

# **CS-GY 6923: Lecture 9**

## **Kernel Methods, Support Vector Machines**

---

NYU Tandon School of Engineering, Akbar Rafiey

# Non-linear methods

- Previous methods studied (regression, logistic regression) are considered linear methods.
- They make predictions based on  $\langle \mathbf{x}, \beta \rangle$  – i.e. based on weighted sums of features.
- Next part of the course: we move on to non-linear methods. Specifically, **kernel methods** and **neural networks**.
- Both are very closely related to feature transformations, which was one technique we saw for using linear methods to learn non-linear concepts.

## Recall: $k$ -nearest neighbor method

**$k$ -NN algorithm:** a simple but powerful baseline for classification.

**Training data:**  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$  where  $y_1, \dots, y_n \in \{1, \dots, q\}$ .

**Classification algorithm:**

Given new input  $\mathbf{x}_{new}$ ,

- Compute  $sim(\mathbf{x}_{new}, \mathbf{x}_1), \dots, sim(\mathbf{x}_{new}, \mathbf{x}_n)$ .<sup>1</sup>
- Let  $\mathbf{x}_{j_1}, \dots, \mathbf{x}_{j_k}$  be the training data vectors with highest similarity to  $\mathbf{x}_{new}$ .
- Predict  $y_{new}$  as *majority*( $y_{j_1}, \dots, y_{j_k}$ ).

---

<sup>1</sup> $sim(\mathbf{x}_{new}, \mathbf{x}_i)$  is any chosen similarity function, like  $1 - \|\mathbf{x}_{new} - \mathbf{x}_i\|_2$ .

# $k$ -nearest neighbor method

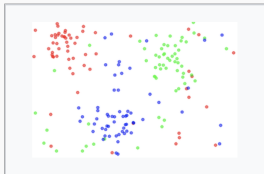


Fig. 1. The dataset.

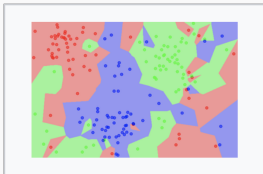


Fig. 2. The 1NN classification map.

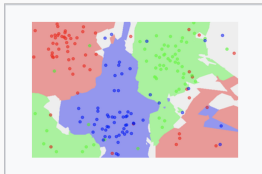


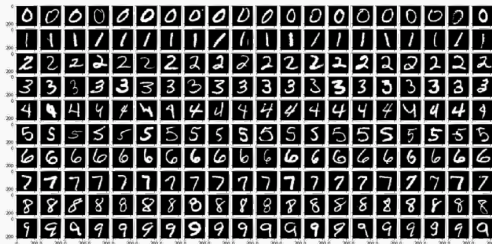
Fig. 3. The 5NN classification map.

- Smaller  $k$ , more complex classification function.
- Larger  $k$ , more robust to noisy labels.

**Works remarkably well for many datasets.**

# MNIST image data

Especially good for large datasets with lots of repetition. Works well on MNIST for example:



≈ 95% Accuracy out-of-the-box.<sup>2</sup>

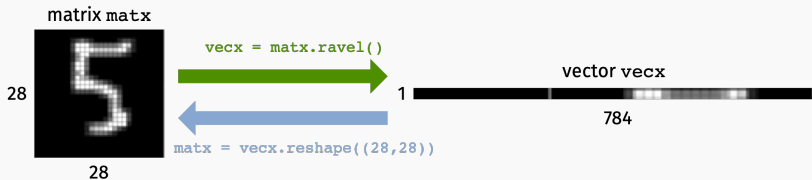
Let's look into this example a bit more...

---

<sup>2</sup>Can be improved to 99.5% with a fancy similarity function!

# MNIST image data

Each pixel is number from  $[0, 1]$ . 0 is black, 1 is white. Represent  $28 \times 28$  matrix of pixel values as a flattened vector.



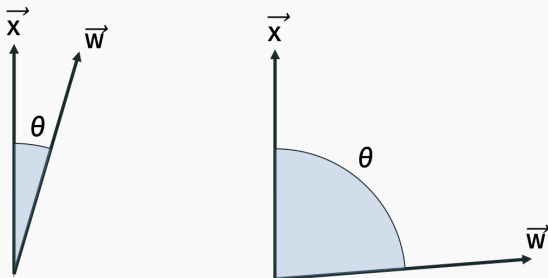
```
xmat = np.array([[1,2,3],[4,5,6],[7,8,9]])  
array([[1, 2, 3],  
       [4, 5, 6],  
       [7, 8, 9]])
```

```
xvec = xmat.ravel()  
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

## Inner product similarity

Given data vectors  $\mathbf{x}, \mathbf{w} \in \mathbb{R}^d$ , the inner product  $\langle \mathbf{x}, \mathbf{w} \rangle$  is a natural similarity measure.

$$\langle \mathbf{x}, \mathbf{w} \rangle = \sum_{i=1}^d x_i w_i = \cos(\theta) \|\mathbf{x}\|_2 \|\mathbf{w}\|_2.$$



Also called “cosine similarity”.

## Inner product similarity

Connection to Euclidean ( $\ell_2$ ) Distance:

$$\|\mathbf{x} - \mathbf{w}\|_2^2 = \|\mathbf{x}\|_2^2 + \|\mathbf{w}\|_2^2 - 2\langle \mathbf{x}, \mathbf{w} \rangle$$

If all data vectors has the same norm, the pair of vectors with largest inner product is the pair with smallest Euclidean distance.



## Inner product for mnist

Inner product between MNIST digits:

$$\mathbf{x} = \begin{bmatrix} \circ & \circ \\ \dots & \dots \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} \circ & \circ \\ \dots & \dots \end{bmatrix}$$

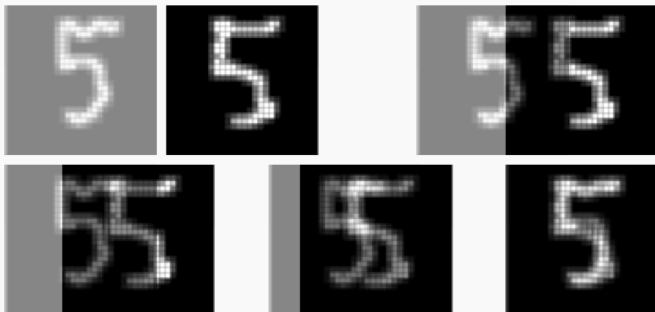


$$\langle \mathbf{x}, \mathbf{w} \rangle = \sum_{i=1}^{28} \sum_{j=1}^{28} \text{matx}[i, j] \cdot \text{matw}[i, j].$$

Inner product similarity is higher when the images have large pixel values (close to 1) in the same locations. I.e. when they have a lot of overlapping white/light gray pixels.

## Inner product for mnist

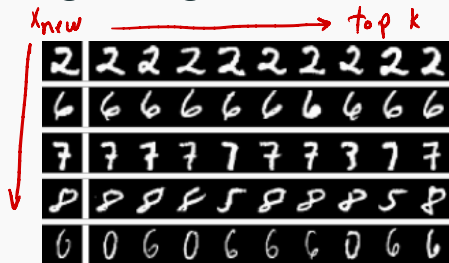
Visualizing the inner product between two images:



Images with high inner product have a lot of overlap.

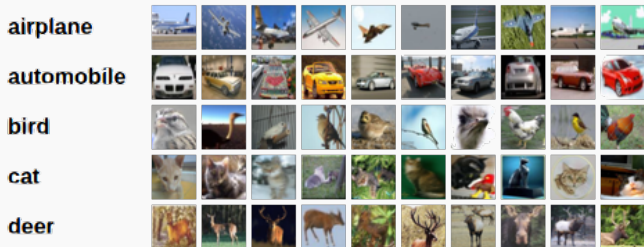
## k-NN algorithm on MNIST

Most similar images during  $k$ -NN search,  $k = 9$ :



## k-NN for other images

Does not work as well for less standardized classes of images:



CIFAR 10 Images

Even after scaling to have same size, converting to separate RGB channels, etc. something as simple as  $k$ -NN won't work.

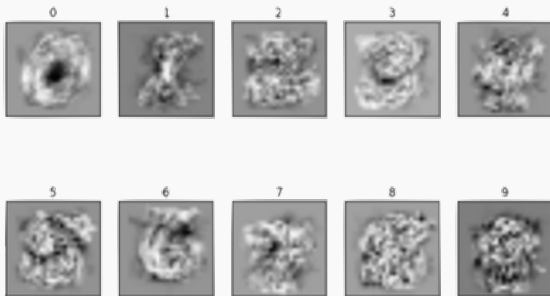
### One-vs.-all or Multiclass Cross-entropy Classification with Logistic Regression:

- Learn  $q$  classifiers with parameters  $\beta^{(1)}, \beta^{(2)}, \dots, \beta^{(q)}$ .
- Given  $\mathbf{x}_{new}$  compute  $\langle \mathbf{x}_{new}, \beta^{(1)} \rangle, \dots, \langle \mathbf{x}_{new}, \beta^{(q)} \rangle$
- Predict class  $y_{new} = \arg \max_j \langle \mathbf{x}_{new}, \beta^{(j)} \rangle$ .

If each  $\mathbf{x}$  is a vector with  $28 \times 28 = 784$  entries than each  $\beta^{(i)}$  also has 784 entries. Each parameter vector can be viewed as a  $28 \times 28$  image.

## Matched filter

Visualizing  $\beta^{(0)}, \dots, \beta^{(9)}$ :

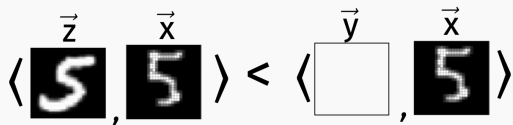


**Logistic regression classification rule:** For an input **5**, compute inner product similarity with all weight matrices and choose most similar one.

In contrast to  $k$ -NN, only need to compute similarity with 10 items instead of  $n$ .

## Diving into similarity

Often the inner product **does not make sense** as a similarity measure between data vectors. Here's an example (recall that smaller inner product means less similar):

$$\langle \vec{z}, \vec{x} \rangle < \langle \vec{y}, \vec{x} \rangle$$


But clearly the first image is more similar.

$$\langle \vec{z}, \vec{x} \rangle < \langle \vec{y}, \vec{x} \rangle$$


Here's a more realistic scenario.

## Kernel functions: a new measure of similarity

A kernel function  $k(\mathbf{x}, \mathbf{y})$  is simply a similarity measure between data points.

$$k(\mathbf{x}, \mathbf{y}) = \begin{cases} \text{large if } \mathbf{x} \text{ and } \mathbf{y} \text{ are similar.} \\ \text{close to 0 if } \mathbf{x} \text{ and } \mathbf{y} \text{ are different.} \end{cases}$$

**Example:** The Radial Basis Function (RBF) kernel, aka the Gaussian kernel:

$$k(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|_2^2/\sigma^2}$$

for some scaling factor  $\sigma$ .

$$k\left(\overset{\vec{z}}{\begin{array}{|c|} \hline \text{5} \\ \hline \end{array}}, \overset{\vec{x}}{\begin{array}{|c|} \hline \text{5} \\ \hline \end{array}}\right) > k\left(\overset{\vec{y}}{\begin{array}{|c|} \hline \square \\ \hline \end{array}}, \overset{\vec{x}}{\begin{array}{|c|} \hline \text{5} \\ \hline \end{array}}\right)$$



## Kernel functions: a new measure of similarity

Lots of kernel functions involve transformations of  $\langle \mathbf{x}, \mathbf{y} \rangle$  or  $\|\mathbf{x} - \mathbf{y}\|_2$ :

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

But you can imagine much more complex similarity metrics.

## How to use a kernel function?

For  $k$ -nearest neighbors, can easily replace inner product with whatever similarity function you want.

For logistic regression, it is less clear how to do so.

# How to use a kernel function?

## Logistic Regression Loss:

$$L(\beta^{(1)}, \dots, \beta^{(q)}) = - \sum_{i=1}^n \sum_{\ell=1}^q \mathbb{1}[y_i = \ell] \cdot \log \frac{e^{\langle \beta^{(\ell)}, \mathbf{x}_i \rangle}}{\sum_{j=1}^q e^{\langle \beta^{(j)}, \mathbf{x}_i \rangle}}$$

Loss inherently involves inner product between each  $\beta^{(j)}$  and each data vector  $\mathbf{x}_i$ .

**Solution:** Only work with similarity metrics that can be expressed as inner products.

## Kernel functions from feature transformation

A positive semidefinite (PSD) kernel is any similarity function with the following form:

$$k(\mathbf{x}, \mathbf{w}) = \phi(\mathbf{x})^T \phi(\mathbf{w}) \in \mathcal{R}$$

where  $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^m$  is a some feature transformation function.

# Kernel functions and feature transformation

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

$$\phi: \mathbb{R}^3 \rightarrow \mathbb{R}^{10}$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

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

$$\phi(\mathbf{w}) = \begin{bmatrix} \frac{1}{\sqrt{2}}w_1 \\ \frac{1}{\sqrt{2}}w_2 \\ \vdots \\ \vdots \\ \vdots \end{bmatrix}$$

$$\begin{aligned} (\mathbf{x}^T \mathbf{w} + 1)^2 &= (x_1 w_1 + x_2 w_2 + x_3 w_3 + 1)^2 \\ &= 1 + 2x_1 w_1 + 2x_2 w_2 + 2x_3 w_3 + x_1^2 w_1^2 + x_2^2 w_2^2 + x_3^2 w_3^2 \\ &\quad + 2x_1 w_1 x_2 w_2 + 2x_1 w_1 x_3 w_3 + 2x_2 w_2 x_3 w_3 \\ &= \phi(\mathbf{x})^T \phi(\mathbf{w}). \end{aligned}$$

## Kernel functions and feature transformation

$$\phi(x)^T \phi(y)$$

Not all similarity metrics are positive semidefinite (PSD), but all of the ones we saw earlier are:

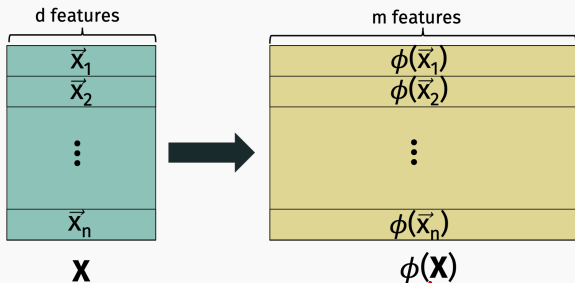
- Gaussian RBF Kernel:  $k(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|_2^2/\sigma^2}$
- Laplace Kernel:  $k(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|_2/\sigma}$
- Polynomial Kernel:  $k(\mathbf{x}, \mathbf{y}) = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^q$ .

And there are many more...

# Kernel functions and feature transformation

Feature transformations  $\iff$  new similarity metrics.

To work with the similarity  $k(\cdot, \cdot)$  in place of the inner product  $\langle \cdot, \cdot \rangle$ , it suffices to replace every data point  $\mathbf{x}_1, \dots, \mathbf{x}_n$  by  $\phi(\mathbf{x}_1), \dots, \phi(\mathbf{x}_n)$ .



$\uparrow$  do LR on  $\phi(\mathbf{X})$

# Kernel functions and feature transformation

$$\phi: \mathbb{R}^d \rightarrow \mathbb{R}^m$$

There are two major issues with this:

- While  $\phi(\mathbf{x})$  is sometimes simple and explicit. **More often, it is not.** We might be able to show a kernel is PSD without easily being able to write down  $\phi(\mathbf{x})$ .
- Transform dimension  $m$  is often very large: e.g.  $m = O(d^q)$  for a degree  $q$  polynomial kernel. For many kernels (e.g. the Gaussian kernel)  $m$  is actually *infinite*.

So doing the feature transformation explicitly would have very high computational cost. Ideally we would like algorithms that run in better than  $O(\infty)$  time.



## Reparameterization trick

For simplicity, let's just consider the binary cross entropy/logistic regression loss:

$$L(\beta) = - \sum_{j=1}^n y_j \log(h(\mathbf{X}\beta)_j) + (1 - y_j) \log(1 - h(\mathbf{X}\beta)_j)$$

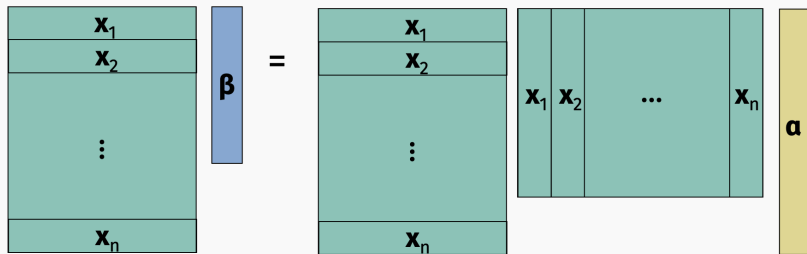
where  $h(z) = \frac{1}{1+e^{-z}}$ .

# Reparameterization trick

**Reminder from linear algebra:** Without loss of generality, can assume that  $\beta$  lies in the row span of  $\mathbf{X}$ .  $\beta = \sum \alpha_i x_i = \mathbf{X}^T \alpha$

So for any  $\beta \in \mathbb{R}^d$ , there exists a vector  $\alpha \in \mathbb{R}^n$  such that:

$$\underline{\mathbf{X}} \beta = \underline{\mathbf{X}} \underline{\mathbf{X}}^T \alpha.$$



$$\beta = r + a$$

$$\Rightarrow \mathbf{X} \beta = \mathbf{X}(r + a) = \mathbf{X}r + \mathbf{X}a$$

## Reparameterization trick

**Logistic Regression Equivalent Formulation:** Given data matrix  $\mathbf{X} \in \mathbb{R}^{n \times d}$  and binary label vector  $\mathbf{y} \in \{0, 1\}^n$  for class  $i$ , find  $\alpha \in \mathbb{R}^n$  to minimize the loss:

$$L(\alpha) = - \sum_{j=1}^n y_j \log(h(\mathbf{X}\mathbf{X}^T \alpha)_j) + (1 - y_j) \log(1 - h(\mathbf{X}\mathbf{X}^T \alpha)_j)$$

$\downarrow \quad \searrow \rightarrow$   
 $n \times d \quad d \times n \quad \rightarrow \quad n \times n$

Can still be minimized via gradient descent:

$$\nabla L(\alpha) = \mathbf{X}\mathbf{X}^T (h(\mathbf{X}\mathbf{X}^T \alpha) - \mathbf{y}).$$

$$\mathbf{X}\mathbf{X}^T [i, j] = \langle x_i, x_j \rangle$$

$$\nabla L(\beta) = \mathbf{X}\mathbf{X}^T (h(\mathbf{X}\beta) - \mathbf{y})$$

u<sub>27</sub>

## Reparameterization trick

If we use a non-linear data transformation  $\phi$  (corresponding to a PSD kernel), then the loss is:

$$-\sum_{j=1}^n y_j \log(h(\phi(\mathbf{X})\phi(\mathbf{X})^T \boldsymbol{\alpha})_j) + (1 - y_j) \log(1 - h(\phi(\mathbf{X})\phi(\mathbf{X})^T \boldsymbol{\alpha})_j)$$

$$\begin{aligned} \phi(x) \phi(x)^T [i, j] &= \langle \phi(x_i), \phi(x_j) \rangle \\ &= \phi(x_i)^T \phi(x_j) = k(x_i, x_j) \end{aligned}$$

# Kernel matrix

$\mathbf{K} = \phi(\mathbf{X})\phi(\mathbf{X})^T$  is called the kernel Gram matrix.

$$\phi(\mathbf{X})\phi(\mathbf{X})^T = \begin{array}{|c|} \hline \phi(\vec{x}_1) \\ \hline \phi(\vec{x}_2) \\ \hline \vdots \\ \hline \phi(\vec{x}_n) \\ \hline \end{array} \begin{array}{|c|c|c|c|} \hline \phi(\vec{x}_1) & \phi(\vec{x}_2) & \dots & \phi(\vec{x}_n) \\ \hline \end{array} = \begin{array}{|c|} \hline k(\vec{x}_i, \vec{x}_j) \\ \hline \end{array} \mathbf{K}$$

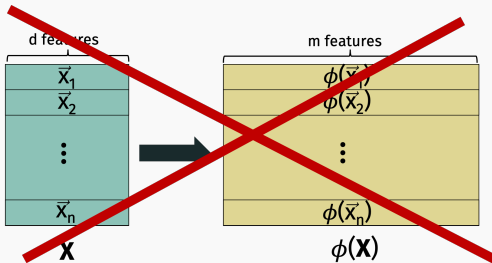
$$X \succeq 0 \text{ iff}$$

$$v^T X v \geq 0 \quad \forall v$$

# Kernel trick

We never need to actually compute  $\phi(\mathbf{x}_1), \dots, \phi(\mathbf{x}_n)$  explicitly!

- For training we just need the kernel matrix  $\mathbf{K}$ , which requires computing  $k(\mathbf{x}_i, \mathbf{x}_j)$  for all  $i, j$ .



We can always work with a finite sized  $n \times n$  matrix.

## Take away:

- Logistic regression can be combined with any positive semidefinite kernel matrix, and the model can be trained in time independent of the transform dimension  $m$ .

# Kernel trick: prediction

## Prediction:

- Prediction can also be done efficiently. For a new input  $\mathbf{x}_{new}$ , we need to compute:

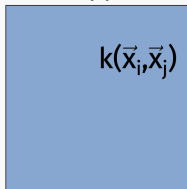
$$\begin{aligned}\langle \phi(\mathbf{x}_{new}), \boldsymbol{\beta} \rangle &= \langle \phi(\mathbf{x}_{new}), \phi(\mathbf{X})^T \boldsymbol{\alpha} \rangle \\ &= \langle \phi(\mathbf{x}_{new}), \sum_{i=1}^n \phi(\mathbf{x}_i) \alpha_i \rangle = \sum_{i=1}^n \alpha_i \langle \phi(\mathbf{x}_{new}), \phi(\mathbf{x}_i) \rangle.\end{aligned}$$

Each term in the sum  $\langle \phi(\mathbf{x}_{new}), \phi(\mathbf{x}_i) \rangle = k(\mathbf{x}_{new}, \mathbf{x}_i)$  can be computed without explicit feature transformation.



## Beyond the kernel trick

The kernel matrix  $\mathbf{K}$  is still  $n \times n$  though which is huge when the size of the training set  $n$  is large. Has made the kernel trick less appealing in some modern ML applications.


$$k(\vec{x}_i, \vec{x}_j)$$

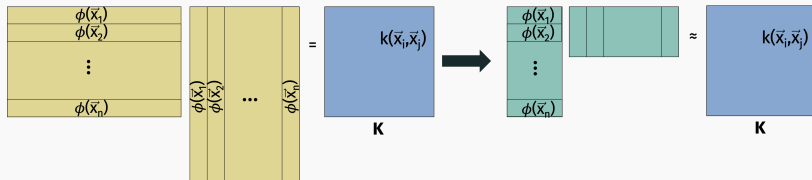
$\mathbf{K}$

There is an inherent quadratic dependence on  $n$  in the computational and space complexity of kernel methods.

- 10,000 data points  $\rightarrow$  runtime scales as  $\sim 100,000,000$ ,  $\mathbf{K}$  takes 800MB of space.
- 1,000,000 data points  $\rightarrow$  runtime scales as  $\sim 10^{12}$ ,  $\mathbf{K}$  takes 8TB of space.

# Beyond the kernel trick

Many algorithmic advances in recent years partially address this computational challenge (random Fourier features methods, Nystrom methods, etc.)



# Kernel regression

The kernel trick can also be applied outside of classification. E.g. to regression:

$$\min_{\beta} \|\mathbf{X}\beta - \mathbf{y}\|_2^2 + \lambda \|\beta\|_2^2 \rightarrow \min_{\alpha} \|\mathbf{X}\mathbf{X}^T \alpha - \mathbf{y}\|_2^2 + \lambda \|\mathbf{X}^T \alpha\|_2^2$$

$\hookrightarrow \beta = \mathbf{X}^T \alpha$

Replace  $\mathbf{X}\mathbf{X}^T$  by kernel matrix  $\mathbf{K}$  during training.

**Prediction:**

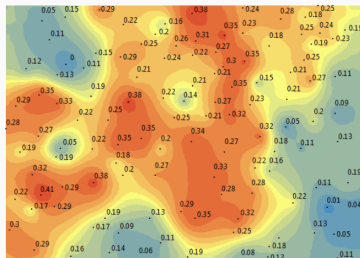
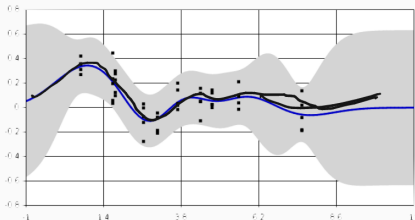
$$y_{new} = \sum_{i=1}^n \alpha_i \cdot k(\mathbf{x}_{new}, \mathbf{x}_i).$$

$$\sum_{i=1}^n (\langle \mathbf{x}_i, \beta \rangle - y_i)^2 + \|\beta\|_2^2$$
$$\sum_{i=1}^n w_i (\langle \mathbf{x}_i, \beta \rangle - y_i)$$

**Added benefit:** Relatively numerically stable. E.g. is a much better option for performing multivariate or even single variate polynomial regression than direct feature expansion.

# Kernel regression

We won't study kernel regression in detail, but kernel regression with non-linear kernels like  $e^{-\|\mathbf{x}-\mathbf{y}\|_2^2}$  is a very important statistical tool, especially when dealing with spatial or temporal data.



Also known as Gaussian Process (GP) Regression or Kriging.

12:20

# Support Vector Machines

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

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



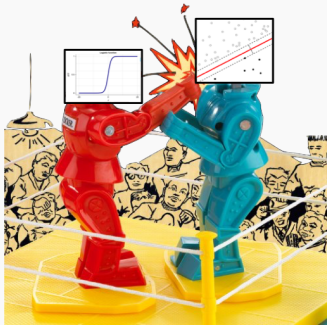
- Invented in 1963 by Alexey Chervonenkis and Vladimir Vapnik. Also founders of VC-theory.
- First combined with non-linear kernels in 1993.

## SVM's vs. logistic regression

For some reason, SVMs are more commonly associated 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 mostly for historical reasons and connections to theoretical machine learning.
- In the early 2000s SVMs were 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 the jury is still out...

# SVM's vs. logistic regression

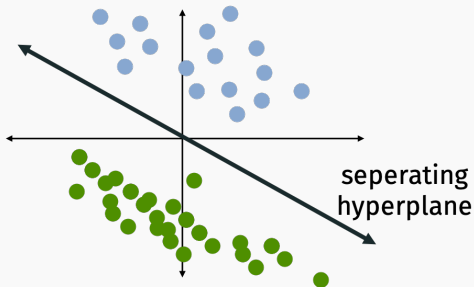


**Next lab:** Machina-a-machina comparison of SVMs vs. logistic regression for a MNIST digit classification problem. Which provides better accuracy? Which is faster to train?



## Linearly separable data

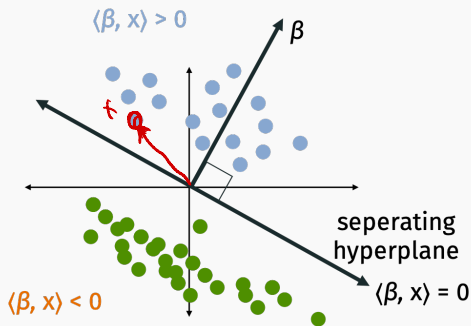
We call a dataset with binary labels linearly separable if it can be perfectly classified with a linear classifier:



This the realizable setting we discussed in the learning theory lecture.

## Linearly separable data

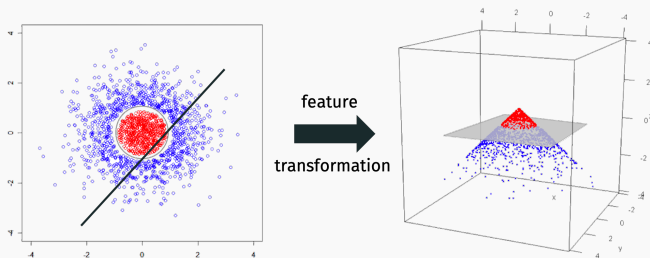
Formally, there exists a parameter  $\beta$  such that  $\langle \beta, \mathbf{x} \rangle > 0$  for all  $\mathbf{x}$  in class 1 and  $\langle \beta, \mathbf{x} \rangle < 0$  for all  $\mathbf{x}$  in class 0.



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

## Linearly separable data

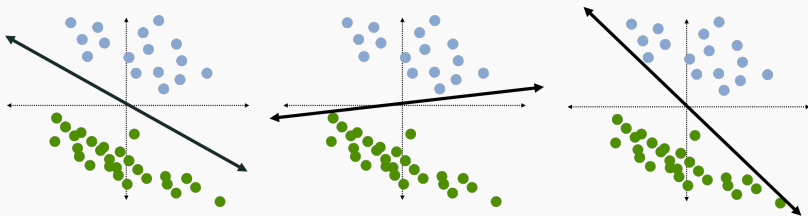
A data set might be linearly separable when using a non-linear kernel/feature transformation even if it is not separable in the original space.



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

# Margin

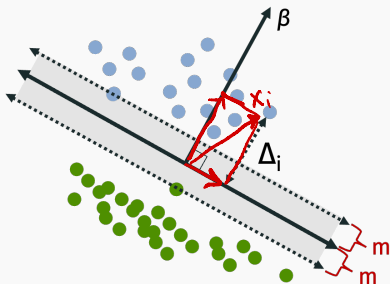
When data is linearly separable, there are typically multiple valid separating hyperplanes.



**Question from Vapnik and Chervonenkis:** Which hyperplane/classification rule is best?

# Margin

The **margin**  $m$  of a separating hyperplane is the minimum  $\ell_2$  (Euclidean) distance between a point in the dataset and the hyperplane.

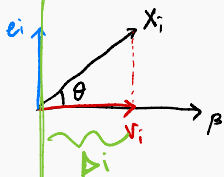


$$m = \min_i \Delta_i$$

where

$$\Delta_i = \frac{|\langle x_i, \beta \rangle|}{\|\beta\|_2}$$

# Margin



$$\|v_i\|_2 = \cos \theta \cdot \|x_i\|_2 \Rightarrow v_i = \cos \theta \|x_i\|_2 \cdot \frac{\beta}{\|\beta\|_2}$$

$$v_i = \langle v_i, \beta \rangle \frac{\beta}{\|\beta\|_2}$$

We have that  $x_i = v_i + e_i$  where  $v_i$  is parallel to  $\beta$  and  $e_i$  is perpendicular.

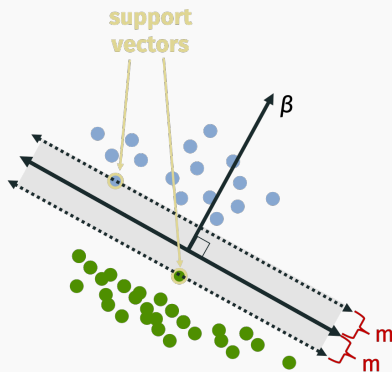
$$\Delta_i = \|v_i\|_2 = \frac{1}{\|v_i\|_2} \cdot \langle v_i, v_i \rangle = \frac{1}{\|v_i\|_2} \cdot \frac{\|v_i\|_2}{\|\beta\|_2} \cdot |\langle v_i, \beta \rangle| = \frac{|\langle v_i, \beta \rangle|}{\|\beta\|_2}$$

Finally, we have that  $\langle x_i, \beta \rangle = \langle v_i, \beta \rangle$  because  $\langle e_i, \beta \rangle = 0$ .

$$\begin{aligned} \langle x_i, \beta \rangle &= \langle v_i + e_i, \beta \rangle = \langle v_i, \beta \rangle + \langle e_i, \beta \rangle \\ &= \langle v_i, \beta \rangle \end{aligned}$$

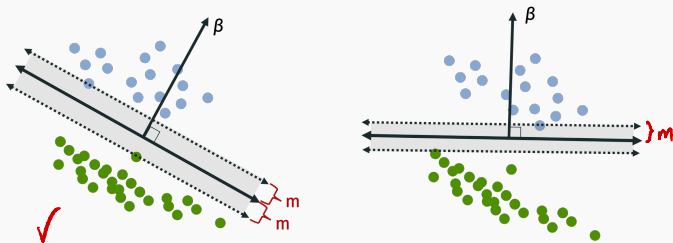
# Support vector

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



# Hard-margin svm

A hard-margin 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  $m^*$ .

$$\begin{aligned}
 m^* &= \max_{\beta} \left[ \min_{i \in \{1, \dots, n\}} \frac{|\langle \mathbf{x}_i, \beta \rangle|}{\|\beta\|_2} \right] \\
 &= \max_{\beta} \left[ \min_{i \in \{1, \dots, n\}} \frac{y_i \langle \mathbf{x}_i, \beta \rangle}{\|\beta\|_2} \right]
 \end{aligned}$$

where  $y_i = -1, 1$  depending on what class  $\mathbf{x}_i$ .<sup>3</sup>

---

<sup>3</sup>Note that this is a different convention than the 0, 1 class labels we typically use.

# Hard-margin svm

Equivalent formulation:

$$m^* = \max_{\mathbf{v}: \|\mathbf{v}\|_2=1} \left[ \min_{i \in \{1, \dots, n\}} y_i \cdot \langle \mathbf{x}_i, \mathbf{v} \rangle \right]$$

$$\text{Let } \mathbf{v}^* = \operatorname{argmax}_{\mathbf{v}: \|\mathbf{v}\|_2=1} \left[ \min_{i \in \{1, \dots, n\}} y_i \cdot \langle \mathbf{x}_i, \mathbf{v} \rangle \right]$$

$$\text{For all } i \in \{1, \dots, n\} \quad y_i \cdot \langle \mathbf{x}_i, \mathbf{v}^* \rangle \geq m^*$$

margin of any data point

$$\Rightarrow \underbrace{\frac{1}{m^*}}_{c} \cdot y_i \cdot \langle \mathbf{x}_i, \mathbf{v}^* \rangle \geq 1$$

$$\frac{1}{m^*} = \min_{\mathbf{v}: \|\mathbf{v}\|_2=1} \underbrace{c}_{\beta} \quad \text{subject to } c \cdot y_i \cdot \langle \mathbf{x}_i, \mathbf{v} \rangle \geq 1 \text{ for all } i.$$

$$= \min_{\mathbf{v}: \|\mathbf{v}\|_2=1} \underbrace{\|\mathbf{c} \cdot \mathbf{v}\|_2}_{\beta} \quad \text{subject to } y_i \cdot \langle \mathbf{x}_i, \underbrace{\mathbf{c} \cdot \mathbf{v}}_{\beta} \rangle \geq 1 \text{ for all } i.$$

$= c \|\mathbf{v}\|_2 = c \cdot 1 = c$

# Hard-margin svm

Equivalent formulation:

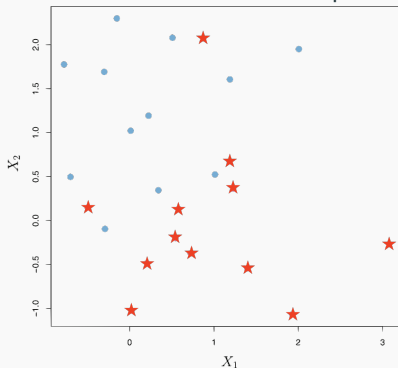
$$\min_{\beta} \|\beta\|_2^2 \quad \text{subject to} \quad \underline{y_i \cdot \langle \mathbf{x}_i, \beta \rangle \geq 1 \text{ for all } i.}$$

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.

# Hard-margin svm

Hard-margin SVMs have a few critical issues in practice:

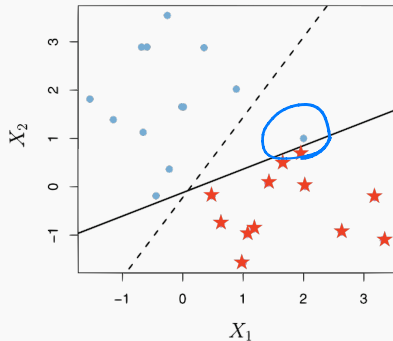
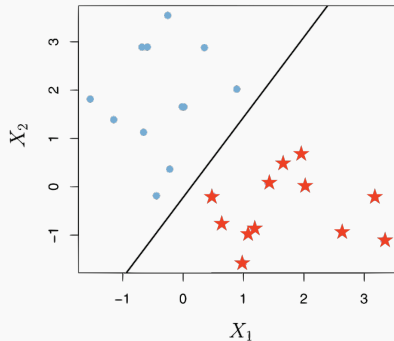


**Data might not be linearly separable, in-which case the maximum margin classifier is not even defined.**

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

# Hard-margin svm

Another critical issue in practice:



**Hard-margin SVM classifiers are not robust.**

## Soft-margin svm

**Solution:** Allow the classifier to make some mistakes!

**Hard margin objective:**

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

**Soft margin objective:**

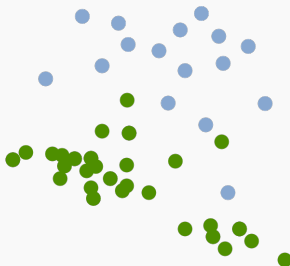
$$\vec{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)$$

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

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

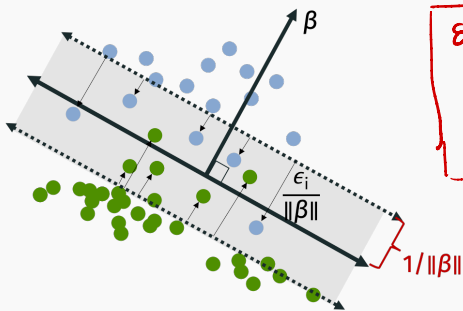
$C \geq 0$  is a non-negative tuning parameter.

Example of a non-separable problem:



## Soft-margin svm

Recall that  $\Delta_i = \frac{y_i \cdot \langle \mathbf{x}_i, \boldsymbol{\beta} \rangle}{\|\boldsymbol{\beta}\|_2}$ .



$\epsilon_i = 0$  means  $x_i$  is outside of this band.

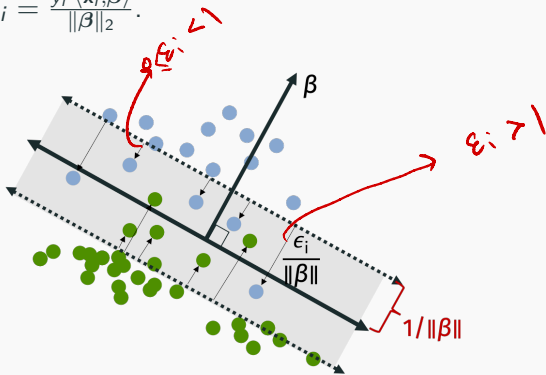
Soft margin objective:

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



## Soft-margin svm

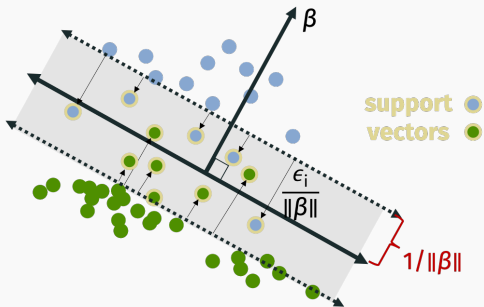
Recall that  $\Delta_i = \frac{y_i \cdot \langle \mathbf{x}_i, \boldsymbol{\beta} \rangle}{\|\boldsymbol{\beta}\|_2}$ .



Soft margin objective:

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

# Soft-margin svm



Any  $x_i$  with a non-zero  $\epsilon_i$  is a support vector.

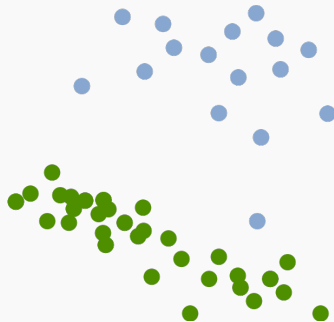
### Soft margin objective:

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

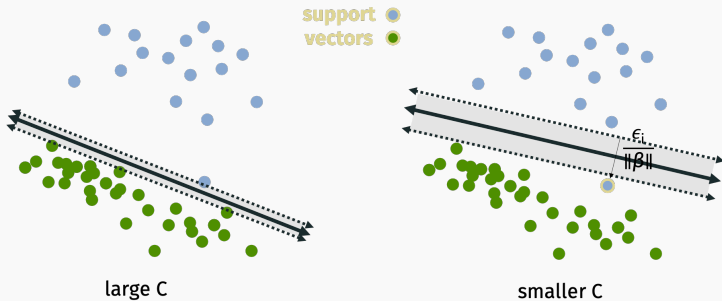
- Large  $C$  means penalties are punished more in objective  $\implies$  smaller margin, less support vectors.
- Small  $C$  means penalties are punished less in objective  $\implies$  larger margin, more support vectors.

When data is linearly separable, as  $C \rightarrow \infty$  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.

# Dual formulation

Reformulation of soft-margin objective:

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

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  $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ , which makes it clear how to combine the soft-margin SVM with a kernel.
- **Possible to prove that  $\alpha_i$  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. This is not the case for the logistic regression classifier.**

## Comparison to logistic regression

Some basic transformations of the soft-margin objective:

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

$\epsilon_i \geq 1 - y_i \cdot \langle \mathbf{x}_i, \beta \rangle$   
 $\epsilon_i = 1 - y_i \cdot \langle \mathbf{x}_i, \beta \rangle$

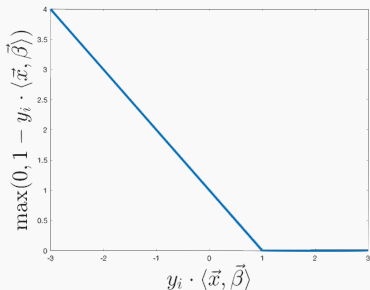
$$\min_{\beta} \|\beta\|_2^2 + C \sum_{i=1}^n \max(0, 1 - y_i \cdot \langle \mathbf{x}_i, \beta \rangle).$$
$$\min_{\beta} \|\beta\|_2^2 / C + \sum_{i=1}^n \max(0, 1 - y_i \cdot \langle \mathbf{x}_i, \beta \rangle)$$
$$\min_{\beta} \lambda \|\beta\|_2^2 + \sum_{i=1}^n \max(0, 1 - y_i \cdot \langle \mathbf{x}_i,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

$\lambda = \frac{1}{C}$

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

# Hinge loss

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



**Soft-margin SVM:**

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



## Logistic loss

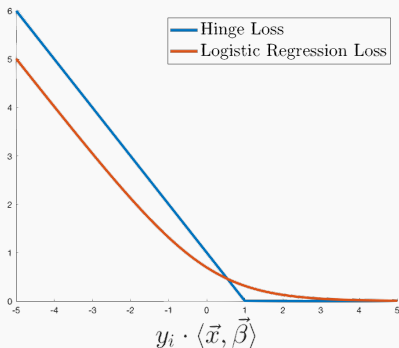
Recall the logistic loss for  $y_i \in \{0, 1\}$ :

$$\begin{aligned}L(\beta) &= - \sum_{i=1}^n y_i \log(h(\langle \mathbf{x}_i, \beta \rangle)) + (1 - y_i) \log(1 - h(\langle \mathbf{x}_i, \beta \rangle)) \\&= - \sum_{i=1}^n y_i \log \left( \frac{1}{1 + e^{-\langle \mathbf{x}_i, \beta \rangle}} \right) + (1 - y_i) \log \left( \frac{e^{-\langle \mathbf{x}_i, \beta \rangle}}{1 + e^{-\langle \mathbf{x}_i, \beta \rangle}} \right) \\&= - \sum_{i=1}^n y_i \log \left( \frac{1}{1 + e^{-\langle \mathbf{x}_i, \beta \rangle}} \right) + (1 - y_i) \log \left( \frac{1}{1 + e^{\langle \mathbf{x}_i, \beta \rangle}} \right)\end{aligned}$$

## Comparison of SVM to logistic regression

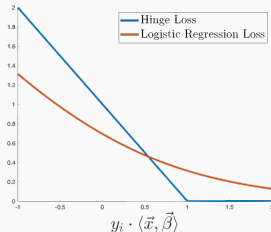
Compare this to the logistic regression loss reformulated for  $y_i \in \{-1, 1\}$ ):

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



## Comparison to logistic regression

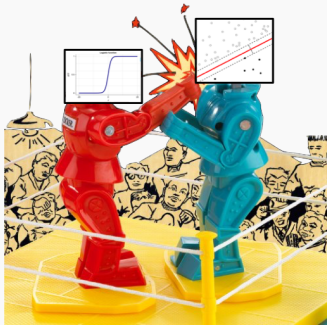
So, in the end, the function minimized when finding  $\beta$  for the standard **soft-margin SVM** is very similar to the objective function minimized when finding  $\beta$  using **logistic regression with  $\ell_2$  regularization**. Sort of...



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

# Comparison to logistic regression

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



- Work through Demo 6: `demo_mnist_svm.ipynb`.
- Lab 5 on SVM vs. Logistic Regression