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 $\langle \mathbf{x}, \boldsymbol{\beta} \rangle$ – 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: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ where $y_1, \ldots, y_n \in \{1, \ldots, q\}$.

Classification algorithm:

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

- Compute $sim(\mathbf{x}_{new}, \mathbf{x}_1), \ldots, sim(\mathbf{x}_{new}, \mathbf{x}_n)$.[1]
- Let $\mathbf{x}_{j_1}, \ldots, \mathbf{x}_{j_k}$ be the training data vectors with highest similarity to $\mathbf{x}_{new}$.
- Predict y_{new} as $majority(y_{j_1}, \ldots, y_{j_k})$.

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

k-nearest neighbor method

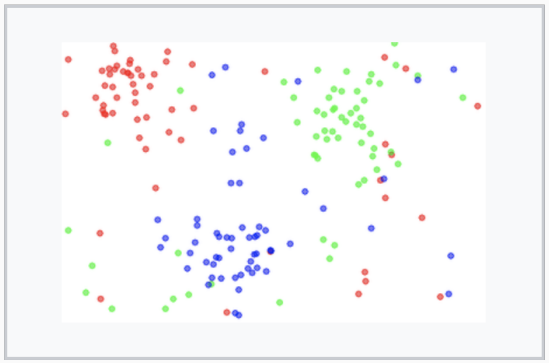


Fig. 1. The dataset.

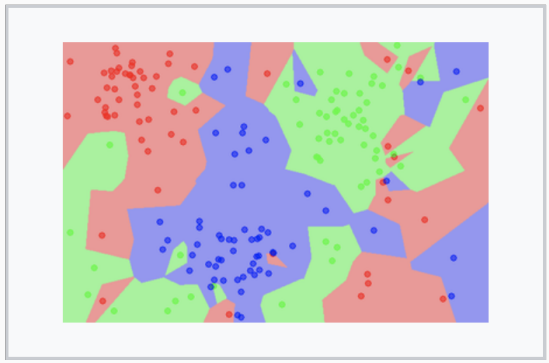


Fig. 2. The 1NN classification map.

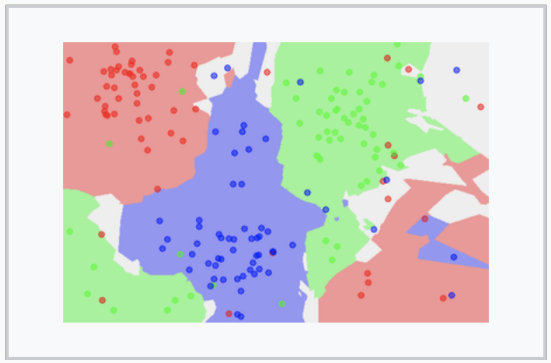


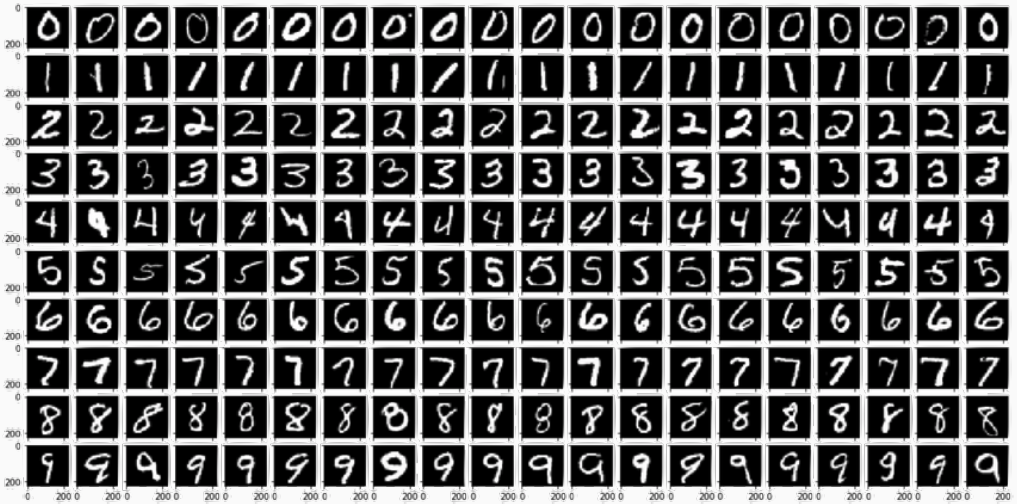
Fig. 3. The 5NN classification map.

- 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:



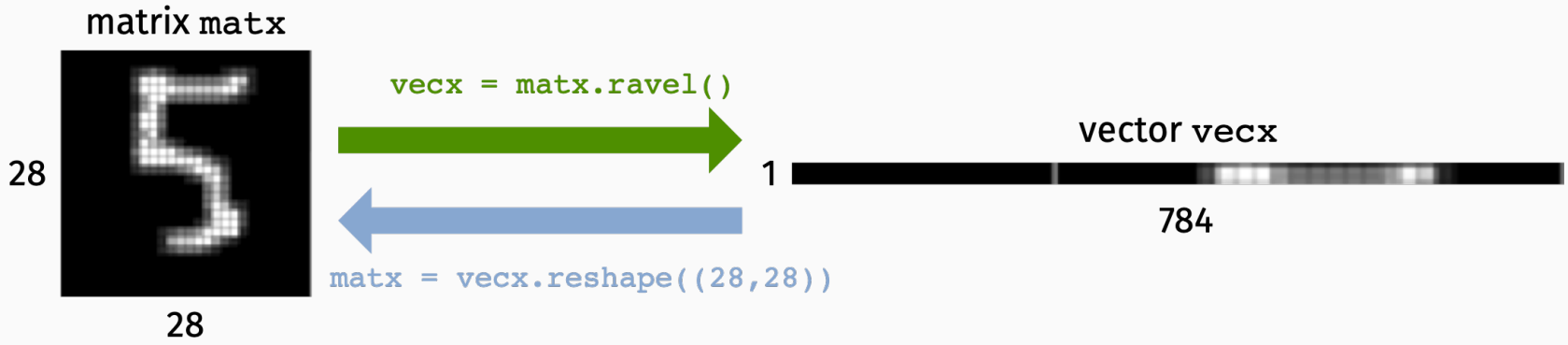
$\approx 95\%$ **Accuracy out-of-the-box.**[2]

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

[2] 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 28×28 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]])
```

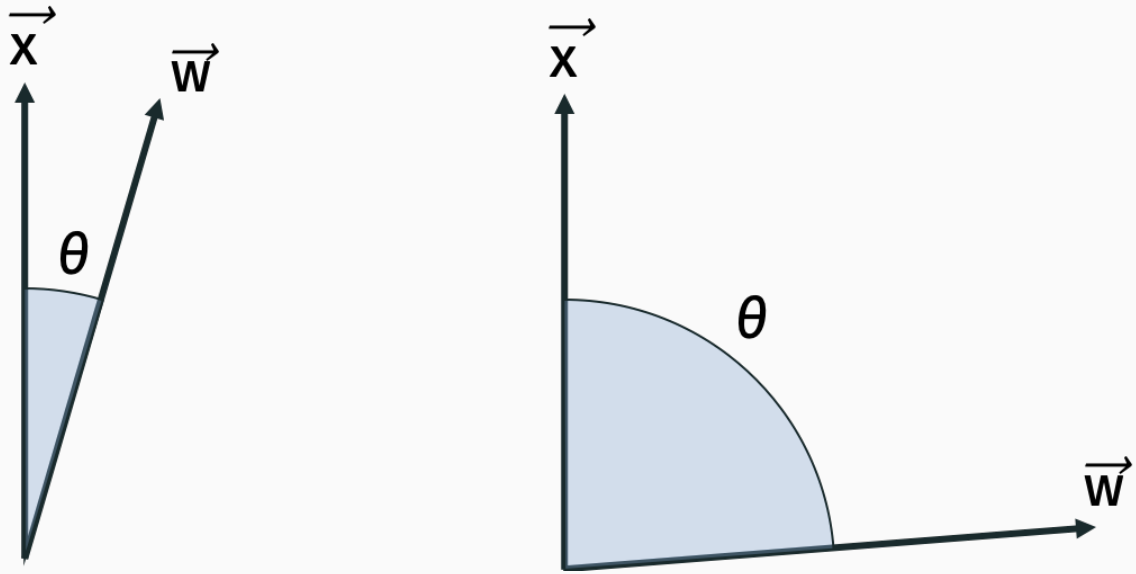
```
xvec = xmat.ravel()
```

```
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 (ℓ_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

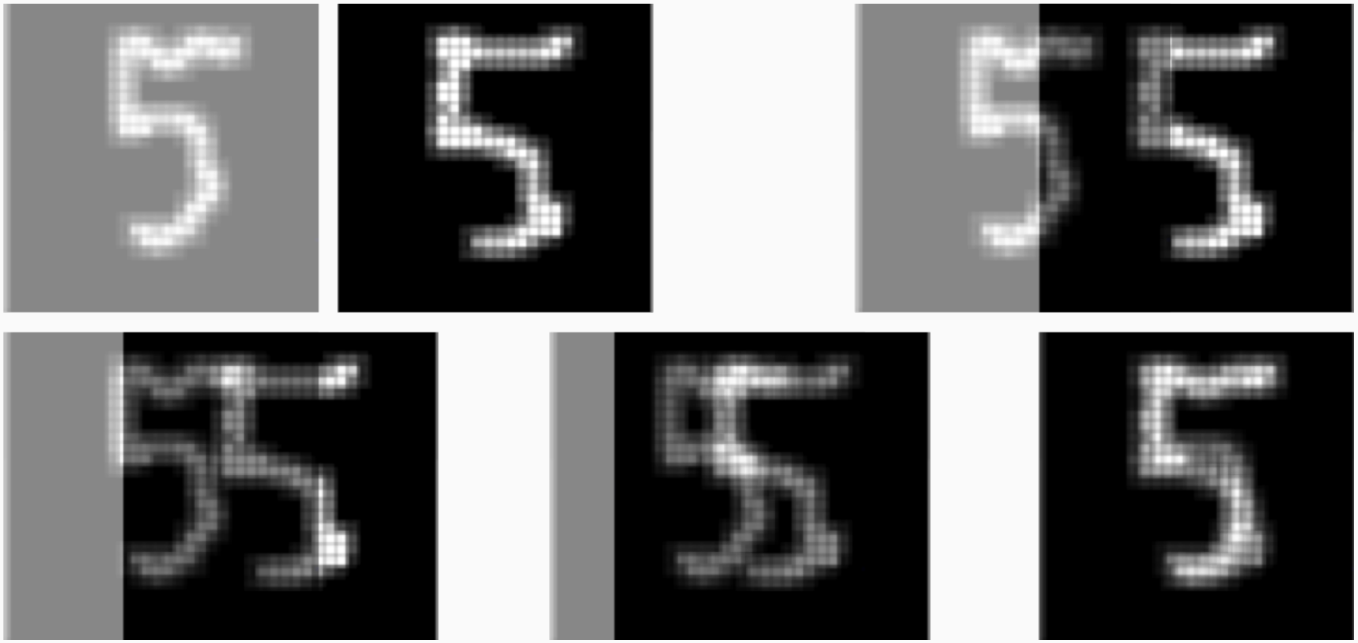
Inner product between MNIST digits:

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

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

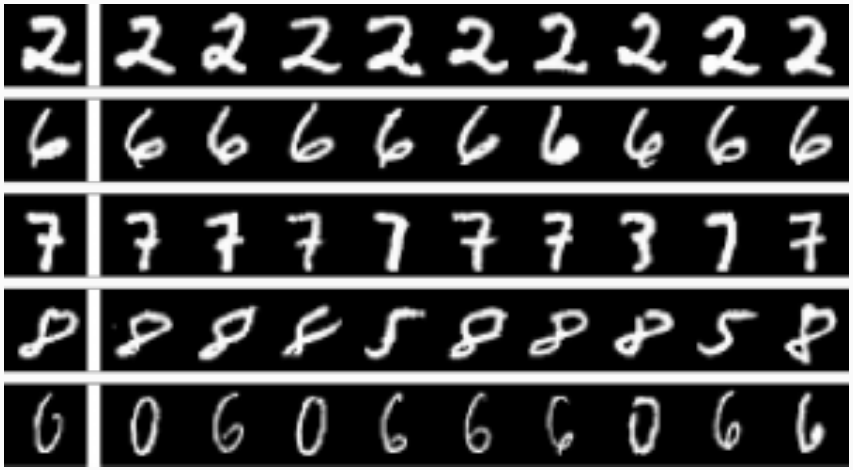
Visualizing the inner product between two images:



Images with high inner product have a lot of overlap.

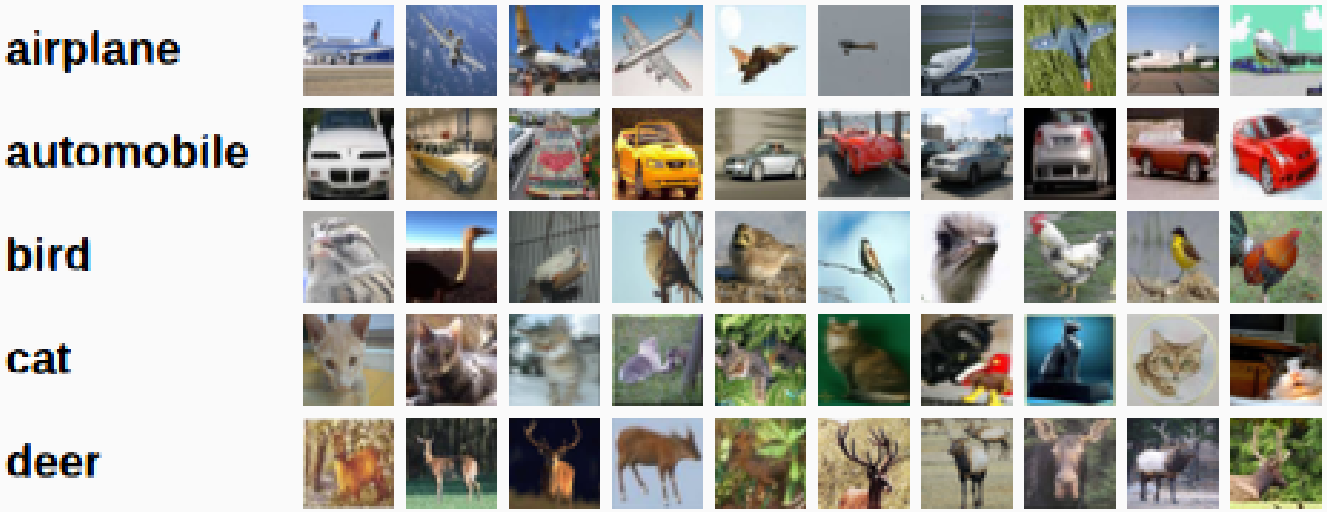
k-NN algorithm on MNIST

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.

Another view on logistic regression

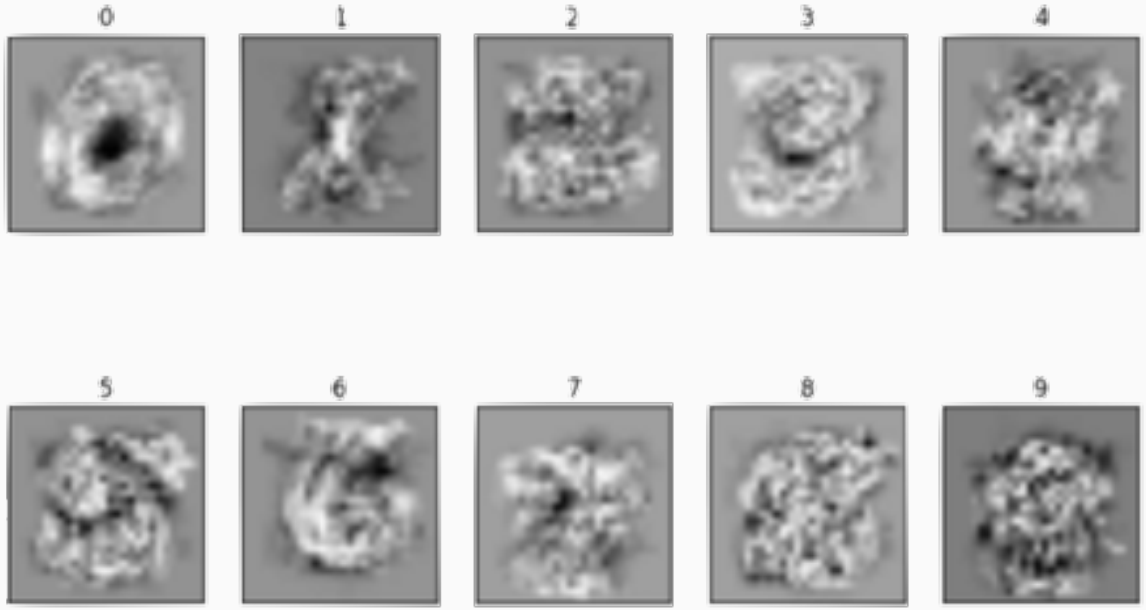
One-vs.-all or Multiclass Cross-entropy Classification with Logistic Regression:

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

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

Matched filter

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

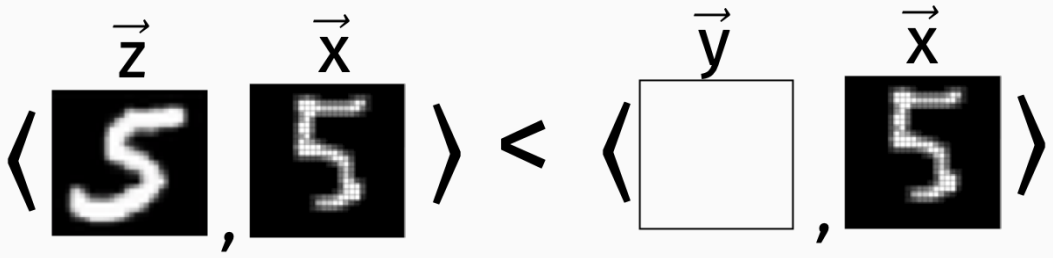


Logistic regression classification rule: For an input  , compute inner product similarity with all weight matrices and choose most similar one.

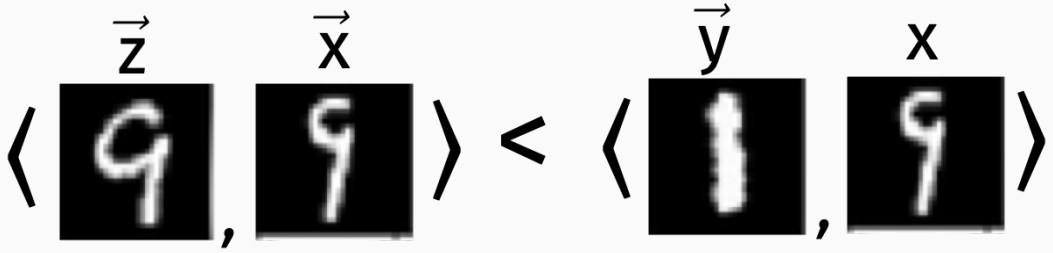
In contrast to k-NN, only need to compute similarity with 10 items instead of n.

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):



But clearly the first image is more similar.



Here's a more realistic scenario.

Kernel functions: a new measure of similarity

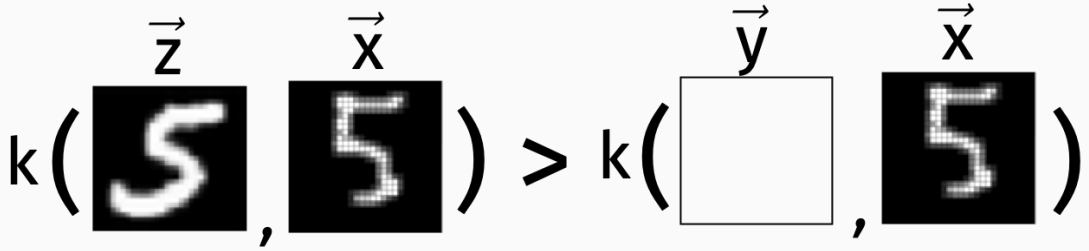
A kernel function $k(\mathbf{x}, \mathbf{y})$ is simply a similarity measure between data points.

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

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 σ.



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:

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

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:

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

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

Kernel functions and feature transformation

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

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

$$\begin{aligned}(\mathbf{x}^T\mathbf{w} + 1)^2 &= (x_1w_1 + x_2w_2 + x_3w_3 + 1)^2 \\ &= 1 + 2x_1w_1 + 2x_2w_2 + 2x_3w_3 + x_1^2w_1^2 + x_2^2w_2^2 + x_3^2w_3^2 \\ &\quad + 2x_1w_1x_2w_2 + 2x_1w_1x_3w_3 + 2x_2w_2x_3w_3 \\ &= \phi(\mathbf{x})^T\phi(\mathbf{w}).\end{aligned}$$

Kernel functions and feature transformation

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

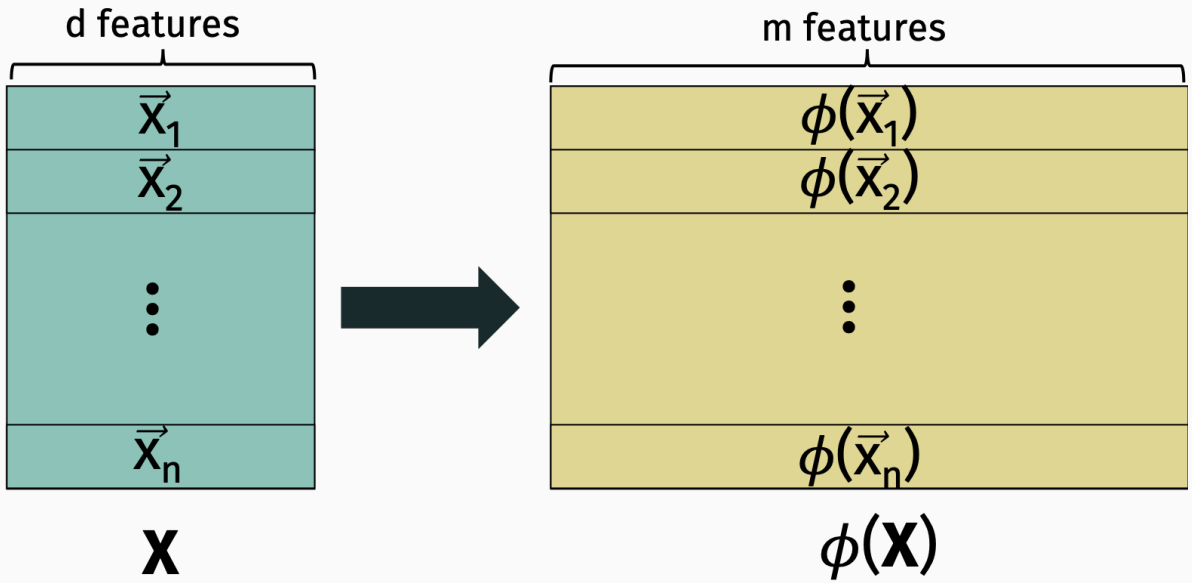
- Gaussian RBF Kernel: $k(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|_2^2/\sigma^2}$
- Laplace Kernel: $k(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|_2/\sigma}$
- Polynomial Kernel: $k(\mathbf{x}, \mathbf{y}) = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^q$.

And there are many more...

Kernel functions and feature transformation

Feature transformations $\Longleftrightarrow$ 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 $\mathbf{x}_1, \ldots, \mathbf{x}_n$ by $\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)$.



Kernel functions and feature transformation

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 then $O(\infty)$ time.

Reparameterization trick

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

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

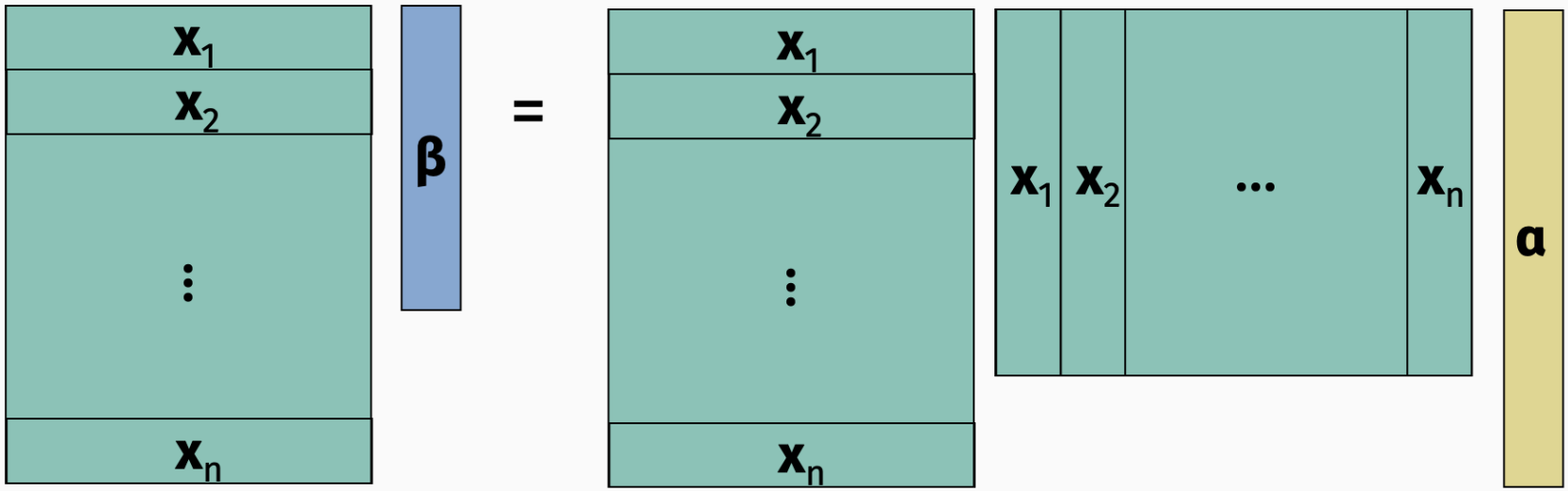
where $h(z) = \frac{1}{1+e^{-z}}$.

Reparameterization trick

Reminder from linear algebra: Without loss of generality, can assume that β lies in the row span of $\mathbf{X}$.

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

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



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 $\boldsymbol{\alpha} \in \mathbb{R}^n$ to minimize the loss:

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

Can still be minimized via gradient descent:

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

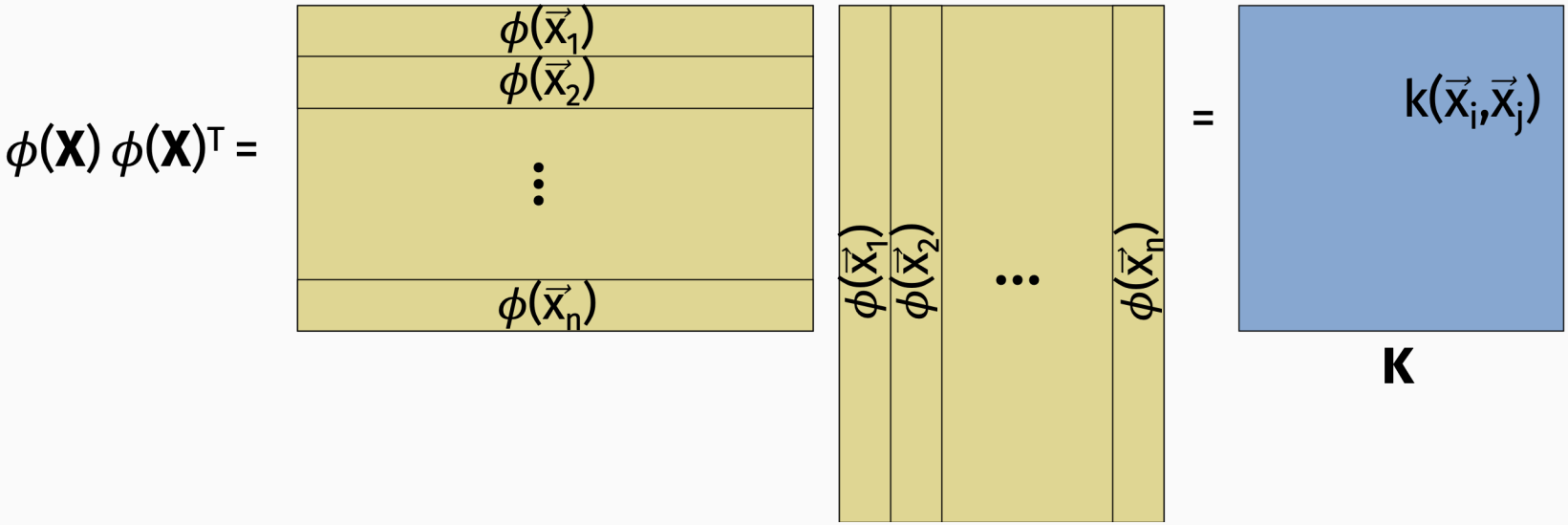
Reparameterization trick

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

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

Kernel matrix

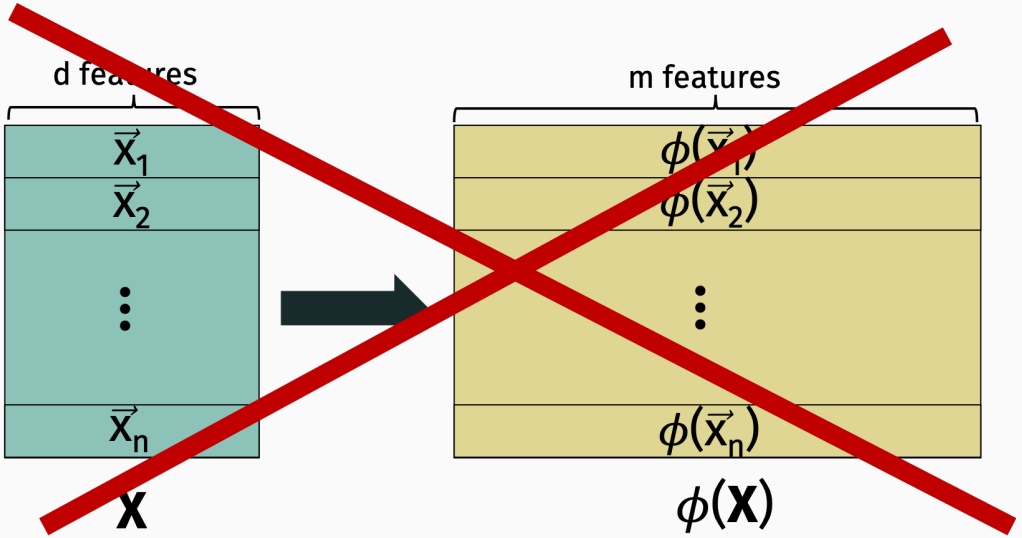
$\mathbf{K} = \phi(\mathbf{X})\phi(\mathbf{X})^T$ is called the kernel Gram matrix.



Kernel trick

We never need to actually compute $\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)$ 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.

Kernel trick

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:

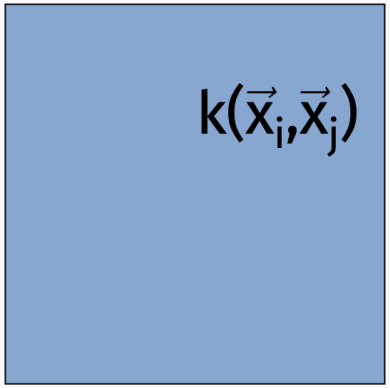
- Prediction can also be done efficiently. For a new input $\mathbf{x}_{new}$, we need to compute:

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

  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.

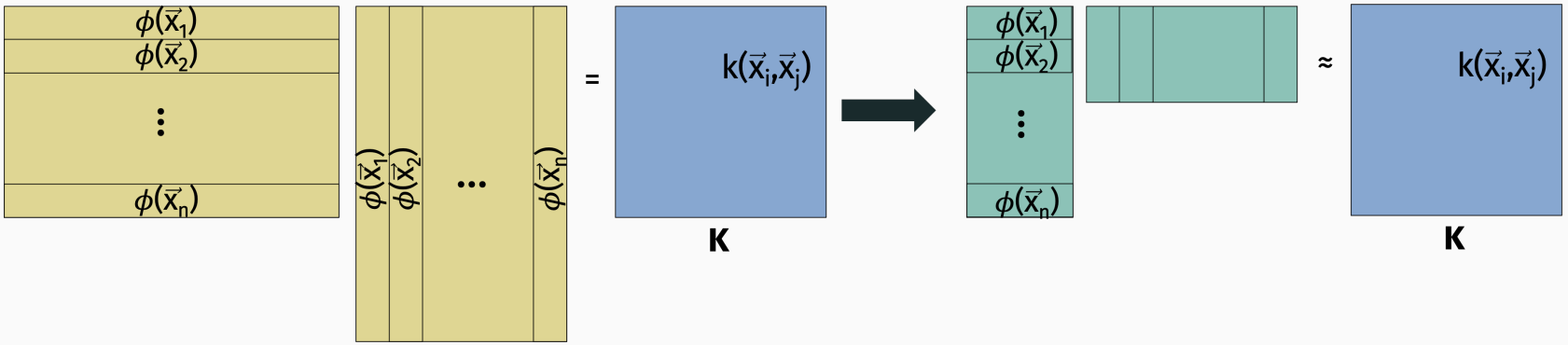


There is an inherent quadratic dependence on n in the computational and space complexity of kernel methods.

- $10,000$ 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:

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

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

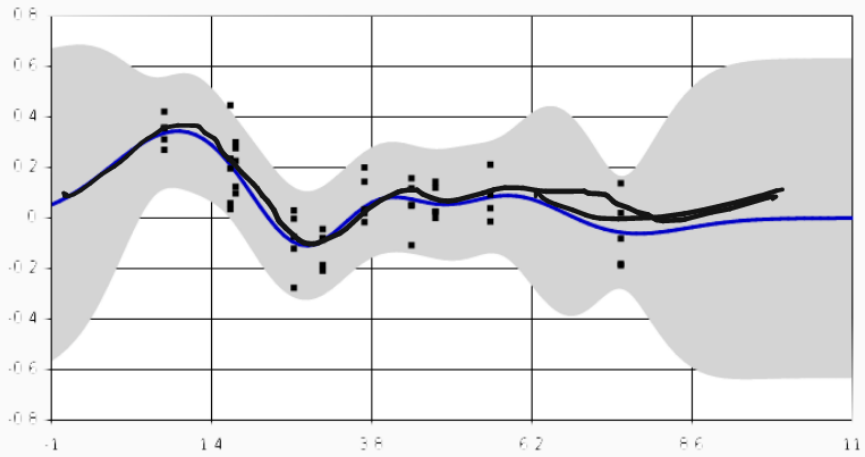
Prediction:

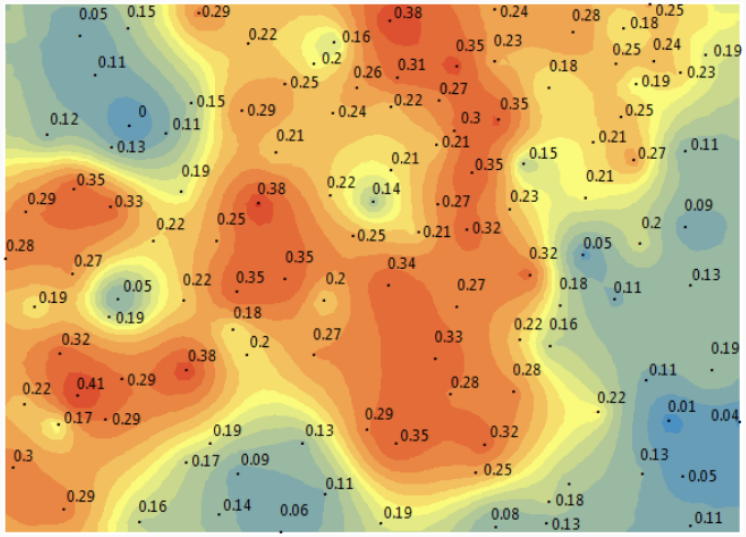
$$y_{new} = \sum_{i=1}^{n} \alpha_i \cdot k(\mathbf{x}_{new}, \mathbf{x}_i).$$

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.

Support Vector Machines

Today

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.

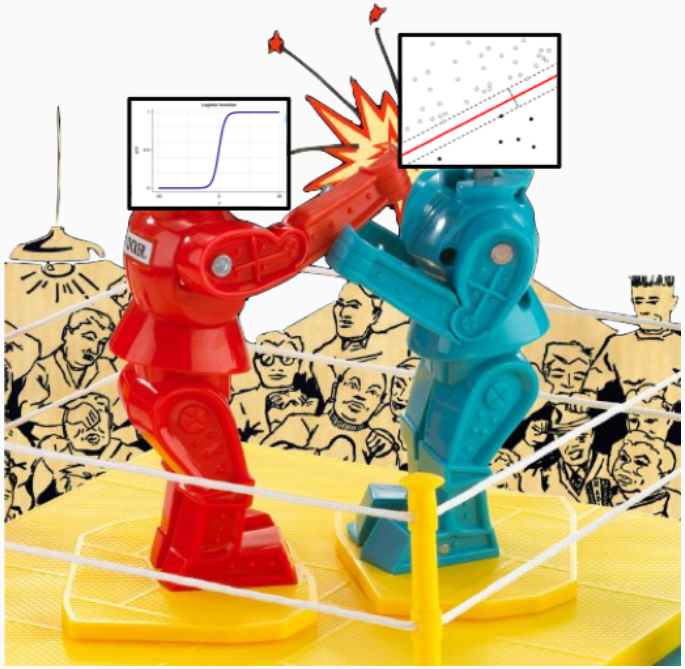
- Invented in 1963 by Alexey Chervonenkis and Vladimir Vapnik. Also founders of VC-theory.
- First combined with non-linear kernels in 1993.

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 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 the jury is still out...

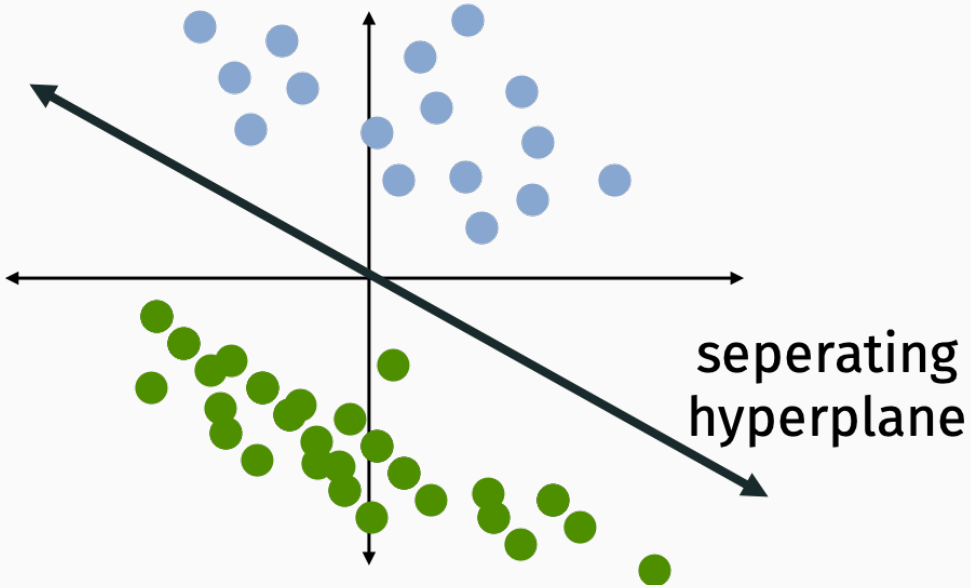
SVM's vs. logistic regression



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

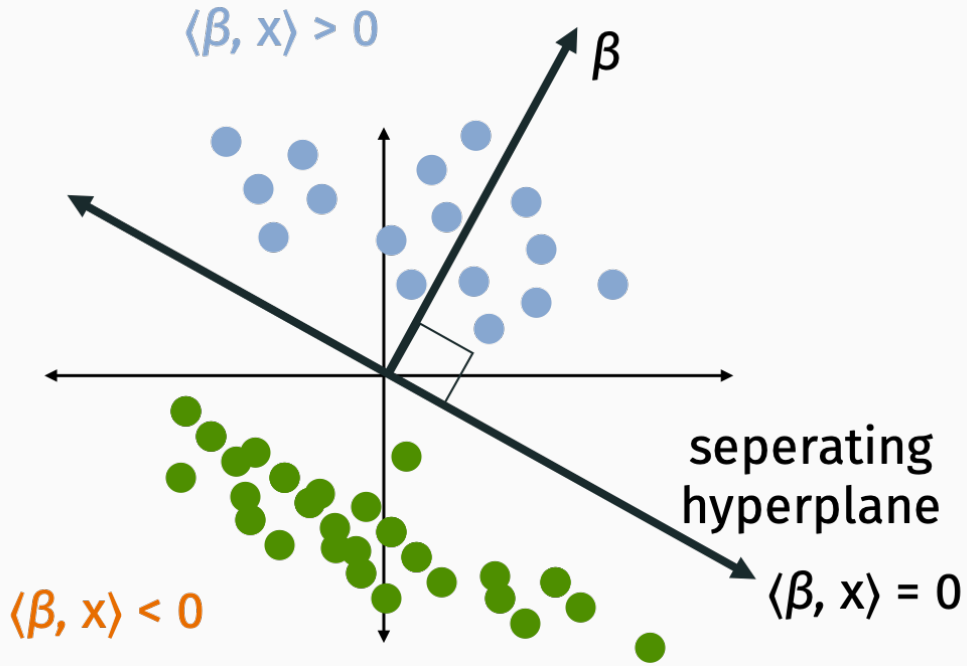
We call a dataset with binary labels linearly separable if it can be perfectly classified with a linear classifier:



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

Linearly separable data

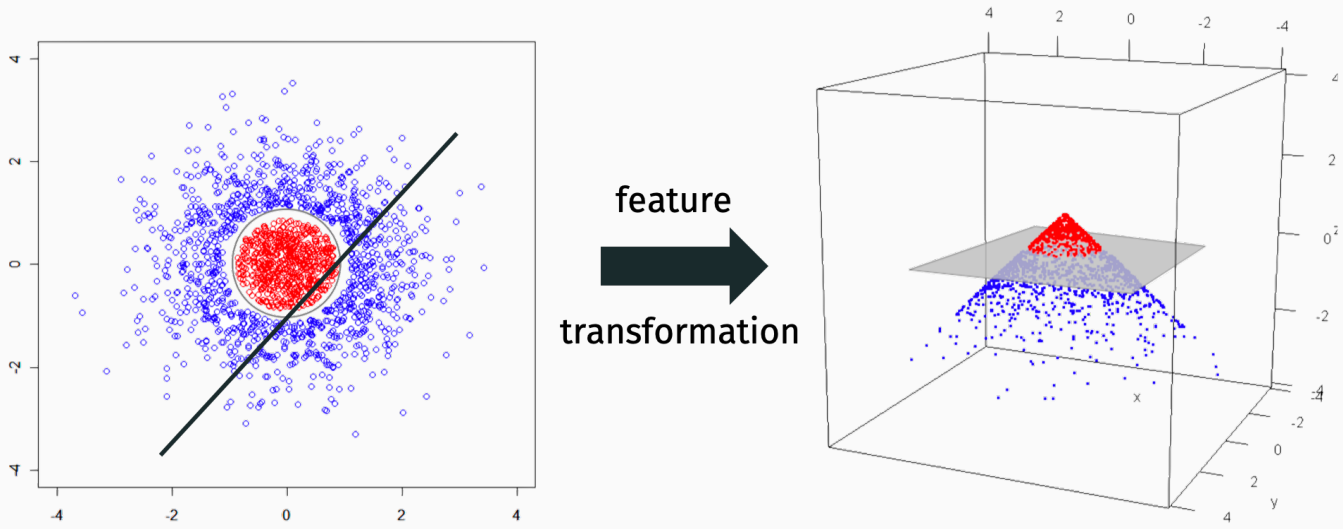
Formally, there exists a parameter β such that $\langle \beta, \mathbf{x} \rangle > 0$ for all $\mathbf{x}$ in class 1 and $\langle \beta, \mathbf{x} \rangle < 0$ for all $\mathbf{x}$ in class 0.



Note that if we multiply β 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

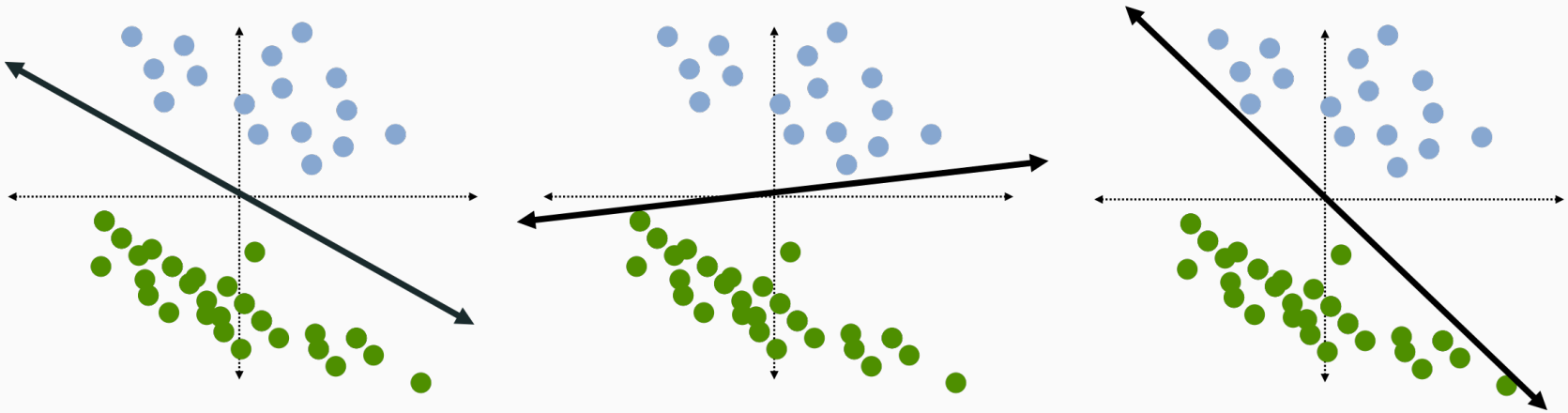
A data set might be linearly separable when using a non-linear kernel/feature transformation even if it is not separable in the original space.



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

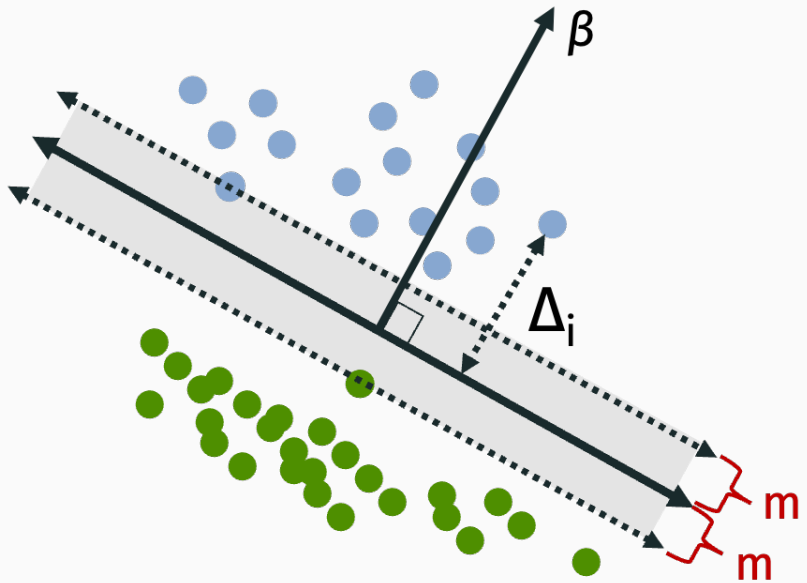
When data is linearly separable, there are typically multiple valid separating hyperplanes.



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

Margin

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



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

Margin

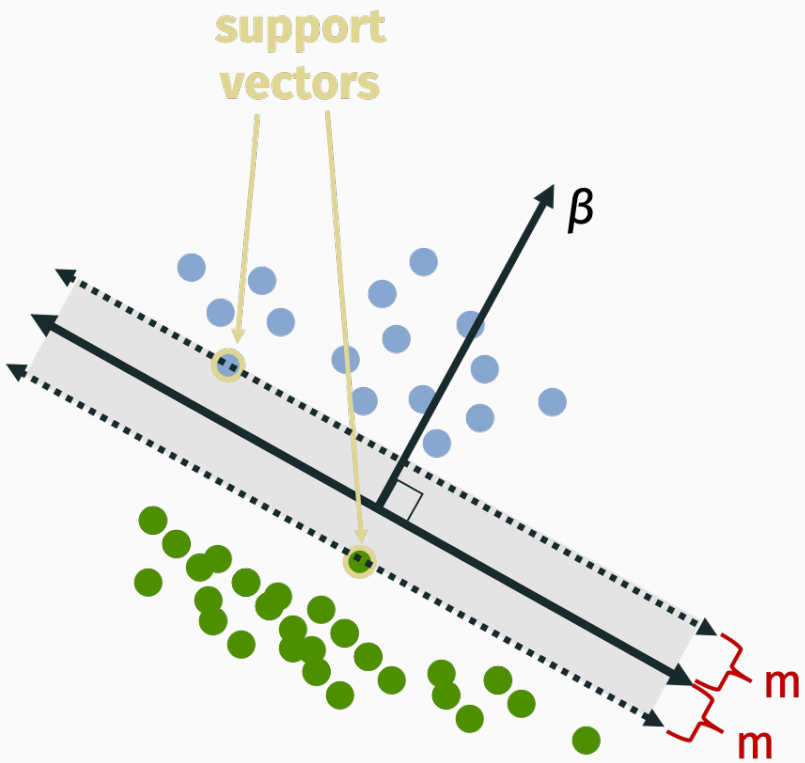
We have that $\mathbf{x}_i = \mathbf{v}_i + \mathbf{e}_i$ where $\mathbf{v}_i$ is parallel to $\boldsymbol{\beta}$ and $\mathbf{e}_i$ is perpendicular.

$$\Delta_i = \|\mathbf{v}_i\|_2 = \frac{1}{\|\mathbf{v}_i\|_2} \cdot \langle \mathbf{v}_i, \mathbf{v}_i \rangle = \frac{1}{\|\mathbf{v}_i\|_2} \cdot \frac{\|\mathbf{v}_i\|_2}{\|\boldsymbol{\beta}\|_2} \cdot |\langle \mathbf{v}_i, \boldsymbol{\beta}_i \rangle| = \frac{|\langle \mathbf{v}_i, \boldsymbol{\beta} \rangle|}{\|\boldsymbol{\beta}\|_2}.$$

Finally, we have that $\langle \mathbf{x}_i, \boldsymbol{\beta} \rangle = \langle \mathbf{v}_i, \boldsymbol{\beta} \rangle$ because $\langle \mathbf{e}_i, \boldsymbol{\beta} \rangle = 0$.

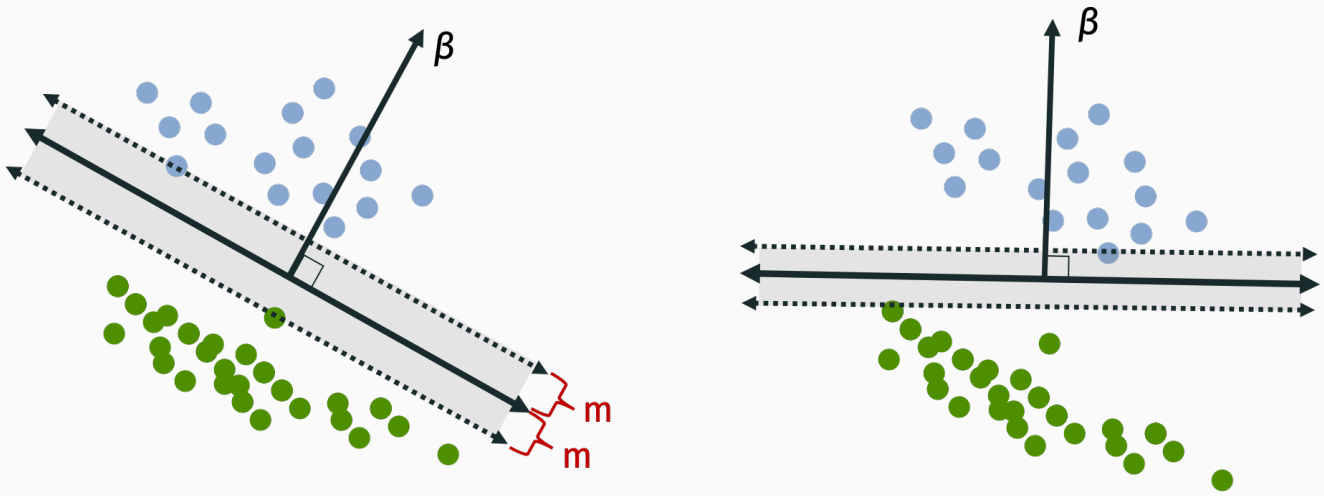
Support vector

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.

margin

Denote the maximum margin by m^*.

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

where $y_i = -1, 1$ depending on what class $\mathbf{x}_i$.[3]

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

Hard-margin svm

Equivalent formulation:

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

$$\begin{aligned} \frac{1}{m^*} &= \min_{\substack{\mathbf{v}:\|\mathbf{v}\|_2=1 \\ c}} c \qquad \text{subject to} \quad c \cdot y_i \cdot \langle \mathbf{x}_i, \mathbf{v} \rangle \geq 1 \text{ for all } i. \\ &= \min_{\substack{\mathbf{v}:\|\mathbf{v}\|_2=1 \\ c}} \|c \cdot \mathbf{v}\|_2 \quad \text{subject to} \quad y_i \cdot \langle \mathbf{x}_i, c \cdot \mathbf{v} \rangle \geq 1 \text{ for all } i. \end{aligned}$$

Hard-margin svm

Equivalent formulation:

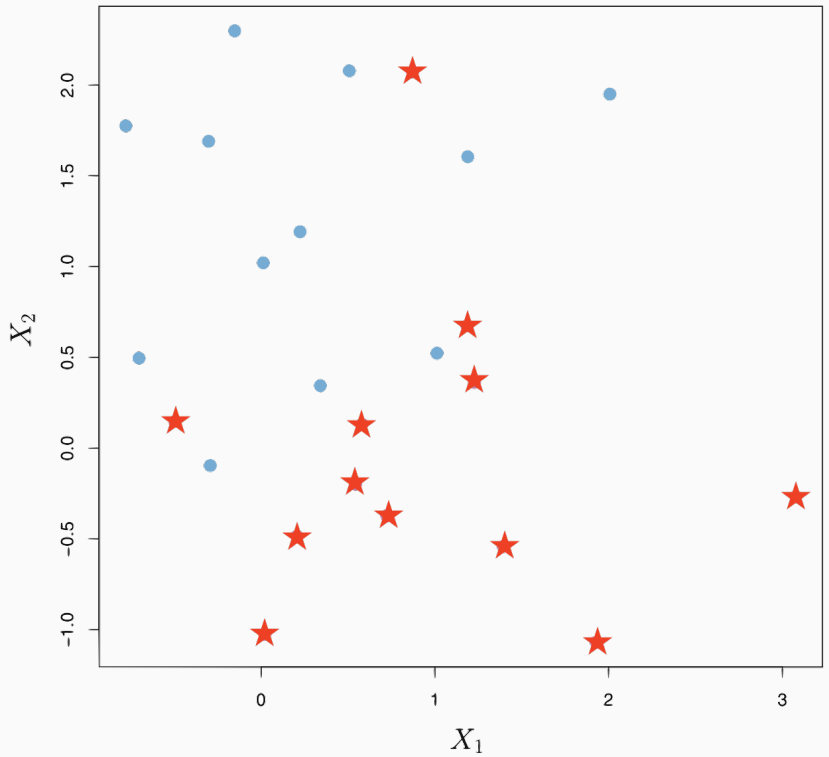
$$\min_{\boldsymbol{\beta}} \|\boldsymbol{\beta}\|_2^2 \qquad \text{subject to} \qquad y_i \cdot \langle \mathbf{x}_i, \boldsymbol{\beta} \rangle \geq 1 \text{ for all } i.$$

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

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:

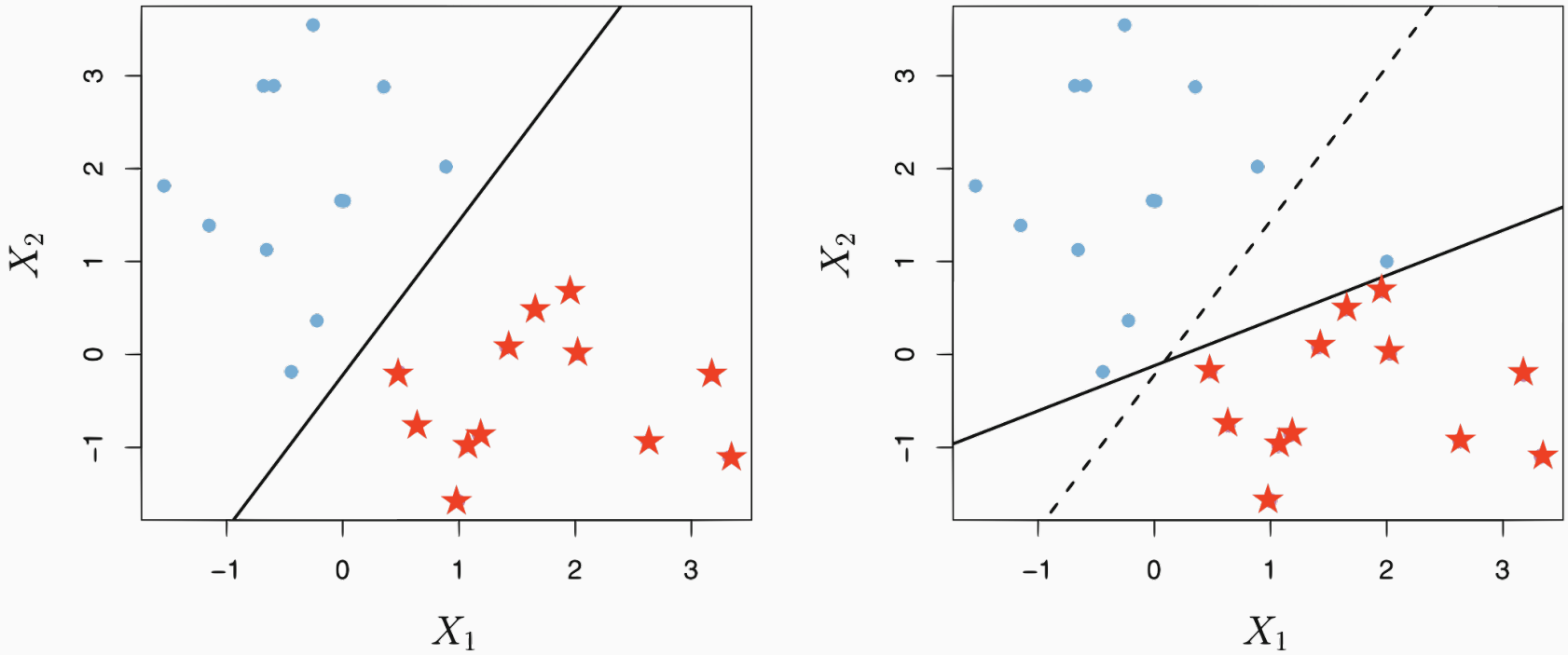


Data might not be linearly separable, in-which case the maximum margin classifier is not even defined.

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:

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

Soft margin objective:

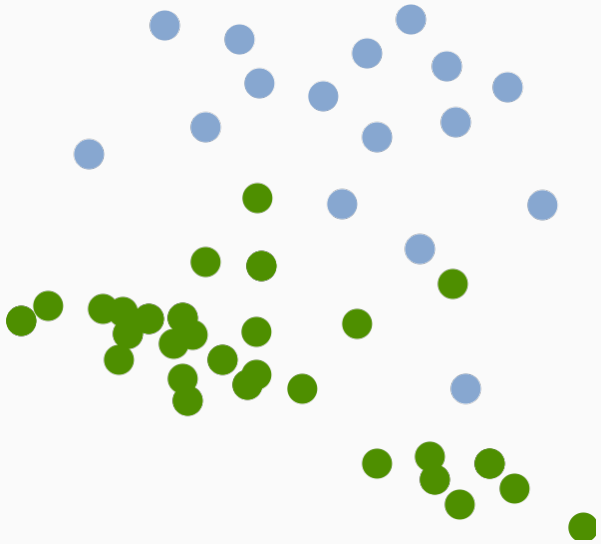
$$\min_{\beta} \|\beta\|_2^2 + C \sum_{i=1}^{n} \epsilon_i \quad \text{subject to} \quad y_i \cdot \langle \mathbf{x}_i, \beta \rangle \geq 1 - \epsilon_i \text{ for all } i.$$

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.

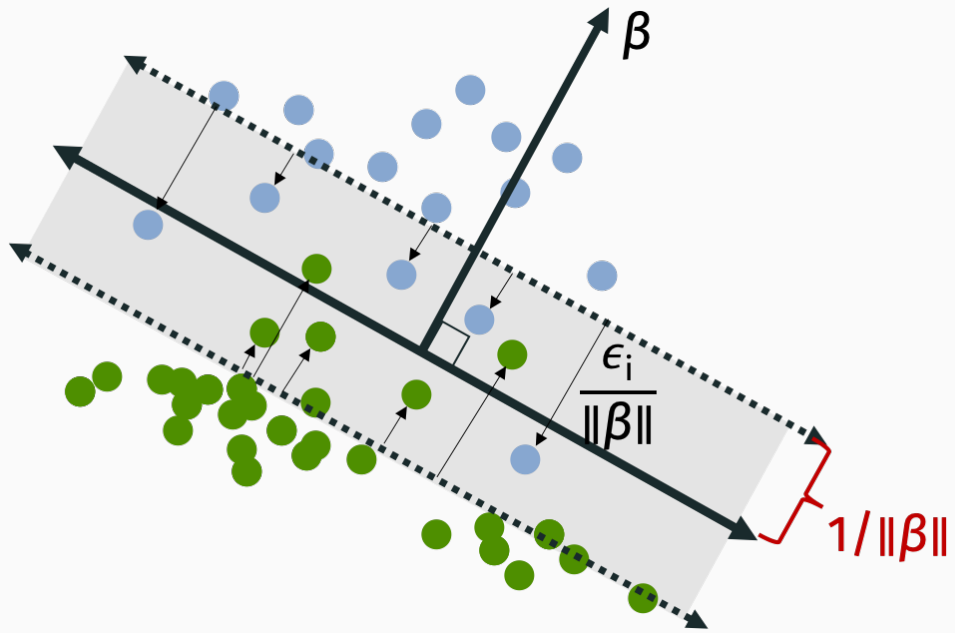
Soft-margin svm

Example of a non-separable problem:



Soft-margin svm

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

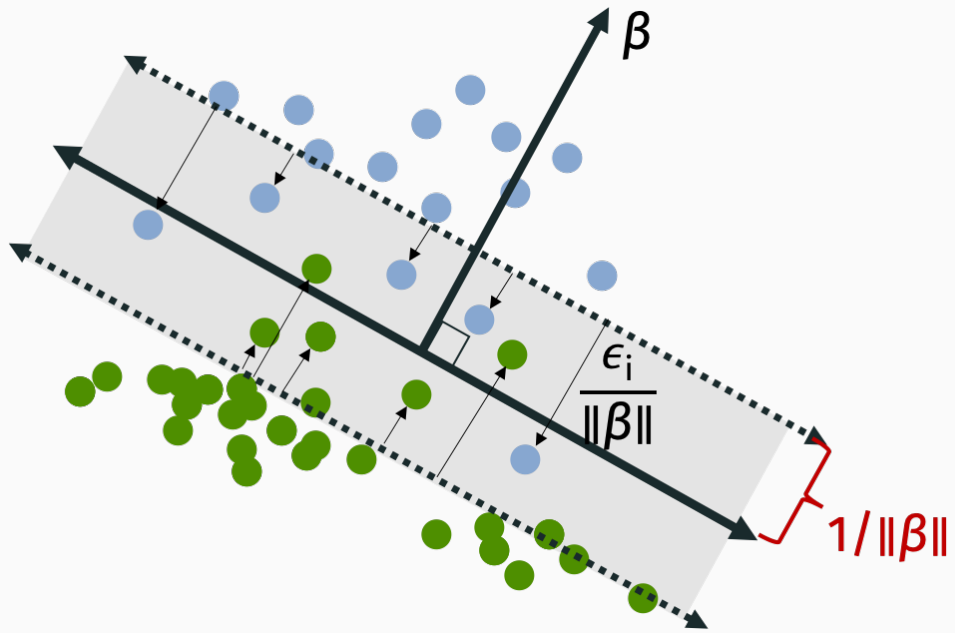


Soft margin objective:

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

Soft-margin svm

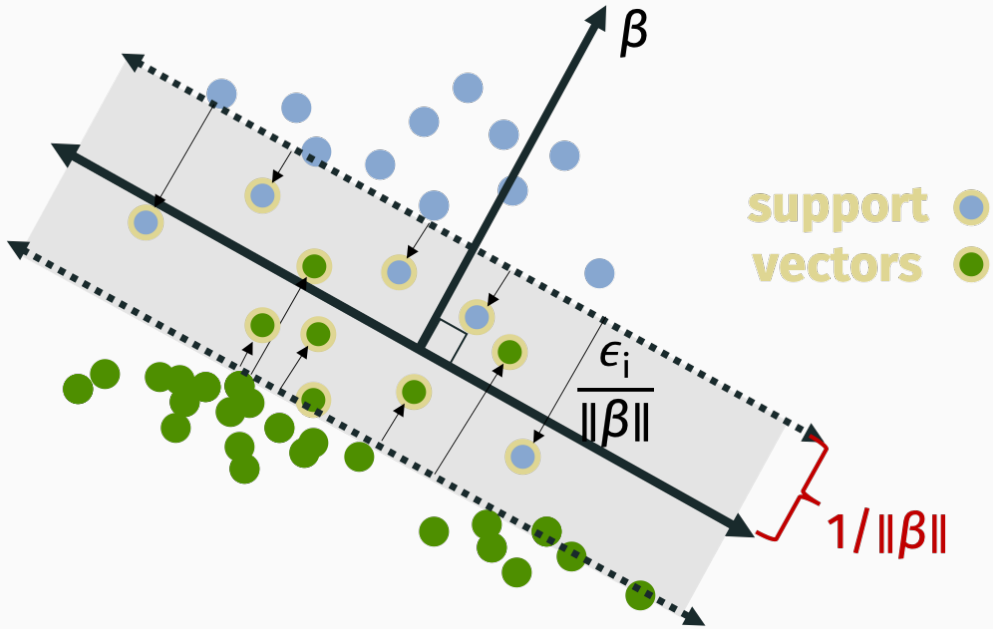
Recall that $\Delta_i = \frac{y_i \cdot \langle \mathbf{x}_i, \boldsymbol{\beta} \rangle}{\|\boldsymbol{\beta}\|_2}$.



Soft margin objective:

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

Soft-margin svm



Any $\mathbf{x}_i$ with a non-zero ϵ_i is a support vector.

Effect of c

Soft margin objective:

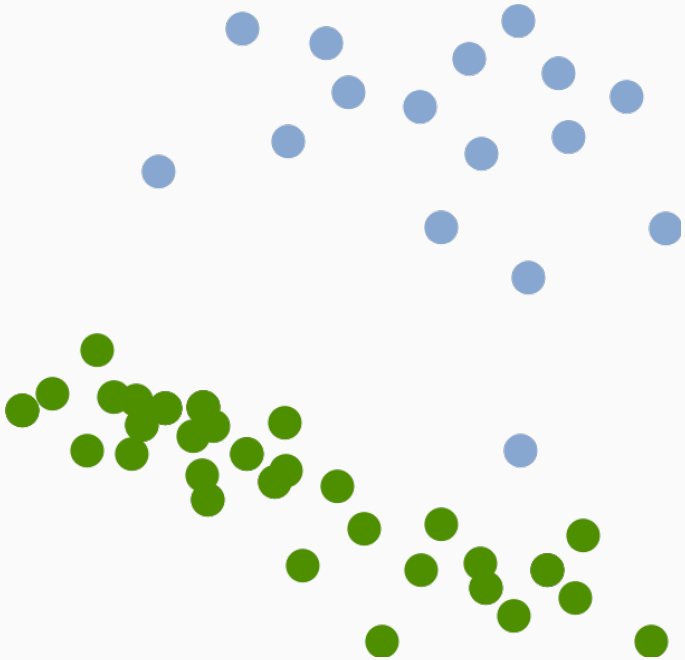
$$\min_{\boldsymbol{\beta}} \|\boldsymbol{\beta}\|_2^2 + C \sum_{i=1}^{n} \epsilon_i.$$

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

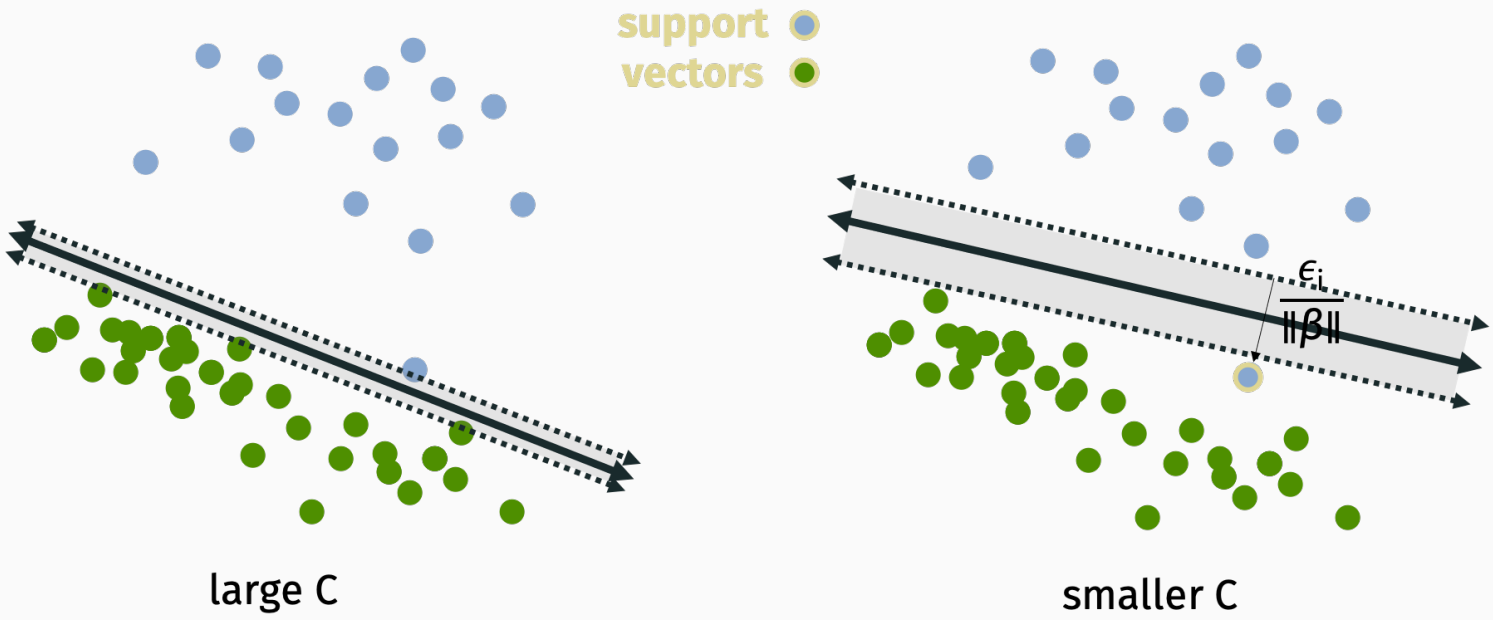
When data is linearly separable, as $C \to \infty$ we will always get a separating hyperplane. A smaller value of C might lead to a more robust solution.

Effect of c

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:

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

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

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

Comparison to logistic regression

Some basic transformations of the soft-margin objective:

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

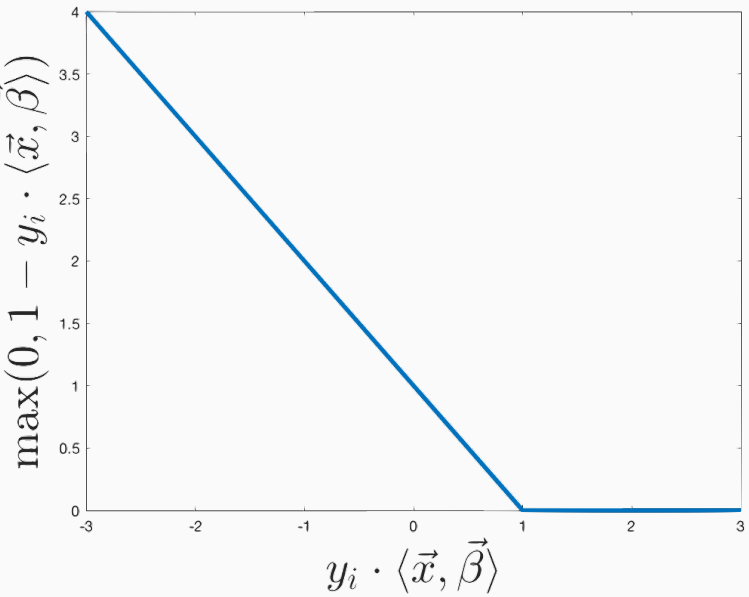
$$\min_{\beta} \|\beta\|_2^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i \cdot \langle \mathbf{x}_i, \beta \rangle).$$

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

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:

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

Logistic loss

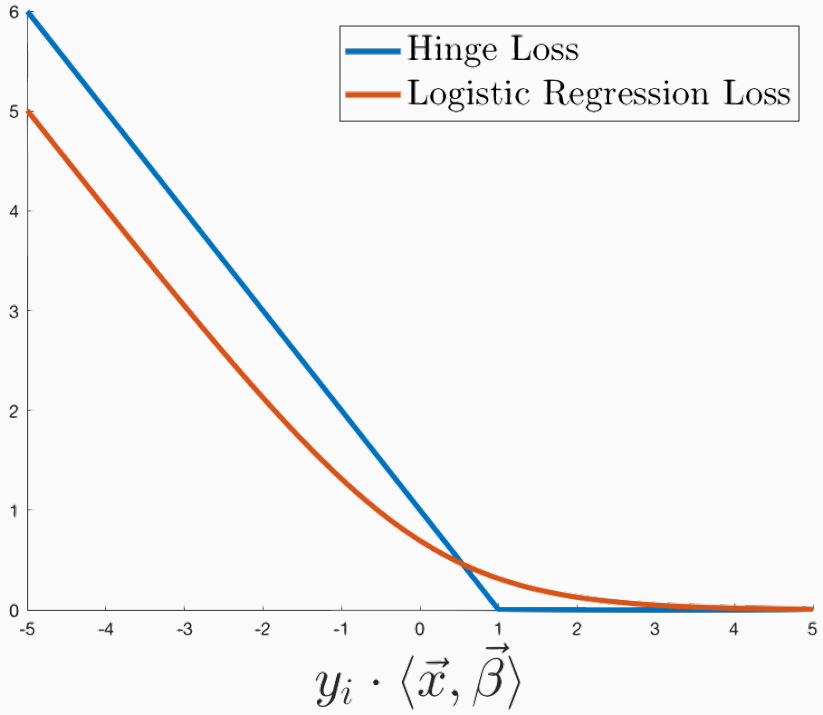
Recall the logistic loss for $y_i \in \{0, 1\}$:

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

Comparison of SVM to logistic regression

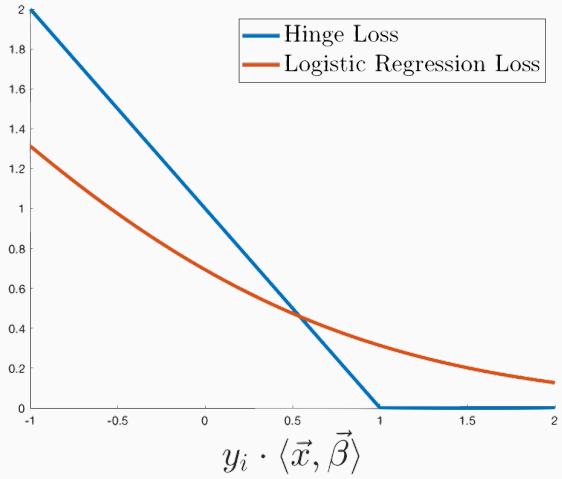
Compare this to the logistic regression loss reformulated for $y_i \in \{-1, 1\}$):

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



Comparison to logistic regression

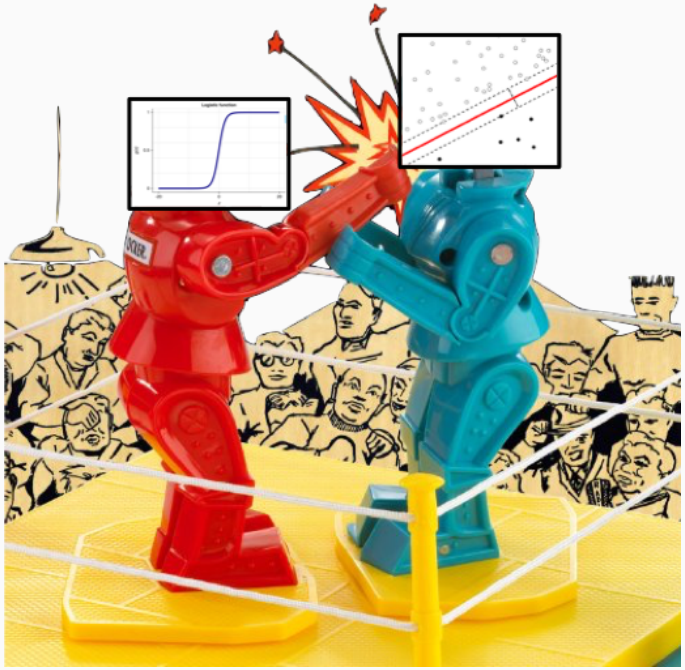
So, in the end, the function minimized when finding β for the standard **soft-margin SVM** is very similar to the objective function minimized when finding β using **logistic regression with ℓ_2 regularization.** Sort of...



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

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