CS-UY 4563: Lecture 14
Support Vector Machines

NYU Tandon School of Engineering, Prof. Christopher Musco

- Project topic/teams due on **Wednesday** via email.
    - Sign up for a meeting time after you send me the email.
- Lab `lab_grad_descent_partial.ipynb` due **Thursday night**.
    - We don't have enough time to do the topic of optimization justice, so take my class next semester if you want to learn more.

How to use **non-linear kernels** with logistic regression.

- Often leads to better classification than basic linear logistic regression.
- Equivalent to feature transformation, but often computationally faster.

Commonly used positive semidefinite (PSD) kernel functions:

- Linear (inner-product) kernel: $k(\vec{x}, \vec{y}) = \langle \vec{x}, \vec{y} \rangle$
- Gaussian RBF Kernel: $k(\vec{x}, \vec{y}) = e^{-\|\vec{x}-\vec{y}\|_2^2/\sigma^2}$
- Laplace Kernel: $k(\vec{x}, \vec{y}) = e^{-\|\vec{x}-\vec{y}\|_2/\sigma}$
- Polynomial Kernel: $k(\vec{x}, \vec{y}) = (\langle \vec{x}, \vec{y} \rangle + 1)^q$.

Recall: Every PSD kernel has a corresponding feature transformation $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^m$.

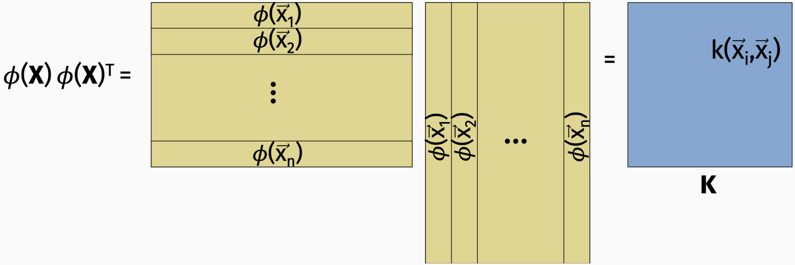
$$k(\vec{x}, \vec{y}) = \phi(\vec{x})^T \phi(\vec{y}))$$

Sometimes $\phi(\vec{x})$ is simple and explicit. **More often, it is not.**

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \phi(\vec{x}) = \begin{bmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \sqrt{2}x_3 \\ x_1^2 \\ x_2^2 \\ x_3^2 \\ \sqrt{2}x_1x_2 \\ \sqrt{2}x_1x_3 \\ \sqrt{2}x_2x_3 \end{bmatrix}$$

Degree 2 polynomial kernel, $k(\vec{x}, \vec{w}) = (\vec{x}^T\vec{w} + 1)^2$.

Typically doesn't matter because we only need to compute the kernel Gram matrix $\mathbf{K}$ to retrofit algorithms like logistic or linear regression to use non-linear kernels.



(If this stuff interests you, understanding the kernel feature maps ϕ which correspond to different kernels is a large part of my current research. This understanding can lead to faster kernel methods.)

Support Vector Machines (SVMs): Another algorithm for finding linear classifiers which is as popular as logistic regression.

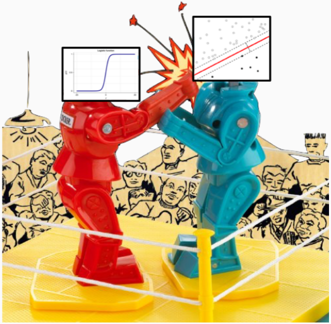
- Can also be combined with kernels.
- Developed from a pretty different perspective.
- But final algorithm is not that different.

- Invented in 1963 by Alexey Chervonenkis and Vladimir Vapnik. Also founders of VC-theory.
- First combined with non-linear kernels in 1993.

For some reason, SVMs are more commonly used with non-linear kernels. For example, `sklearn`'s SVM classifier (called SVC) has support for non-linear kernels built in by default. Its logistic regression classifier does not.

- I believe this is <u>mostly</u> for historical reasons and connections to theoretical machine learning.
- In the early 2000s SVMs where a "hot topic" in machine learning and their popularity persists.
- It is not clear to me if they are better than logistic regression, but honestly I'm not sure...

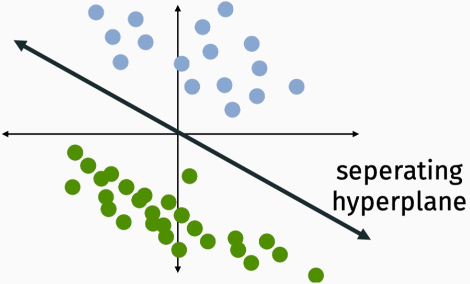
Next lab: `lab_mnist_partial.ipynb`.



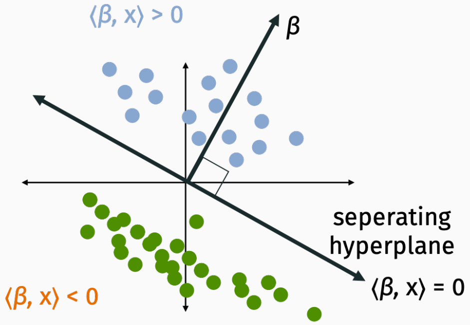
Machina-a-machina comparison of SVMs vs. logistic regression for a MNIST digit classification problem. Which provides better accuracy? Which is faster to train?

20% extra credit on lab if you can beat my simple baseline.

We call a dataset with binary labels <u>linearly separable</u> if it can be perfectly classified with a linear classifier:

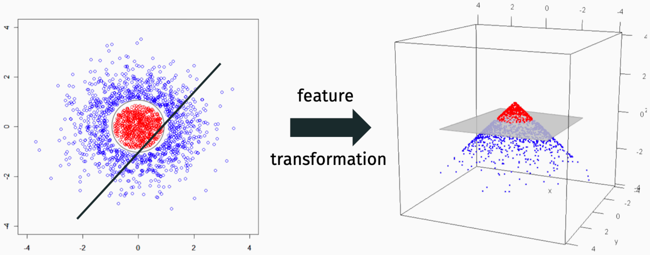


Formally, there exists a parameter $\vec{\beta}$ such that $\langle \vec{\beta}, \vec{x} \rangle > 0$ for all $\vec{x}$ in class 1 and $\langle \vec{\beta}, \vec{x} \rangle < 0$ for all $\vec{x}$ in class 0.



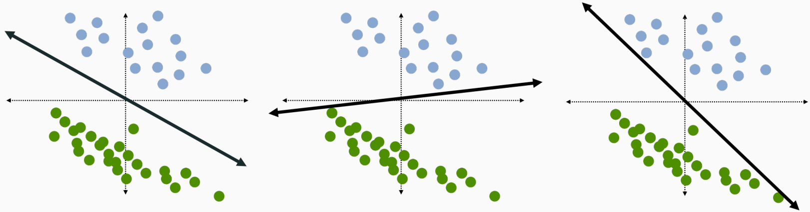
Note that if we multiply $\vec{\beta}$ by any constant c, $c\vec{\beta}$ gives the same separating hyperplane because $\langle c\vec{\beta}, \vec{x} \rangle = c\langle \vec{\beta}, \vec{x} \rangle$.

A data set might be linearly separable when using a non-kernel/feature transformation even if it is not separable in the original space.



This data is separable when using a degree-2 polynomial kernel. If suffices for $\phi(\vec{x})$ to contain x_1^2 and x_2^2.

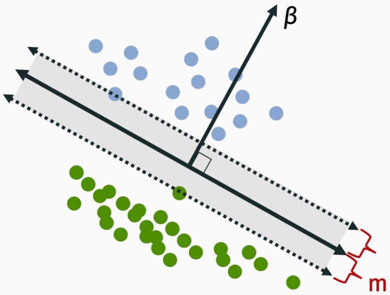
When data is linearly separable, there are typically multiple valid separating hyperplanes.



Which hyperplane/classification rule is best?

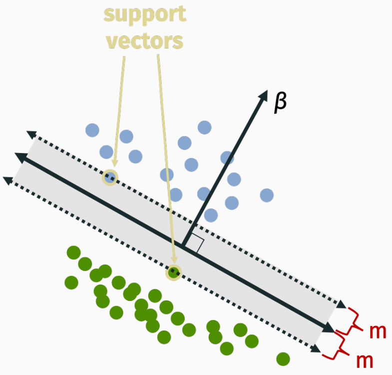
MARGIN

The **margin** m of a separating hyperplane is the <u>minimum</u> ℓ_2 (Euclidean) distance between a point in the dataset and the hyperplane.

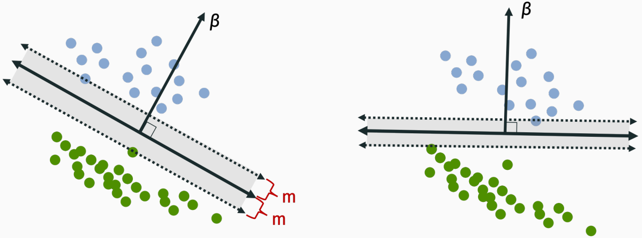


$$m = \min_i \Delta_i \qquad \text{where} \qquad \Delta_i = \frac{|\langle \vec{x}_i, \vec{\beta} \rangle|}{\|\vec{\beta}\|_2}$$

A **support vector** is any data point $\vec{x}_i$ such that $\frac{|\langle \vec{x}_i, \vec{\beta} \rangle|}{\|\vec{\beta}\|_2} = m$.



A <u>hard-margin</u> support vector machine (SVM) classifier finds the **maximum margin (MM) linear classifier**.



I.e. the separating hyperplane which maximizes the margin m.

Denote the maximum margin by m^*.

$$m^* = \max_{\vec{\beta}} \left[\min_{i \in 1,\ldots,n} \frac{|\langle|\vec{x}_i, \beta\rangle|}{\|\vec{\beta}\|_2} \right]$$
$$= \max_{\vec{\beta}} \left[\min_{i \in 1,\ldots,n} \frac{y_i \cdot \langle \vec{x}_i, \vec{\beta}\rangle}{\|\vec{\beta}\|_2} \right]$$

where $y_i = -1, 1$ depending on what class $\vec{x}_i$.[1]

[1] Note that this is a different convention than the 0, 1 class labels we typically use.

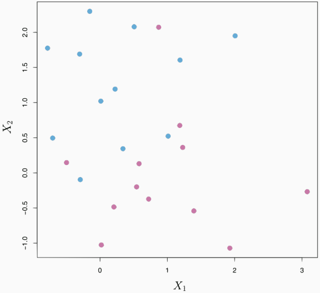
Equivalent formulation:

$$\min_{\vec{\beta}} \|\vec{\beta}\|_2^2 \qquad \text{subject to} \qquad y_i \cdot \langle \vec{x}_i, \vec{\beta} \rangle \geq 1 \text{ for all } i.$$

Under this formulation $m = \frac{1}{\|\beta\|_2}$.

This is a **constrained optimization problem.** In particular, a linearly constrained quadratic program, which is a type of problem we have efficient optimization algorithms for.

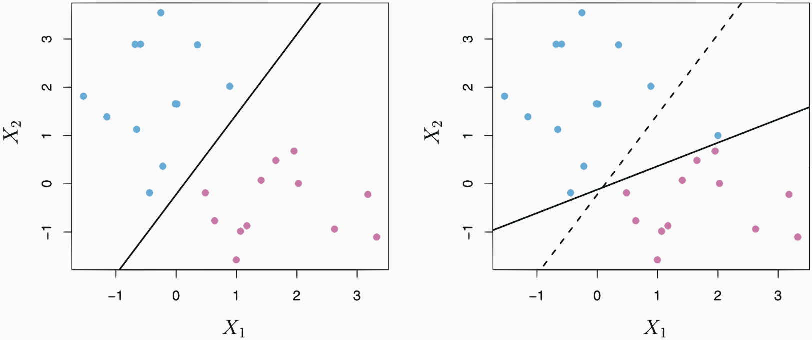
While important in theory, hard-margin SVMs have a few critical issues in practice:



Data might not be linearly separable, in-which case the maximum margin classifier is not even defined.

Less likely to be an issue when using a non-linear kernel. If K is full rank then perfect separation is always possible. And typically it is, e.g. for an RBF kernel or moderate degree polynomial kernel.

While important in theory, hard-margin SVMs have a few critical issues in practice:



Hard-margin SVM classifiers are not robust.

Solution: Allow the classifier to make some mistakes!

Hard margin objective:

$$\min_{\vec{\beta}} \|\vec{\beta}\|_2^2 \qquad \text{subject to} \qquad y_i \cdot \langle \vec{x}_i, \vec{\beta} \rangle \geq 1 \text{ for all } i.$$

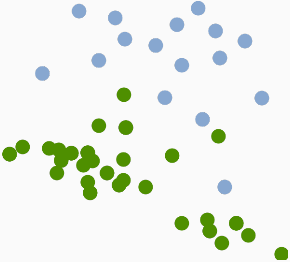
Soft margin objective:

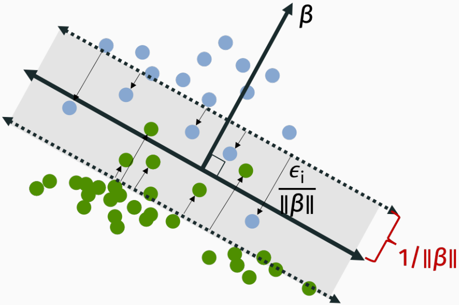
$$\min_{\vec{\beta}} \|\vec{\beta}\|_2^2 + C \sum_{i=1}^{n} \epsilon_i \quad \text{subject to} \quad y_i \cdot \langle \vec{x}_i, \vec{\beta} \rangle \geq 1 - \epsilon_i \text{ for all } i.$$

where $\epsilon_i \geq 0$ is a non-negative "slack variable". This is the magnitude of the "error" made on example $\vec{x}_i$.

$C \geq 0$ is a non-negative tuning parameter.

Example of a non-separable problem:

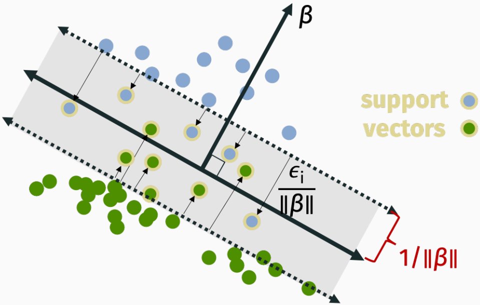




Soft margin objective:

$$\min_{\vec{\beta}} \|\vec{\beta}\|_2^2 + C\sum_{i=1}^{n} \epsilon_i \quad \text{subject to} \quad y_i \cdot \langle \vec{x}_i, \vec{\beta} \rangle \geq 1 - \epsilon_i \text{ for all } i.$$

SOFT-MARGIN SVM



Any $\vec{x}_i$ with a non-zero ϵ_i is a <u>support vector</u>.

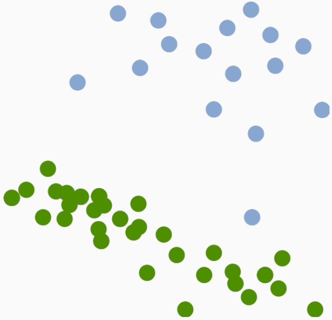
Soft margin objective:

$$\min_{\vec{\beta}} \|\vec{\beta}\|_2^2 + C\sum_{i=1}^{n} \epsilon_i.$$

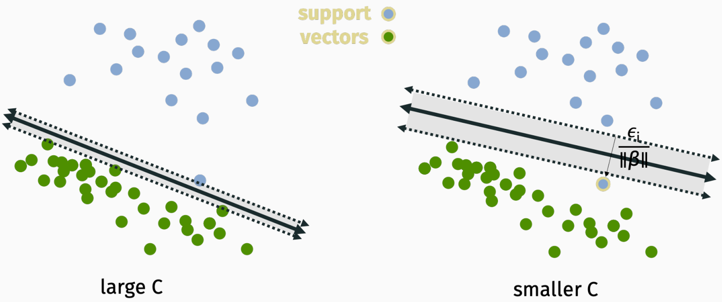
- Large C means penalties are punished more in objective $\implies$ smaller margin, less support vectors.
- Small C means penalties are punished less in objective $\implies$ larger margin, more support vectors.

When data is linearly separable, as $C \to \infty$ we will always get a separating hyperplane. A smaller value of C might lead to a more robust solution.

Example dataset:



EFFECT OF C



The classifier on the right is intuitively more robust. So for this data, a smaller choice for C might make sense.

Reformulation of soft-margin objective:

$$\max_{\vec{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_i \langle \vec{x}_i, \vec{x}_j \rangle - \frac{1}{2C}\sum_{i=1}^{n} \alpha_i^2$$

$$\text{subject to} \;\; \alpha_i \geq 0, \;\; \sum_{i=1}^{n} \alpha_i y_i = 0.$$

Obtained by taking the Lagrangian dual of the objective. Beyond the scope of this class, but important for a few reasons:

- Objective only depends on inner products $\langle \vec{x}_i, \vec{x}_j \rangle$, which makes it clear how to combine the soft-margin SVM with a kernel.
- Dual formulation can be solved faster in low-dimensions.
- Possible to prove that α_i is only non-zero for the support vectors. When classifying a new data point, only need to compute inner products (or the non-linear kernel inner product) with this subset of training vectors.

Some basic transformations of the soft-margin objective:

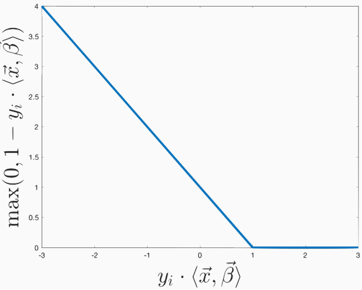
$$\min_{\vec{\beta}} \|\vec{\beta}\|_2^2 + C\sum_{i=1}^{n} \epsilon_i.$$

$$\min_{\vec{\beta}} \|\vec{\beta}\|_2^2 + C\sum_{i=1}^{n} \max(0, 1 - y_i \cdot \langle \vec{x}_i, \vec{\beta}\rangle).$$

$$\min_{\vec{\beta}} \lambda\|\vec{\beta}\|_2^2 2 + \sum_{i=1}^{n} \max(0, 1 - y_i \cdot \langle \vec{x}_i, \vec{\beta}\rangle).$$

These are all equivalent. $\lambda = 1/C$ is just another scaling parameter.

Hinge-loss: $\max(0, 1 - y_i \cdot \langle \vec{x}_i, \vec{\beta} \rangle)$. Recall that $y_i \in \{-1, 1\}$.

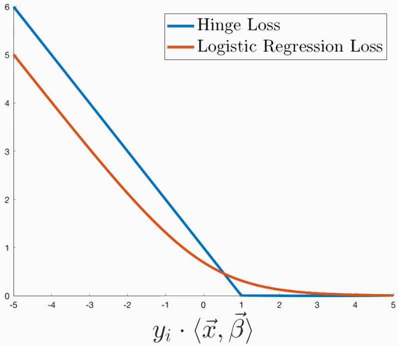


Soft-margin SVM:

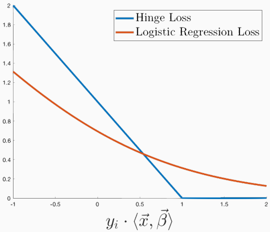
$$\min_{\vec{\beta}} \left[\sum_{i=1}^{n} \max(0, 1 - y_i \cdot \langle \vec{x}_i, \vec{\beta} \rangle) + \lambda \|\vec{\beta}\|_2^2 \right]. \quad (1)$$

Compare this to the logistic regression loss (slightly reformulated for $y_i \in \{-1, 1\}$):

$$\sum_{i=1}^{n} -\log(1 - \frac{1}{1 - e^{y_i \cdot \langle \vec{x}_i, \vec{\beta} \rangle}} \rangle$$

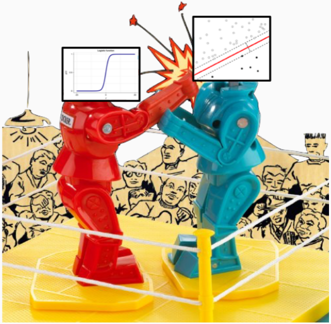


So, in the end, the function minimized when finding $\vec{\beta}$ for the standard **soft-margin SVM** is very similar to the objective function minimized when finding $\vec{\beta}$ using **logistic regression with ℓ_2 regularization.** Sort of...



Both functions can be optimized using first-order methods like gradient descent. This is now a common choice for large problems.

The jury is still out on how different these methods are...



- Work through **`demo_mnist_svm.ipynb`**.
- Then complete lab **`lab_mnist_partial.ipynb`**.