MATH 565: Lecture 1 (01/13/2026)

Today: * logistics, syllabus, ...
* problems in ML, optimization for them

This is Optimization for Machine Learning (Math 565)

I'm Bala Krishnamoorthy (call me Bala).

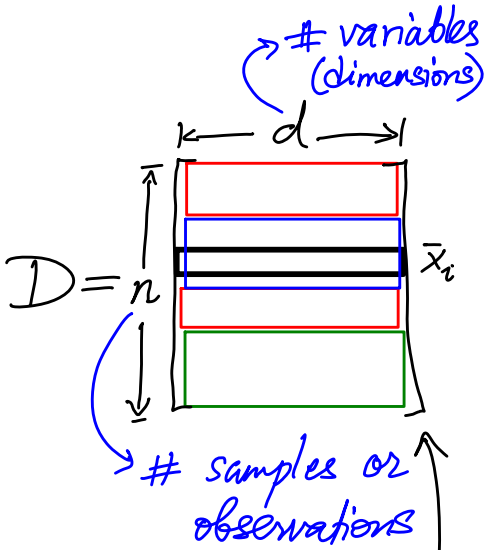
We will **not** use Canvas. All materials for the class will be posted on the course web page: https://bala-krishnamoorthy.github.io/Math565.html

Download the book PDF (accessible via WSU Libraries)!

* Check the syllabus on the course web page.
* homework assignments will be posted at least a week before its due.

Machine Learning Problems

1. **Clustering** We are given a data matrix that is $n \times d$ for n observations (or samples) each having values for d variables (dimension=d). The i^{th} observation is denoted $\bar{x}_i$ (d-vector), which forms the i^{th} row of D (when written as $\bar{x}_i^T$).

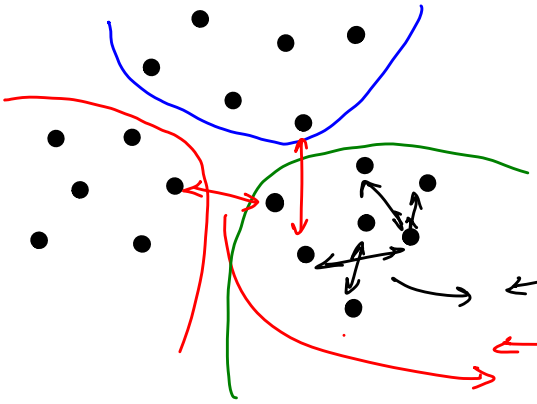


The goal of clustering is to divide the n observations into k disjoint subsets or clusters. The # clusters k may be user-defined, e.g., in k-means clustering, or may be determined as part of the clustering method. Equivalently, the rows of D are to be divided into k disjoint sets (e.g., into red, blue, and green clusters as illustrated here).

Notation: A, X, Y: matrices or sets (uppercase letters)
$\bar{x}, \bar{y}, \bar{\alpha}, \bar{\theta}$: vectors (lower case letters with a bar)
x, β, γ, a: scalars (lower case letters)

Another ("metric") way to think about clustering is illustrated in 2D.

$\bullet \bar{x}_i$



division into 3 clusters (red, blue, green)

$\longleftrightarrow$: within cluster distances
$\longleftrightarrow$: distance to nearest point outside cluster

One objective function used in clustering is to minimize the sum of all within cluster pairwise distances while also maximizing the sum of distances of each point to the nearest point outside its cluster.

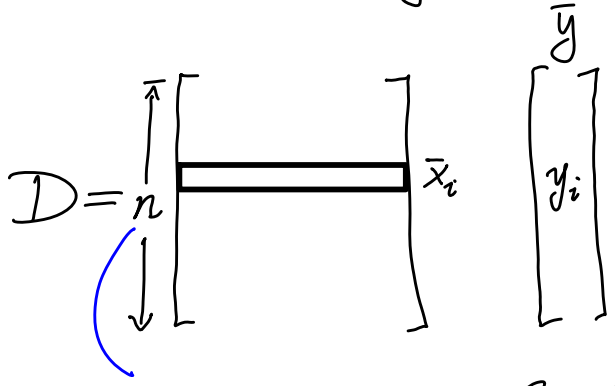
An example: Each $\bar{x}_i$ represents a consumer, and the entries are the $ amounts they spent on fruits, meat, toiletteries, or other items (in a retail or grocery store). The store may be interested in the clusters to target ads for specific products.

Clustering is considered an unsupervised learning problem, since only the data ($\bar{x}_i$) is used without the membership information (of which cluster each $\bar{x}_i$ belongs to).

As opposed to classification/regression problems being introduced now, which are supervised learning problems.

2. Classification

Here, apart from the $n \times d$ data matrix D, we are given also an n-vector $\bar{y}$ of labels for each observation.



For instance, $y_i \in \{-1, 1\}$ (or $\{0, 1\}$) to capture a YES/NO aspect of each observation, e.g., whether customer i is possibly responsive to ads. In another instance, $y_i \in \{R, G, B\}$ (3 colors). Note that such labels are categorical, and it is not obvious how to represent these labels using numerical values. The YES/NO aspect could still be modeled as a continuous numerical value coupled with a cut-off value (to determine YES).

The goal is to build a "model" on D and $\bar{y}$, which is the training set, and use that model to predict the $\bar{y}_t$ values for an external test data set D_t ($n_t \times d$) for which $\bar{y}_t$ is not known.

The model can be described as

$$y_i \approx f(\bar{x}_i).$$

The function to be learned, f, is often parametrized using a weight vector $\bar{w}$, and hence we write

$$y_i \approx f_{\bar{w}}(\bar{x}_i)$$

For instance, in the case of binary classification with $y_i \in \{-1, 1\}$, we could take

$$y_i = \text{sign}\{f_{\bar{w}}(\bar{x}_i)\}$$

Q. How does one choose $\bar{w}$?

We usually pick $\bar{w}$ such that the mismatch between y_i and $f_{\bar{w}}(\bar{x}_i)$, i.e., the "loss", is minimized. The form of f and this loss function are often carefully chosen/constructed to solve

$$\min_{\bar{w}} \sum_{i=1}^{n} \ell\left(y_i - f_{\bar{w}}(\bar{x}_i)\right)$$

ℓ → loss function

Such classification tasks are considered supervised learning, since we are guided by the y_i's (known labels). The goal is to use the built function $f_{\bar{w}}(\bar{x})$ to predict the labels $\bar{y}_t$ for an external or test data set D_t $(n_t \times d)$ (for which the labels are unknown). Since D_t is not used in the training process, such predictions for test sets are called as generalizations.

The step where we want to minimize a loss function makes classification an optimization problem! Before considering details of optimization, we present a simpler ML problem: regression.

3. Regression

We're given D $(n \times d)$ just as in classification, but now, $y_i \in \mathbb{R}$, i.e., it is a numerical value (rather than a label).

The simplest version of regression is linear regression, where the task is to fit a line through the given set of n points (in 2D, and a plane in d-dimensions).

$$y_i = f_{\bar{w}}(\bar{x}_i) = \bar{w}^T \bar{x}_i = \bar{x}_i^T \bar{w}$$

Equivalently, $\bar{y} = D\bar{w}$.

And the loss function usually used is the sum of squared errors:

$$J = \frac{1}{2} \|D\bar{w} - \bar{y}\|^2.$$

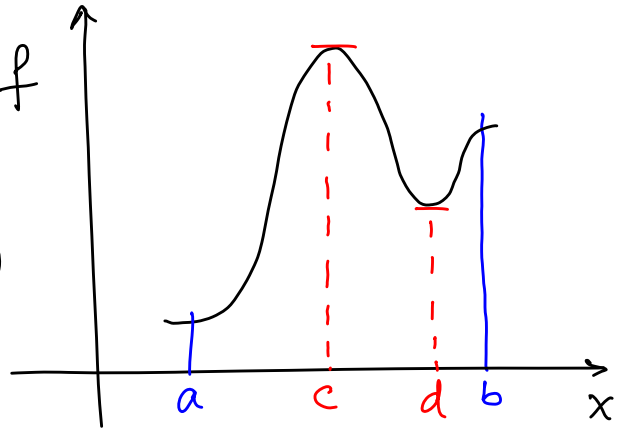
Hence, linear regression becomes the optimization problem:

$$\min_{\bar{w}} \frac{1}{2} \|D\bar{w} - \bar{y}\|.$$

Optimization in 1D Calculus

Find $\min f(x)$ for $a \le x \le b$

* We set $f'(x) = 0$ to find critical points.
* $f''(x) > 0$: local minima (e.g., d)
  $f''(x) < 0$: local maxima (e.g., c)
  $f''(x) = 0$: saddle points



Compare the function values at local minima with those at the end points (a and b) to determine the true ("global") minimum (@ $x = a$ in the figure here).

Optimization in d-dimensions

In general, we consider problems of the form

$$\begin{aligned} &\min f(\bar{x}) \\ &\text{s.t.} \quad \bar{g}(\bar{x}) \leq \bar{0} \\ &\quad\;\; \bar{h}(\bar{x}) = \bar{0} \end{aligned}$$

Under appropriate assumptions, we can specify optimality conditions (local by default, global when the functions are "nice"), e.g., Karush-Kuhn-Tucker (KKT) conditions.

Corresponding to $f'(x)=0$, we have $\nabla f = \bar{0}$ for

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{bmatrix}, \text{ the gradient of } f.$$

Back to linear regression:

Recall: $\min_{\bar{w}} J = \frac{1}{2} \| D\bar{w} - \bar{y} \|^2$

$$\Rightarrow \nabla J = D^T D \bar{w} - D^T \bar{y} = \bar{0}$$

$$\Rightarrow \bar{w} = (D^T D)^{-1} D^T \bar{y}$$

closed form expression!

It turns out that this critical point is the (unique) global minimizer of J. In this sense, linear regression is perhaps the easiest ML problem!

But more generally, we do not get closed form solutions in this form. In fact, we usually use gradient descent to iteratively update $\bar{w}$:

We choose an initial $\bar{w}$ in some way, e.g., randomly. Then, in each iteration, we update

$$\bar{w} \leftarrow \bar{w} - \alpha \nabla J(\bar{w})$$

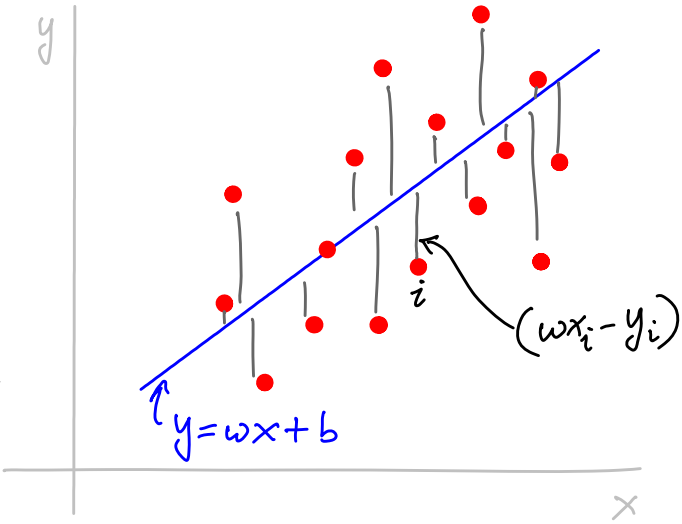
α is the step size, also called the learning rate. We usually need to choose α carefully to ensure the iterations converge. We also have to make assumptions about the function f to guarantee efficient convergence of this gradient-descent method.

MATH 565: Lecture 2 (01/15/2026)

Today:
* regression and extensions
* Regularization
* Support vector machines (SVM)

Recall: Linear regression: Given $D_{n\times d}$, $\bar{y}\in\mathbb{R}^n$, the goal is to find weight vector $\bar{w}\in\mathbb{R}^d$ such that $\bar{y}\approx D\bar{w}$. In detail, we want to minimize the loss function

$$J = \frac{1}{2}\|D\bar{w}-\bar{y}\|^2 = \frac{1}{2}\sum_{i=1}^{n}(y_i-\bar{w}^T\bar{x}_i)^2$$



In 1D, linear regression fits a line $y=wx+b$ through a given set of n points (x_i,y_i) such that the sum of the squared "errors" $\sum_{i=1}^{n}(wx_i-y_i)^2$ is minimized.

J is a convex function and hence has a unique minimum. The first order optimality condition (corresponding to $f'(x)=0$ in 1D) is given by $\nabla J=\bar{0}$

$$J=\frac{1}{2}(D\bar{w}-\bar{y})^T(D\bar{w}-\bar{y}) = \frac{1}{2}\left[\bar{w}^TD^TD\bar{w}-2\bar{w}^TD^T\bar{y}+\bar{y}^T\bar{y}\right]$$

So, $\nabla J = D^TD\bar{w}-D^T\bar{y}=0$

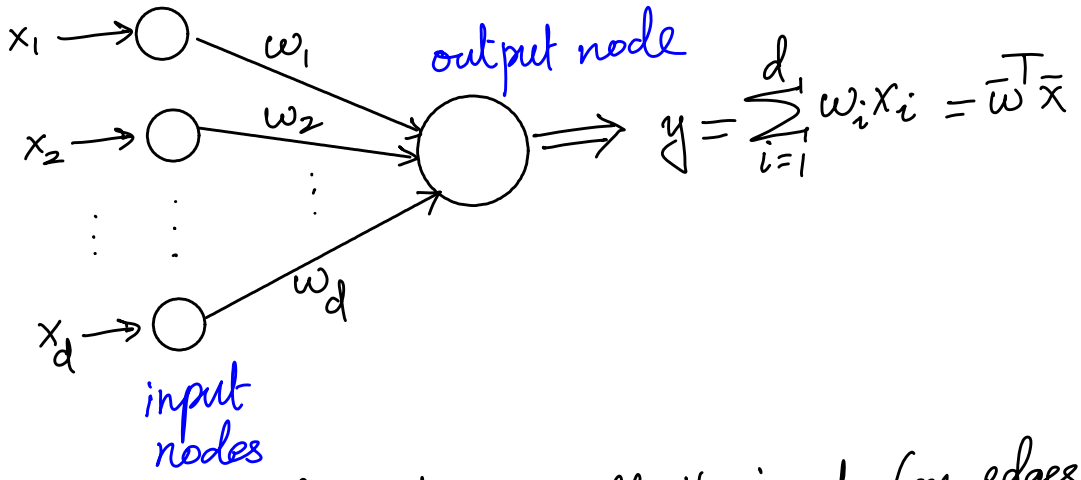
$$\Rightarrow \quad D^TD\bar{w}=D^T\bar{y} \qquad (1)$$

$$\Rightarrow \quad \bar{w}=(D^TD)^{-1}D^T\bar{y} \qquad (2)$$

A Quick aside: Optimization on Computational Graphs

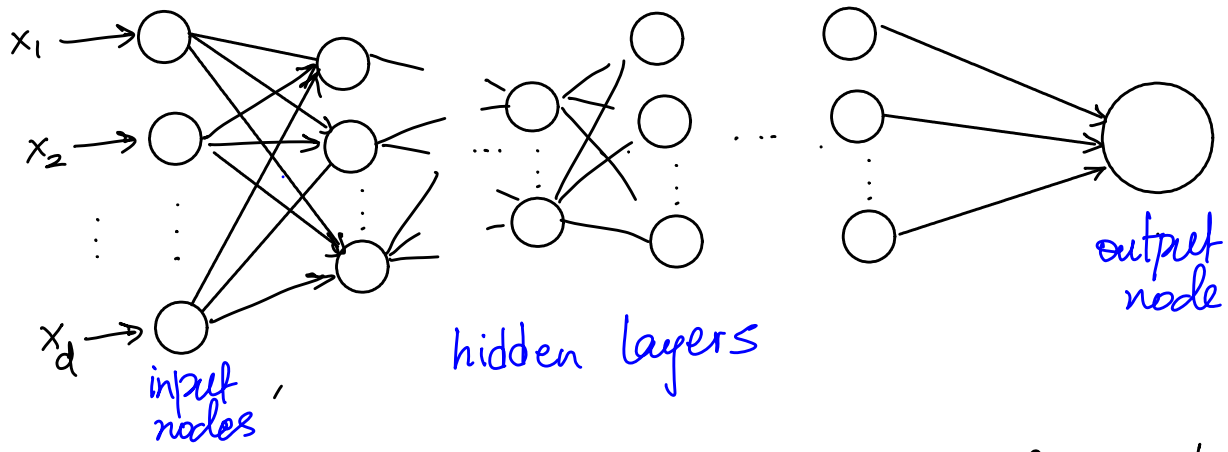
Many machine learning problems can be equivalently posed as problems on computational graphs. Informally, the qualifier "computational" here means the vertices and edges in these graphs are endowed with more general computational steps than, e.g., the weights/costs and edge capacities seen in network flow problems. While we will revisit this topic in detail later on, we give a graph that can be used to model linear regression.

Linear regression graph model



In general, each node takes in all its inputs (on edges coming in to it) and combines them somehow to send output(s) to all nodes to which it is connected. For linear regression, there is one input node for each x_i that uses the weight w_i on its output edge to send $w_i x_i$ to the single output node. And this output node sums up all its inputs to output $\sum_i w_i x_i = \vec{w}^T \vec{x}$.

But the computational graphs could be far more general, with multiple layers of nodes and edges connecting nodes in each layer to many or all nodes in nearby (not necessarily only adjacent) layers.



The functions and data captured on the nodes and edges can also be quite general. As one can imagine, such graphs could capture fairly complicated functions and restrictions. How does one take gradients on computational graphs? We use back propagation. More on this later...

Back to Regression...

Solving a system of linear equations can be considered as a version of linear regression. If there is a solution, then we get $J=0$ (zero loss). But if there is no (exact) solution, we could still get a "best fit" solution (in the least squares sense).

Recall how we found $\bar{w}$:

$$D^T D \bar{w} = D^T \bar{y} \quad \text{——— (1)}$$

$$\bar{w} = (D^T D)^{-1} D^T \bar{y} \quad \text{——— (2)}$$

How do we compute $\bar{w}$ efficiently (for large instances)? Computing $(D^T D)^{-1}$ can be costly, especially when n and d are huge (or even large). We usually use QR decomposition of the data matrix D.

$$D = QR \quad \text{——— (3)}$$

where Q is $n \times d$ with orthonormal columns and R is $d \times d$ upper triangular.

Hence, $Q^T Q = I_d$ (identity matrix).

Using (3) in (1) gives

$$R^T \underbrace{Q^T Q}_{I} R\bar{w} = R^T Q \bar{y}$$

$$\Rightarrow (R^T)^{-1}\left(R^T R\bar{w} = R^T Q\bar{y}\right) \quad \text{——— (4)}$$

$$\Rightarrow R\bar{w} = Q^T \bar{y}$$

This is a triangular system and can be solved by back substitution.
↳ as R is upper triangular

This computation assumes that $D^T D$ is invertible, which may not always hold — e.g., when $n < d$ (we do not have enough samples for the given number of dimensions). This is similar to the setting in which a linear system $A\bar{x} = \bar{b}$ has infinitely many solutions. In this setting, we run the danger of overfitting the (training) data. And when that happens, the trained model may not generalize as well to external test data.

To resolve this situation, we need to do regularization. Intuitively, we need to limit the # w_j's that are non-zero ...

Tikhonov Regularization

We modify the loss function to

$$J_W = \frac{1}{2}\|D\bar{w} - \bar{y}\|^2 + \frac{\lambda}{2}\|\bar{w}\|^2$$

for the regularization parameter $\lambda > 0$.

Our goal is to somehow enforce several or many of the $w_i = 0$ (i.e., only a few of them are non-zero). One approach to achieve this goal is to add an explicit constraint of the form

$\|\bar{w}\|_0 \leq k$ for $k << d$.

(0-norm (# non-zero entries))

But this constraint is hard to enforce, and also destroy the nice structure of J (convexity). Instead, we add the squared norm penalty $\frac{\lambda}{2}\|\bar{w}\|^2$ as a regularization term. It achieves the same goal — it forces many $w_i = 0$ in the optimal solution. But it is also a strongly convex function ($\lambda > 0$), and hence J_W is so as well ($\frac{1}{2}\|D\bar{w} - \bar{y}\|^2$ term is convex), implying that J_W has a unique optimal solution.

($w_i \neq 0$ unless absolutely needed.)

Setting $\nabla J_W = \bar{0}$ for optimality gives

$$(D^T D + \lambda I)\bar{w} = D^T \bar{y}$$

$$\Rightarrow \bar{w} = (D^T D + \lambda I)^{-1} D^T \bar{y}$$

We'll talk about more details of this regularized model later on...

Modifications of J_w

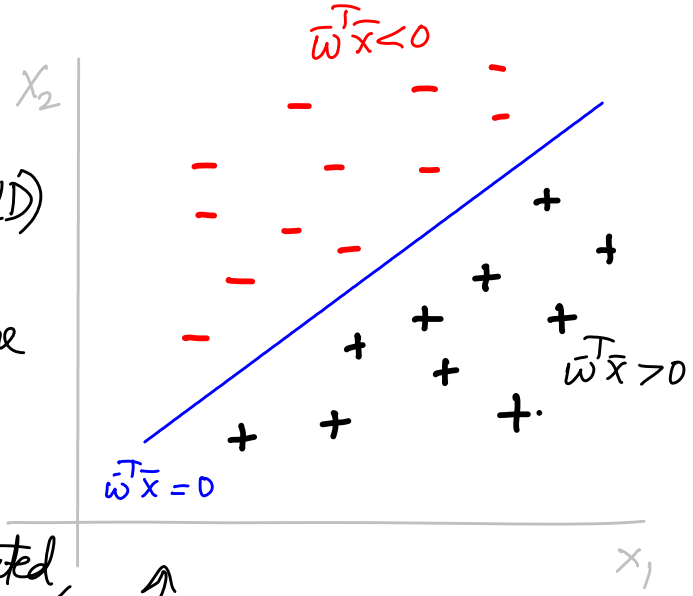
In $J_w = \frac{1}{2}\|D\bar{w} - \bar{y}\|^{\frac{2}{2}} + \frac{1}{2}\lambda\|\bar{w}\|^{\frac{2}{2}}$, also called "fit"

one could consider modifying either (loss/error or regularization) term to an L_1-norm term (instead of L_2). There are advantages and disadvantages for using L_1 terms. L_1-norms are piecewise linear, and when they appear in minimization objective functions as here, they can be easily linearized, making computations more efficient. At the same time, the L_2-terms usually have smoother behavior than their L_1-counterparts.

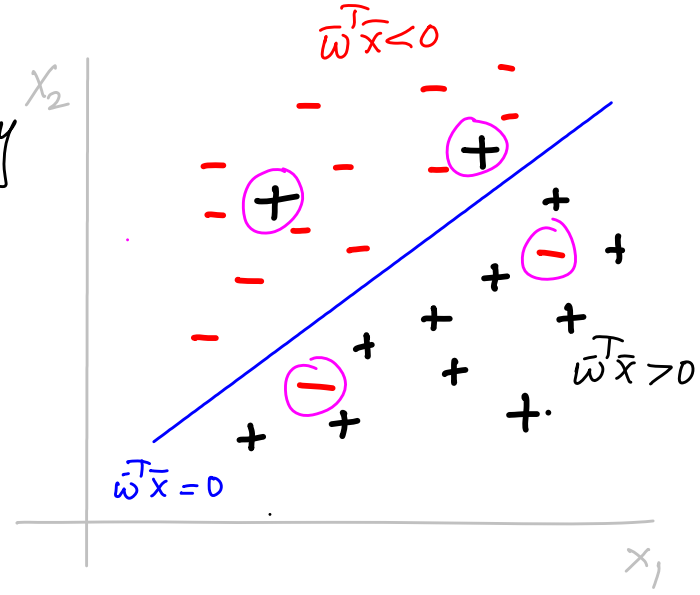
Binary Classification

We now consider the binary classification problem in a bit more detail in the context of regression. Here, we have $y_i \in \{-1, 1\}$, and the goal is to "separate" the $+1$ instances "as best as possible" from the -1 instances.

A direct generalization of linear regression is to find $\bar{w}$ s.t. $\bar{w}^T\bar{x} = 0$ is the separating line (in 2D) or hyperplane (in d-dim), with the $\bar{w}^T\bar{x} > 0$ side capturing all the $+1$ instances and the other side containing all the -1 instances. This may work well in well-separated, i.e., easy, instances as illustrated here.



But when the +1 and −1 instances are not well-separated (as seen here), we would want to modify J_w (loss function) to capture the violations of good separations. Here is one approach. We write



$$J_w = \frac{1}{2}\sum_{i=1}^{n}(y_i - \bar{w}^T\bar{x}_i)^2 + \frac{\lambda}{2}\|\bar{w}\|^2.$$

Using the fact that $y_i^2 = 1$ (as $y_i \in \{-1, 1\}$), we write

$$J_w = \frac{1}{2}\sum_{i=1}^{n} y_i^2(y_i - \bar{w}^T\bar{x}_i)^2 + \frac{\lambda}{2}\|\bar{w}\|^2$$

$$= \frac{1}{2}\sum_{i=1}^{n}(y_i^2 - y_i\bar{w}^T\bar{x}_i)^2 + \frac{\lambda}{2}\|\bar{w}\|^2$$

$$= \frac{1}{2}\sum_{i=1}^{n}(1 - y_i\bar{w}^T\bar{x}_i)^2 + \frac{\lambda}{2}\|\bar{w}\|^2$$

(again, using $y_i^2 = 1$).

The first term will penalize violations of good separation. At the same time, it will also penalize cases that are "extremely" well-separated! For instance, if $y_i = +1$ and $\bar{w}^T\bar{x}_i = 100$, which is a very good separation, the term in J_w will be $(1 - 1\cdot 100)^2 = (99)^2 = 9801$, which is huge!

Hence, we want to set the contribution of well-separated instances (to J_w) as zero — this is the idea used in support vector machines (SVMs)!

Support Vector Machines (SVM)

We set

$$J_{L_2\text{-svm}} = \frac{1}{2}\sum_{i=1}^{n}\left(\max\left\{0, \left[1 - y_i(\bar{w}^T\bar{x}_i)\right]\right\}\right)^2 + \frac{1}{2}\|\bar{w}\|^2$$

A (computationally) better option is to consider an L_1-SVM loss. We rewrite this L_1-SVM problem as a slightly modified optimization problem, as presented below. For many people working in optimization, this SVM model is a natural way to start exploring machine learning problems, as it is a convex optimization problem.

$$\min_{\bar{w}, \bar{\xi}, b} \; J_{SVM} = C\sum_{i=1}^{n}\bar{\xi}_i + \frac{1}{2}\|\bar{w}\|^2$$

$$\text{s.t.} \quad y_i\left(\bar{w}^T\bar{x}_i + b\right) \geq 1 - \bar{\xi}_i, \quad i = 1, \ldots, n$$

$$\bar{\xi}_i \geq 0$$

We will check the details of this model in the next lecture...