations of the feature variables. The statistics and ML literature is filled with ways of determining hyperplanes. However, they lack a manner of determing a best hyperplane. Below is a **linear** classifier trained on the data. While this clearly does a very good job, the points right at the boundary are missclassified. Therefore good intuition might be, if we can classify the most difficult-to-get-right points, then the remainder will fall out naturally.



For example, the four hyperplanes below, each do a decent job of seperating the two classes of problems, and there might even be one of them that achieves the best accuracy at the job, but it is easy to image different hyperplanes that achieve the same accuracy but are not the same plane. As a concrete example, what makes the red line better than the blue line or vice-versa? You could use the loss function defined above to measure the accuracy of each seperating hyperplane, but let's just say that they both achieve the same accuracy? How would one differentiate between them then? One might read that and think, "okay, that's just a distinction without a difference," but it turns out we can come up with a rigerous way of defining best using Support Vector Machines.



Support Vector Machines are one method of determining a "best" hyperplane. Rather than simply finding one boundary plane, an SVM will find three parallel boundary planes parameterized by one normal vector, w and offest b. Specifically:

$$f_\theta(x) = \begin{cases} w^T x + b = 1 \\ w^T x + b = 0 \\ w^T x + b = -1 \end{cases}$$

Then, the SVM algorithm will continually edit the weights of w and b until the margin defined by the $w^T x + b = +-1$ equation maximize the number of correctly classified data-points as defined using the Loss function (e.g. mean squared error as seen in the intro). These margin equations give us the classification boundaries. Points where $w^T x + b \geq 1$ receive classification of 1 while $w^T x + b \leq -1$ receive classification of -1.

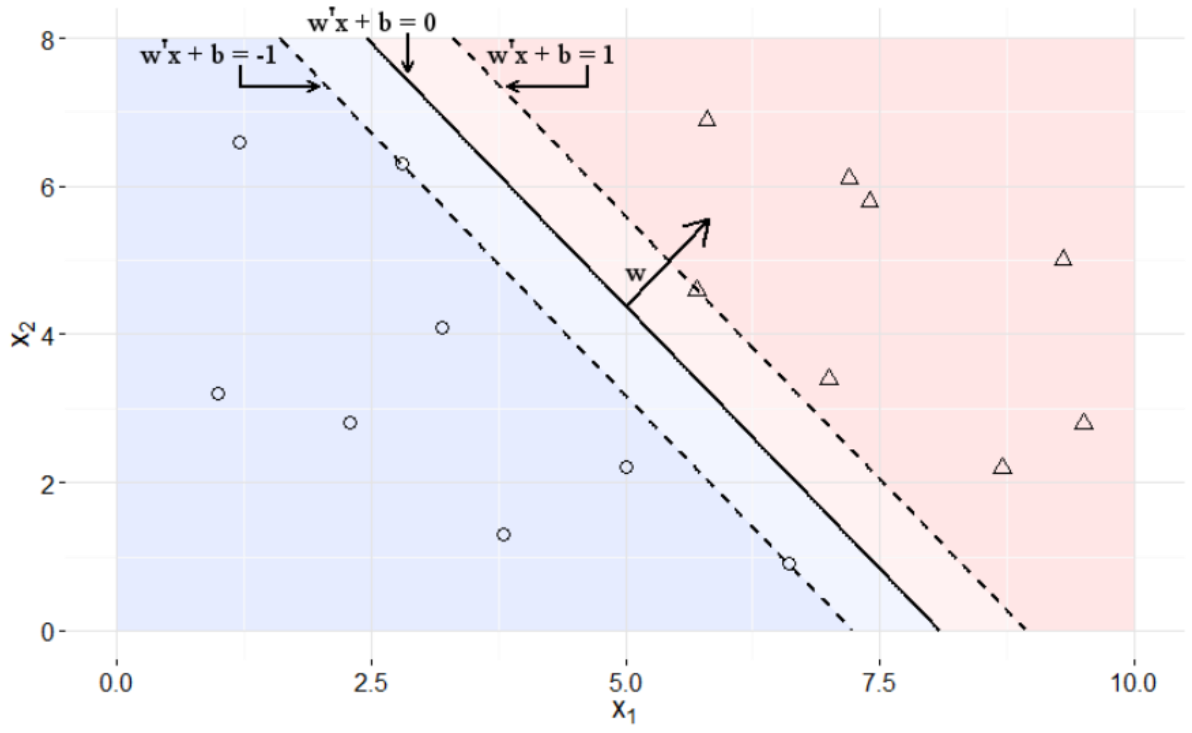


Image citation:
https://storage.googleapis.com/supplemental_media/udacityu/5422370632/Kernel_Methods_and
(https://storage.googleapis.com/supplemental_media/udacityu/5422370632/Kernel_Methods_an

Primal Form

Of course, we haven't yet touched upon how one would actually compute such a set of boundary planes. We can define an SVM using the language of convex optimzation. Specifically, let us begin with defining the primal optimization problem as such:

$$\text{minimize } \frac{||w||^2}{2}$$
$$\text{subject to}$$
$$y_i(x_i \cdot w + b - 1) \geq 0 \forall i$$

We need to verify that the problem is a convex problem. Well, taking a norm is convex, squaring is convex, and scaling by a constant factor is convex. Therefore the composition of these functions is convex. The inequality is a linear inequality, therefore that is convex as well. Therefore, we have met the requirments for defining a Linear Programming problem!

So, let's just say we have labeled data $\{(x_i, y_i) \mid i \in [0, k], k \in \mathbb{N}\}$ where x $\in \mathbb{R}^N$ is the data and y is the class label {+1, -1}. Furthermore, let's say we also have a seperating hyperplane for these points defined by a normal vector w, and an intercept b. Points, q, that lie on the seperating hyperplane, satisfy the equation $w \cdot q + b = 0$.

Let d_- and d_+ be the shortest distance from the sepearting hyperplane to the closest positively (or negatively) labeled point. We can then define the margin as $d_- + d_+$. An SVM therefore looks for the seperating hyperplane with the largest margin. We have control over the placement of this margin by varying the parameters of our normal vector and the intercept. Note that b, is a free variable and there is no constraint as to it being positive or negative.

$$x_i \cdot w + b \geq 1 \forall y_i$$
$$x_i \cdot w + b \leq -1 \forall y_i$$

By adding these two inqualities together you get:

$$y_i(x_i \cdot w + b - 1) \geq 0 \forall y_i$$

Now, consider a point that induces equality in $x_i \cdot w + b \geq 1$. This point, p, lies directly on the hyperplane $x_i \cdot w + b = 1$ where w is normal to the hyperplane. Furthermore, we can determine that the length to the origin from this chosen point is $\frac{|1-b|}{||w||}$. Similarly there is another point, q, that exists on the $x_i \cdot w + b = -1$ hyperplane. This secondary point has distance to the origin $\frac{|-1-b|}{||w||}$. Therefore $d_- = d_+ = \frac{1}{||w||}$ and the margin is therefore $\frac{2}{||w||}$. Therefore we can find a pair of hyperplanes that give a maximum margin by minimizing $\frac{||w||^2}{2}$. This value becomes the Primal problem defined as:

$$\text{minimize } \frac{||w||^2}{2}$$
$$\text{subject to}$$
$$y_i(x_i \cdot w + b - 1) \geq 0 \forall i$$

Dual Form

To get the dual form, we take the primal form and create the Lagrangian of the problem. Creating the Lagrangian involves taking a linear combination of the objective function with weighted constraints. We call these weights lagrange multipliers.

$$\inf_{w,b} \mathcal{L}(x, y, w, b, \alpha) = \frac{||w||^2}{2} - \sum_{i=1}^{l} \alpha_i y_i (x_i \cdot w + b) + \sum_{i=1}^{l} \alpha_i$$

We want to minimize $\mathcal{L}$ with respect to w and b. To do that, we take the derivative of the Langrangian and set it to zero to find the critical points for w and b.

$$\mathcal{L}_w(x, y, w, b, \alpha) = w - \sum_{i=1}^{l} \alpha_i y_i x_i = 0 \rightarrow w^\star = \sum_{i=1}^{l} \alpha_i y_i x_i$$

$$\mathcal{L}_b(x, y, w, b, \alpha) = \sum_{i=1}^{l} \alpha_i y_i \; = 0 \rightarrow b^\star \; = \sum_{i=1}^{l} \alpha_i y_i \; = 0$$

With our constraints defined, we can now plug these back into the Lagrandian for w and b and we get:

$$\mathcal{L}(x, y, \alpha) = -\frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_j y_i x_j^T x_j + \sum_{i=1}^{l} \alpha_i$$

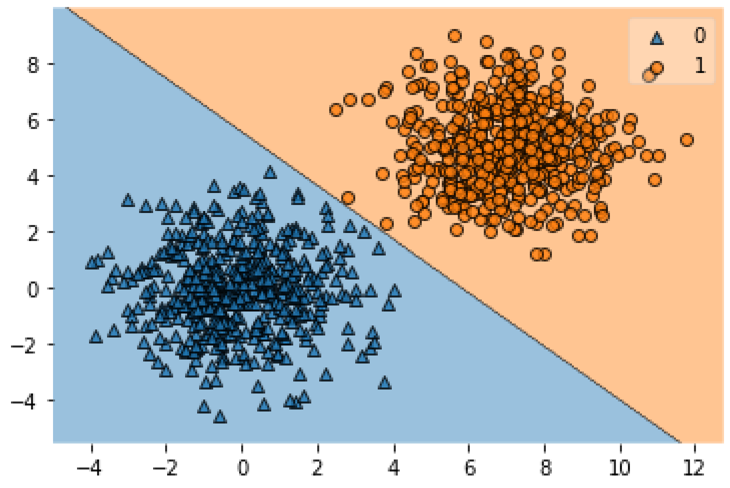
This therefore implies the dual form of:

$$\begin{gathered} \text{minimize } -\frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_j y_i x_j^T x_j + \sum_{i=1}^{l} \alpha_i \\ \text{subject to } \sum_{i} \alpha_i y_i \; = 0 \end{gathered}$$

Note, when I tried switching the minimize to a maximize, CVX threw an error. This will be repeated in the correct section of the derivation once we reach the matlab implementation.

```
Disciplined convex programming error:
   Cannot maximize a(n) convex expression.
```

Using Sci-kit Learn's functionality we can call an implemented version of SVMs to classify out points.

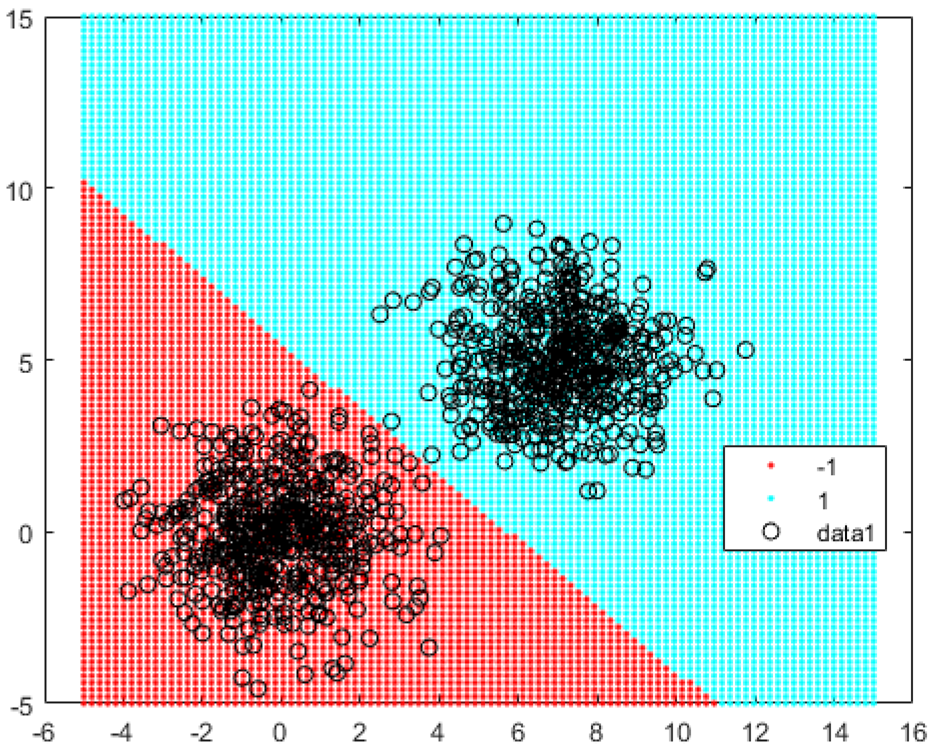


As we can see, the SVM has managed to fix the errors that the linear classifier was unable to get right.

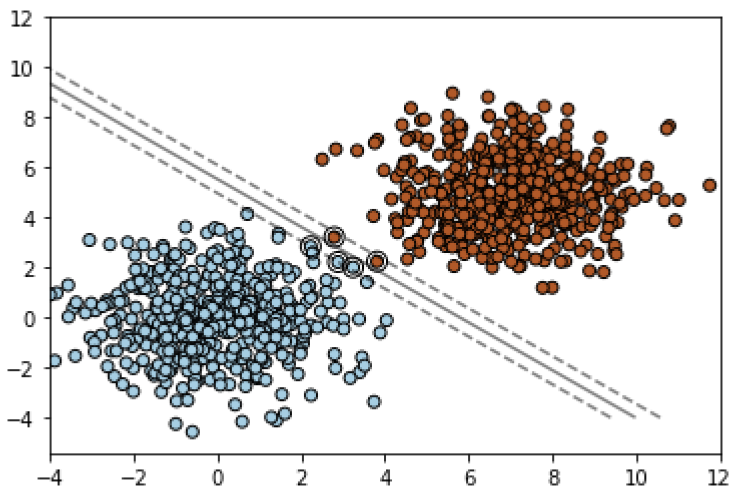
Let us begin with an implementation of the primal form of this optimization problem in matlab's CVX.

```
N = size(data, 1);
M = size(data, 2);

cvx_begin

    variables w(M) b
    minimize 0.5 * sum(square(w))
    subject to
        labels' .* ((data * w) + b) >= 1;

cvx_end
```

This results in the following hyperplane being found:



Next, let us take a moment to look at the boundary but also the support-vectors for this set.



The Dual form looks like this:

```
N = size(pts, 1);
M = size(pts, 2);

K = (labels * labels') .* (pts * pts');
e = ones (N,1); % used to make the sum alpha_i into a dot-product

cvx_begin

    variable a(N)
    dual variables de dp
    minimize -1/2 * transpose(a) * K * a + transpose(e) * a
    subject to
        de : labels * a == 0;
        dp : a >= 0;
cvx_end

% Note, when I tried switching the minimize to a maximize, CVX threw
an error.
% Disciplined convex programming error:
%    Cannot maximize a(n) convex expression.
```

Note that this has been placed into the form of a quadratic program subject to linear constraints:

$$\text{minimize } x^T Q x + \sum_i \alpha_i$$

Now, let us say that we have points that are $almost$ linearly seperable. How might we got about defining hyperplanes for those? For this, let us return to the primal form. In the primal formulation of the SVMs, the algorithm either gets points correct or it does not get them correct. What if we were to instead add in slack variables? So, this would allow us to say: "you almost got this right." For this, let us introduce two new variables, ϵ and C.

First, we need to redefine our hyperplane equations above:

$$x_i \cdot w + b \geq 1 - \epsilon_i \forall y_i \in + \\ x_i \cdot w + b \leq -1 + \epsilon_i \forall y_i \in -$$

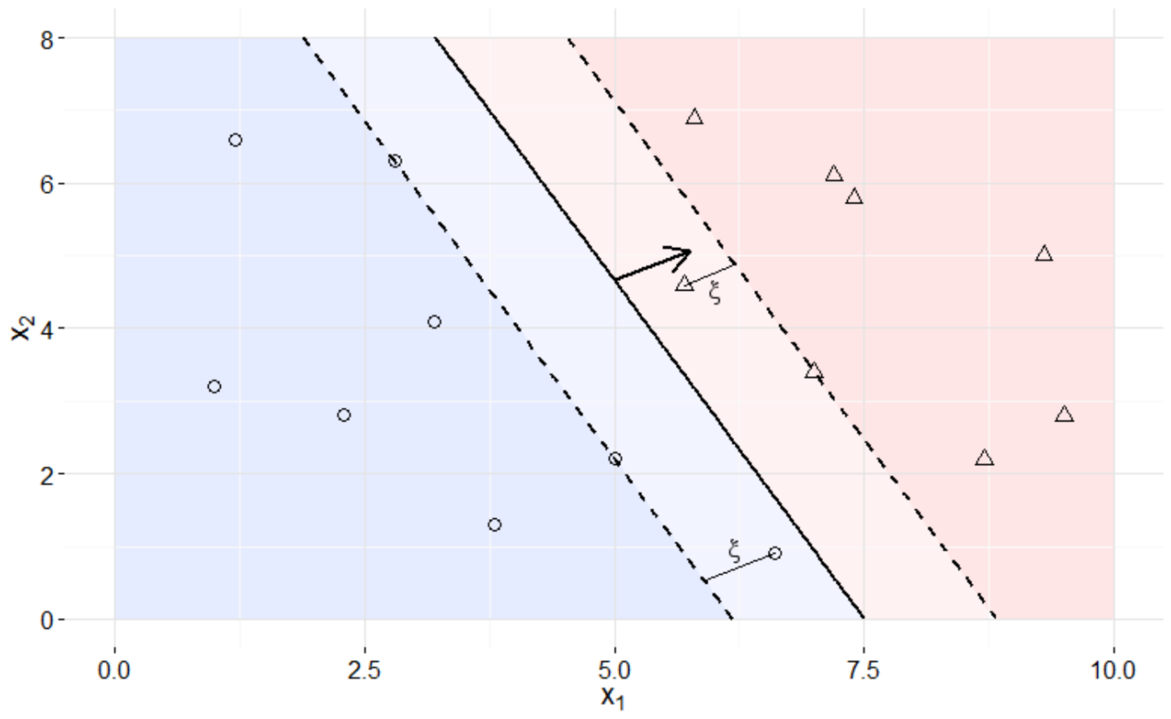


Image citation:
https://storage.googleapis.com/supplemental_media/udacityu/5422370632/Kernel_Methods_and
(https://storage.googleapis.com/supplemental_media/udacityu/5422370632/Kernel_Methods_an

---

When introducing these slack variables into the optimization, we are going to go one step further, and place a hyperparameter C as a weight on their sum. If C is zero, then we recover the original hard-boundary SVM problem forumation. Intuitively, C is going to be our "how much do we want to punish missclassification" parameter.

$$\text{minimize } \frac{||w||^2}{2} - C(\sum_i \epsilon_i)^k \\ \text{subject to} \\ y_i(x_i \cdot w + b - 1) \geq 0$$

When k = 1 or k = 2, this problem is a convex optimization problem. Specifically when k = 1, this is a quadratic programming problem. Furthermore, when k = 1, ϵ_i nor their Langrange multipliers appear in the dual problem. Specifically, the primal Langrangian is:

$$\mathcal{L} = \frac{||w||^2}{2} + C\sum_i \epsilon_i - \sum_i \alpha_i[y_i(x_i \cdot w + b) - 1 + \epsilon_i] - \sum_i \mu\epsilon_i$$

Note, that μ_i are langrange multipliers that enforce positivity on ϵ. Taking derivaties of w and b again, when k = 1, will result in the original problem coming back as our dual optimization problem, but now with a limit on the values α_i can take on.

$$\text{minimize } -\frac{1}{2}\sum_{i,j=1}^{l} \alpha_i\alpha_j y_j y_i x_j^T x_j + \sum_{i=1}^{l}\alpha_i$$
$$\text{subject to } \sum_i \alpha_i y_i = 0$$
$$0 \leq \alpha_i \leq C$$

Quick Aside:

Given data in the form of {$(x_i,\ y_i)$ | $i \in [0, k], k \in \mathbb{N}$}, to get more flexible kernels into this framework, we can move away from the dot product of: $x \cdot x$ and try more expressive combinations of the data. The only constraint is that we need to be able to take the kernel by taking dot products of the data. Other standard kernels are:

$$K(x_i, x_j) = (x_i \cdot x_j + 1)^P \leftarrow \text{polynomial kernel}$$
$$K(x_i, x_j) = e^{\frac{-||x_i - x_j||^2}{2\sigma^2}} \leftarrow \text{gaussian kernel}$$

Then one just replaces everywhere in the problem forumation of $x_i \cdot x_j$ with $K(x_i, x_j)$

Primary Citations:

```
- https://storage.googleapis.com/supplemental_media/udacityu/5422370
632/Kernel_Methods_and_SVMs.pdf
- https://www.di.ens.fr/~mallat/papiers/svmtotirial.pdf
- Boyd's Convex Optimization
```