Machine Learning

CMPT 726

Mo Chen
SFU School of Computing Science

2022-11-10

Lagrangian Duality

Slater's Condition

Slater's Condition:

If $f(\vec{\theta})$ and $g_i(\vec{\theta})$ are convex in $\vec{\theta}$ $\forall i \in \{1, \dots, k\}$ and $h_i(\vec{\theta})$ is linear in $\vec{\theta}$ $\forall i \in \{1, \dots, l\}$,and there exists a point $\vec{\theta}_0$ such that $g_i(\vec{\theta}) < 0$ $\forall i \in \{1, \dots, k\}$ and $h_i(\vec{\theta}) = 0$ $\forall i \in \{1, \dots, l\}$, then strong duality holds.

Strict inequality!

In words:

(1) Convex objective

(2) Convex inequality constraints

(3) Linear equality constraints

(4) Strictly feasible solution

Slater's Condition in SVMs

Recall primal form of SVMs: $\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2$ s.t. $y_i(\vec{w}^\top \vec{x}_i - b) \geq 1 \ \forall i$

Rewrite as: $\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2$ subject to $1 - y_i(\vec{w}^\top \vec{x}_i - b) \leq 0 \ \forall i$

Let's check if Slater's condition holds:

(1) Convex objective

Observe that the objective is a quadratic form: $\frac{1}{2}\|\vec{w}\|_2^2 = \frac{1}{2}\vec{w}^\top \vec{w} = \frac{1}{2}\vec{w}^\top I \vec{w} = \vec{w}^\top \left(\frac{1}{2}I\right)\vec{w}$

$\frac{1}{2}I = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix}$ .The eigenvalues are both $1/2$,and so $\frac{1}{2}I \succ 0$ .Hence the objective is strictly convex.

Slater’s Condition in SVMs

Recall primal form of SVMs: $\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2$ s.t. $y_i(\vec{w}^\top \vec{x}_i - b) \geq 1 \; \forall i$

Rewrite as: $\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2$ subject to $1 - y_i(\vec{w}^\top \vec{x}_i - b) \leq 0 \; \forall i$

Let’s check if Slater’s condition holds:

(2) Convex inequality constraints

For the constraint: $\frac{\partial}{\partial \vec{w}}\left(1 - y_i(\vec{w}^\top \vec{x}_i - b)\right) = -y_i \vec{x}_i$ and $\frac{\partial}{\partial b}\left(1 - y_i(\vec{w}^\top \vec{x}_i - b)\right) = y_i$

$$\frac{\partial^2}{\partial \begin{pmatrix} \vec{w} \\ b \end{pmatrix} \partial \begin{pmatrix} \vec{w} \\ b \end{pmatrix}^\top}\left(1 - y_i(\vec{w}^\top \vec{x}_i - b)\right) = \begin{pmatrix} \frac{\partial^2 \mathcal{L}}{\partial \vec{w} \partial \vec{w}^\top}\left(1 - y_i(\vec{w}^\top \vec{x}_i - b)\right) & \frac{\partial^2 \mathcal{L}}{\partial b \partial \vec{w}^\top}\left(1 - y_i(\vec{w}^\top \vec{x}_i - b)\right) \\ \frac{\partial^2 \mathcal{L}}{\partial \vec{w} \partial b}\left(1 - y_i(\vec{w}^\top \vec{x}_i - b)\right) & \frac{\partial^2 \mathcal{L}}{\partial b \partial b}\left(1 - y_i(\vec{w}^\top \vec{x}_i - b)\right) \end{pmatrix} = \begin{pmatrix} 0 & \vec{0} \\ \vec{0}^\top & 0 \end{pmatrix} \succcurlyeq 0$$

(We’ve actually shown something stronger than convex - when the Hessian is zero, the constraint is linear)

Slater’s Condition in SVMs

Recall primal form of SVMs: $\min\limits_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2$ s.t. $y_i(\vec{w}^\top \vec{x}_i - b) \geq 1 \ \forall i$

Rewrite as: $\min\limits_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2$ subject to $1 - y_i(\vec{w}^\top \vec{x}_i - b) \leq 0 \ \forall i$

Let’s check if Slater’s condition holds:

(3) Linear equality constraints

No equality constraints! So no need to check.

Slater’s Condition in SVMs

Recall primal form of SVMs: $\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2$ s.t. $y_i(\vec{w}^\top \vec{x}_i - b) \geq 1\ \forall i$

Rewrite as: $\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2$ subject to $1 - y_i(\vec{w}^\top \vec{x}_i - b) \leq 0\ \forall i$

Let’s check if Slater’s condition holds:

(4) Strictly feasible solution

Recall that the constraint says that the distance of each training data point to the hyperplane should be at least the margin (which must be positive), provided that we set $\|\vec{w}\|_2 = \frac{1}{m}$.

If the dataset is linearly separable, can make the margin small enough (or equivalently $\|\vec{w}\|_2^2$ large enough) for the constraint to be strictly satisfied.

Equivalence of Primal and Dual

When strong duality holds, $\min_{\vec{\theta}} \max_{\vec{\lambda},\vec{\nu}:\lambda_i \geq 0} \mathcal{L}(\vec{\theta},\vec{\lambda},\vec{\nu}) = p^* = d^* = \max_{\vec{\lambda},\vec{\nu}:\lambda_i \geq 0} \min_{\vec{\theta}} \mathcal{L}\left(\vec{\theta},\vec{\lambda},\vec{\nu}\right)$, and there is a solution $\left(\vec{\theta}^*,\vec{\lambda}^*,\vec{\nu}^*\right)$ that is optimal for both the primal and dual problems.

So, solving the primal problem is equivalent to solving the dual problem.

In the case of SVMs, this means we can equivalently solve the dual problem:

$$\max_{\vec{\lambda}} \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \vec{x}_j^{\top} \vec{x}_i \text{ subject to } \sum_{i=1}^{N} \lambda_i y_i = 0 \text{ and } \lambda_i \geq 0 \ \forall i$$

This gives us the optimal dual variables $\vec{\lambda}^*$, but not the optimal primal variables $\vec{\theta}^* = \begin{pmatrix} \vec{w}^* \\ b^* \end{pmatrix}$.

To find $\vec{\theta}^*$, we need to solve the primal problem, i.e.: $\min_{\vec{\theta}} \max_{\vec{\lambda},\vec{\nu}:\lambda_i \geq 0} \mathcal{L}(\vec{\theta},\vec{\lambda},\vec{\nu}) = \min_{\vec{\theta}} \mathcal{L}\left(\vec{\theta},\vec{\lambda}^*,\vec{\nu}^*\right)$.

Deriving Primal Solution from Dual Solution

In the case of SVMs, we solve $\min_{\vec{w},b} \mathcal{L}\left(\vec{w}, b, \vec{\lambda}^*\right)$.

$$\mathcal{L}\left(\vec{w}, b, \vec{\lambda}^*\right) = \frac{1}{2}\|\vec{w}\|_2^2 + \sum_{i=1}^{N} \lambda_i^*\left(1 - y_i(\vec{w}^\top \vec{x}_i - b)\right) = \frac{1}{2}\vec{w}^\top\vec{w} + \sum_{i=1}^{N}(\lambda_i^* - \lambda_i^* y_i \vec{w}^\top \vec{x}_i + \lambda_i^* y_i b)$$

Recall: $\frac{\partial\left(\vec{x}^\top A\vec{x}\right)}{\partial\vec{x}} = (A + A^\top)\vec{x}$ and $\frac{\partial\left(\vec{a}^\top\vec{x}\right)}{\partial\vec{x}} = \vec{a}$

$$0 = \frac{\partial\mathcal{L}}{\partial\vec{w}} = \frac{1}{2}(I + I^\top)\vec{w} - \sum_{i=1}^{N}\lambda_i^* y_i \vec{x}_i = \frac{1}{2}(2I)\vec{w} - \sum_{i=1}^{N}\lambda_i^* y_i \vec{x}_i = \vec{w} - \sum_{i=1}^{N}\lambda_i^* y_i \vec{x}_i \Longrightarrow \vec{w}^* = \sum_{i=1}^{N}\lambda_i^* y_i \vec{x}_i$$

(This was the same as the derivation of the optimum of the inner optimization problem in the dual)

Deriving Primal Solution from Dual Solution

To find b^*, we use the constraints in the primal problem, i.e.:$y_i(\vec{w}^\top \vec{x}_i - b) \geq 1 \ \forall i$ .Recall that these are margin constraints after setting $||\vec{w}||_2 = \frac{1}{m}$, where m is the margin.

Let's consider the cases where $y_i = 1$ and $y_i = -1$ separately:

$\vec{w}^\top \vec{x}_i - b \geq 1 \ \forall i \text{ such that } y_i = 1$ $\quad -\vec{w}^\top \vec{x}_i + b \geq 1 \ \forall i \text{ such that } y_i = -1$

Recall: The margin is the distance from hyperplane to the closest data point. So the distance of some data points to the hyperplane must be exactly the margin (these data points are said to lie **on the margin**). Hence,

$$\min_{i:y_i=1} \left\{\vec{w}^\top \vec{x}_i - b\right\} = 1, \qquad \min_{i:y_i=-1} \left\{-\vec{w}^\top \vec{x}_i + b\right\} = 1$$

Deriving Primal Solution from Dual Solution

$$\min_{i:y_i=1}\{\vec{w}^\top\vec{x}_i - b\} = 1, \qquad \min_{i:y_i=-1}\{-\vec{w}^\top\vec{x}_i + b\} = 1$$
$$\min_{i:y_i=1}\{\vec{w}^\top\vec{x}_i\} - b = 1, \qquad \min_{i:y_i=-1}\{-\vec{w}^\top\vec{x}_i\} + b = 1$$
$$\min_{i:y_i=1}\{\vec{w}^\top\vec{x}_i\} - b = 1, \qquad -\max_{i:y_i=-1}\{\vec{w}^\top\vec{x}_i\} + b = 1$$

Now we subtract the equation on the left from the equation on the right:

$$\left(-\max_{i:y_i=-1}\{\vec{w}^\top\vec{x}_i\} + b\right) - \left(\min_{i:y_i=1}\{\vec{w}^\top\vec{x}_i\} - b\right) = 0$$
$$-\max_{i:y_i=-1}\{\vec{w}^\top\vec{x}_i\} + b - \min_{i:y_i=1}\{\vec{w}^\top\vec{x}_i\} + b = 0$$
$$2b - \max_{i:y_i=-1}\{\vec{w}^\top\vec{x}_i\} - \min_{i:y_i=1}\{\vec{w}^\top\vec{x}_i\} = 0$$
$$b = \frac{\max_{i:y_i=-1}\{\vec{w}^\top\vec{x}_i\} + \min_{i:y_i=1}\{\vec{w}^\top\vec{x}_i\}}{2}$$

Deriving Primal Solution from Dual Solution

$$b = \frac{\max\limits_{i:y_i=-1}\{\vec{w}^\top \vec{x}_i\} + \min\limits_{i:y_i=1}\{\vec{w}^\top \vec{x}_i\}}{2}$$

This formula characterizes the optimal b for any choice of $\vec{w}$.

To find the optimal b overall, we can simply apply the formula to the optimal $\vec{w}$, denoted as $\vec{w}^*$:

$$b^* = \frac{\max\limits_{i:y_i=-1}\left\{{\vec{w}^*}^\top \vec{x}_i\right\} + \min\limits_{i:y_i=1}\left\{{\vec{w}^*}^\top \vec{x}_i\right\}}{2}$$

Primal vs. Dual of SVMs

Since solving the dual problem is equivalent to solving the primal problem, we can choose either approach. Which should we pick?

Recall the primal and dual forms of the SVM:

- Primal: $\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2$ s.t. $y_i(\vec{w}^\top \vec{x}_i - b) \geq 1 \; \forall i$
- Dual: $\max_{\vec{\lambda}} \sum_{i=1}^N \lambda_i - \frac{1}{2}\sum_{i=1}^N \sum_{j=1}^N \lambda_i \lambda_j y_i y_j \vec{x}_j^\top \vec{x}_i$ s.t. $\sum_{i=1}^N \lambda_i y_i = 0 \text{ and } \lambda_i \geq 0 \; \forall i$

The primal optimizes over $n+1$ variables, whereas the dual optimizes over N variables, where n and N denote the dimensionality and the number of data points respectively.

When $n \ll N$ (dimensionality much less than the number of data points), more efficient to solve the primal. Otherwise, more efficient to solve the dual.

Later will see another reason to solve the dual.

Karush-Kuhn-Tucker (KKT) Conditions

When strong duality holds, the optimal solution $\left(\vec{\theta}^*, \vec{\lambda}^*, \vec{\nu}^*\right)$ to both the primal and dual problems must satisfy the Karush-Kuhn-Tucker (KKT) conditions:

$$g_i\left(\vec{\theta}^*\right) \leq 0 \ \forall i \in \{1, \ldots, k\}$$
$$h_i\left(\vec{\theta}^*\right) = 0 \ \forall i \in \{1, \ldots, l\}$$

Primal Feasibility

$$\lambda_i^* \geq 0 \ \forall i \in \{1, \ldots, k\}$$

Dual Feasibility

$$\frac{\partial f}{\partial \vec{\theta}}\left(\vec{\theta}^*\right) + \sum_{i=1}^{k} \lambda_i^* \frac{\partial g_i}{\partial \vec{\theta}}\left(\vec{\theta}^*\right) + \sum_{i=1}^{l} \nu_i^* \frac{\partial h_i}{\partial \vec{\theta}}\left(\vec{\theta}^*\right) = 0$$

Stationarity

$$\lambda_i^* g_i\left(\vec{\theta}^*\right) = 0 \ \forall i \in \{1, \ldots, k\}$$

Complementary Slackness

Most important/useful

Complementary Slackness

Complementary slackness says that $\lambda_i^* g_i\left(\vec{\theta}^*\right) = 0 \ \forall i \in \{1, \ldots, k\}$ .So,

- $\lambda_i^* > 0$ implies that $g_i\left(\vec{\theta}^*\right) = 0$
  - If a dual variable for an inequality constraint is positive, the constraint must be an equality.
- $g_i\left(\vec{\theta}^*\right) < 0$ implies that $\lambda_i^* = 0$
  - If an inequality constraint is strict, its corresponding dual variable must be zero.

Note: When $\lambda_i^* = 0$ or $g_i(\vec{\theta}^*) = 0$, nothing is implied; the other factor can be zero or non-zero.

Complementary Slackness in SVMs

Recall the primal and dual forms of the SVM:

- Primal: $\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2$ s.t. $y_i(\vec{w}^\top \vec{x}_i - b) \geq 1 \; \forall i$
- Dual: $\max_{\vec{\lambda}} \sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \vec{x}_j^\top \vec{x}_i$ s.t. $\sum_{i=1}^{N} \lambda_i y_i = 0$ and $\lambda_i \geq 0 \; \forall i$

Complementary slackness says that $\lambda_i^* g_i\left(\vec{\theta}^*\right) = 0 \; \forall i \in \{1, \dots, k\}$

In the case of SVM, $g_i\left(\vec{\theta}^*\right) = 1 - y_i(\vec{w}^{*\top}\vec{x}_i - b^*)$. So:

- Training data points whose dual variables are positive must lie on the margin.
- Training data points outside the margin must have zero dual variables.

Support Vectors

Recall the primal solution of the SVM in terms of the dual solution:

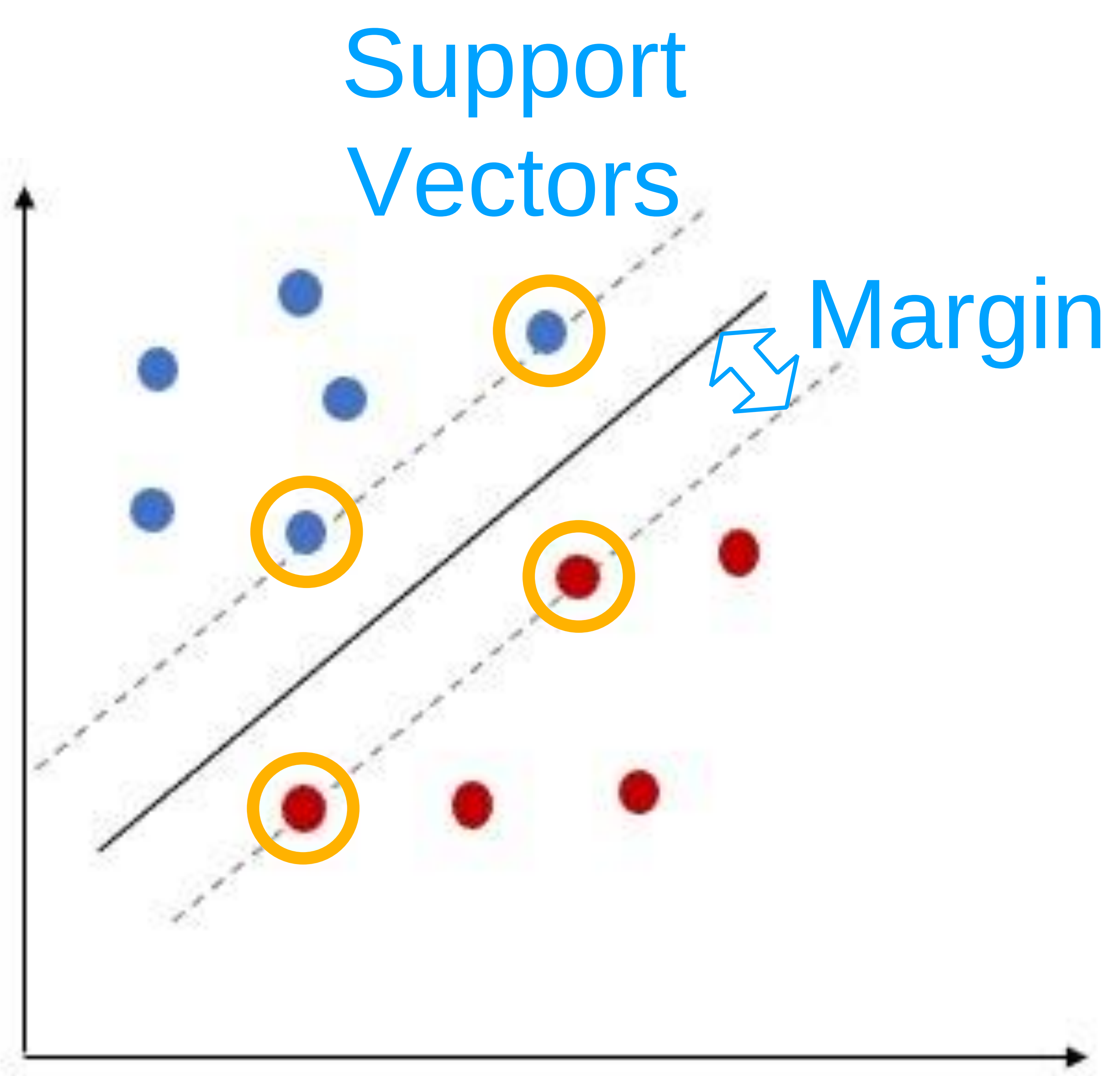
$$\vec{w}^* = \sum_{i=1}^{N} \lambda_i^* y_i \vec{x}_i \, , \qquad b^* = \frac{\max_{i:y_i=-1} \left\{ {\vec{w}^*}^\top \vec{x}_i \right\} + \min_{i:y_i=1} \left\{ {\vec{w}^*}^\top \vec{x}_i \right\}}{2}$$

Observe that only non-zero dual variables λ_i^* affect $\vec{w}^*$, which in turn affects b^*.

Hence, the optimal hyperplane can only be affected by data points on the margin.

- These data points are the support vectors.
- By complementary slackness, they must lie on the margin.

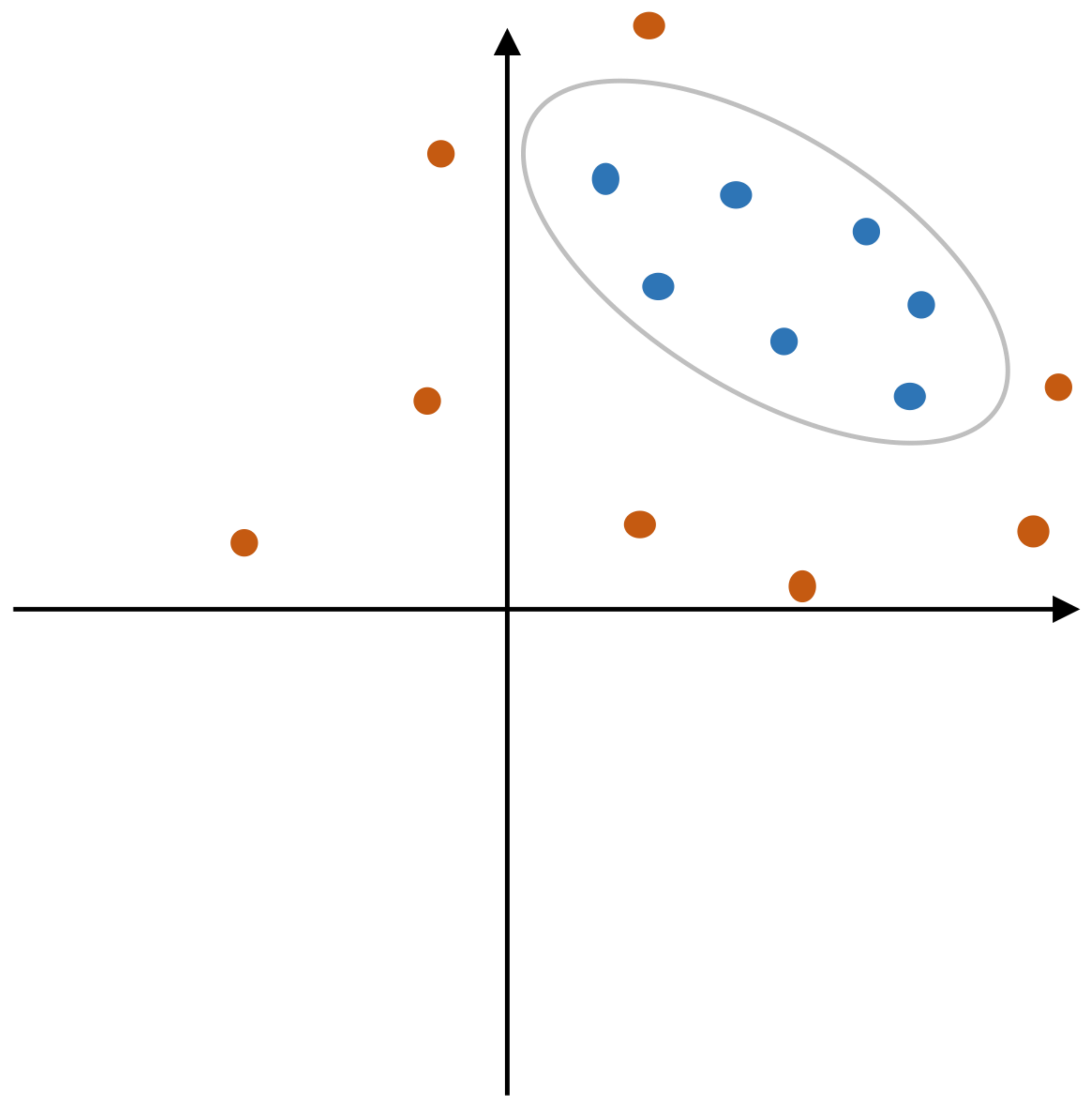
Data points outside the margin do not affect the optimal hyperplane.



Features

What if the dataset is not linearly separable?

Often the classes can be separated by a curve.



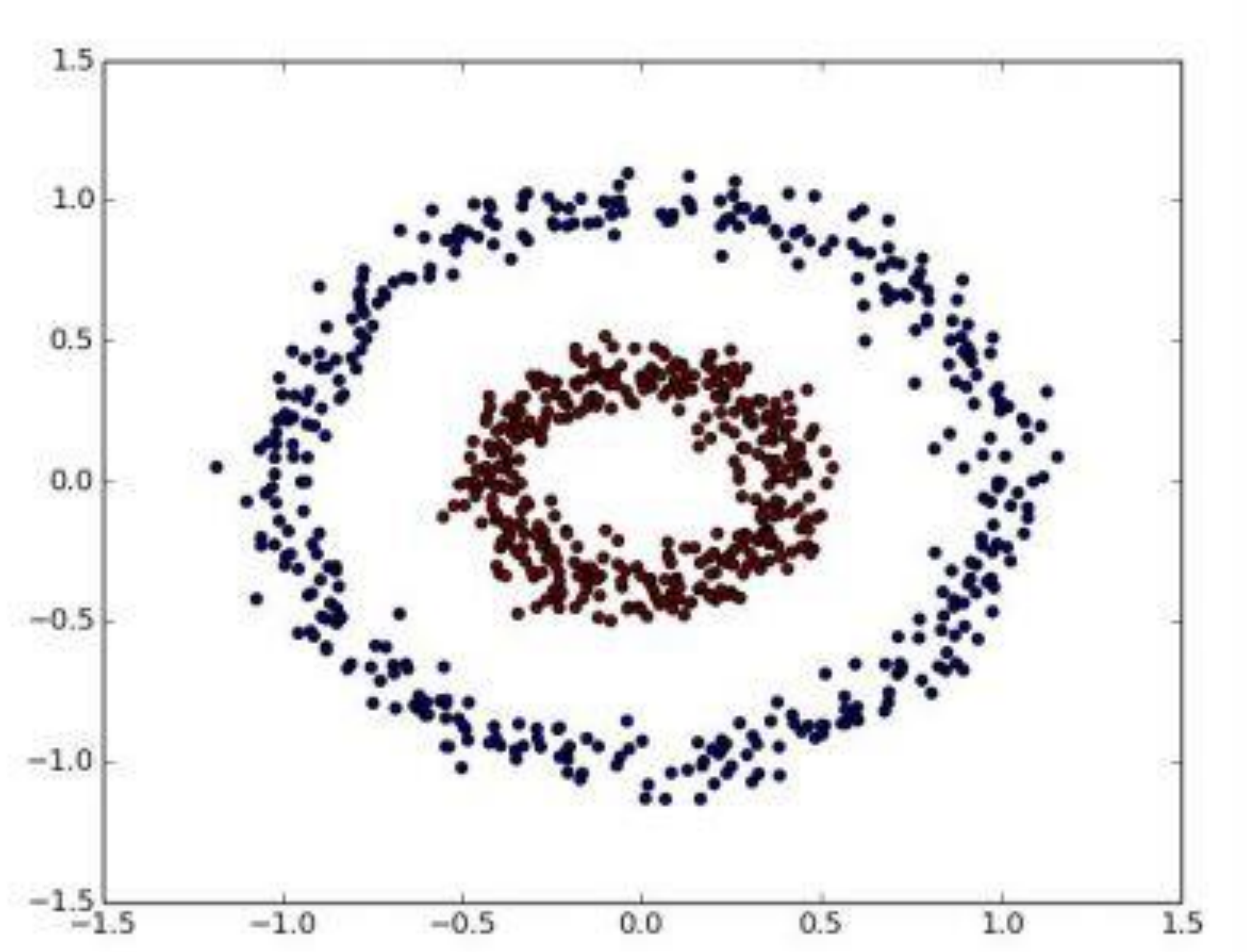
Features

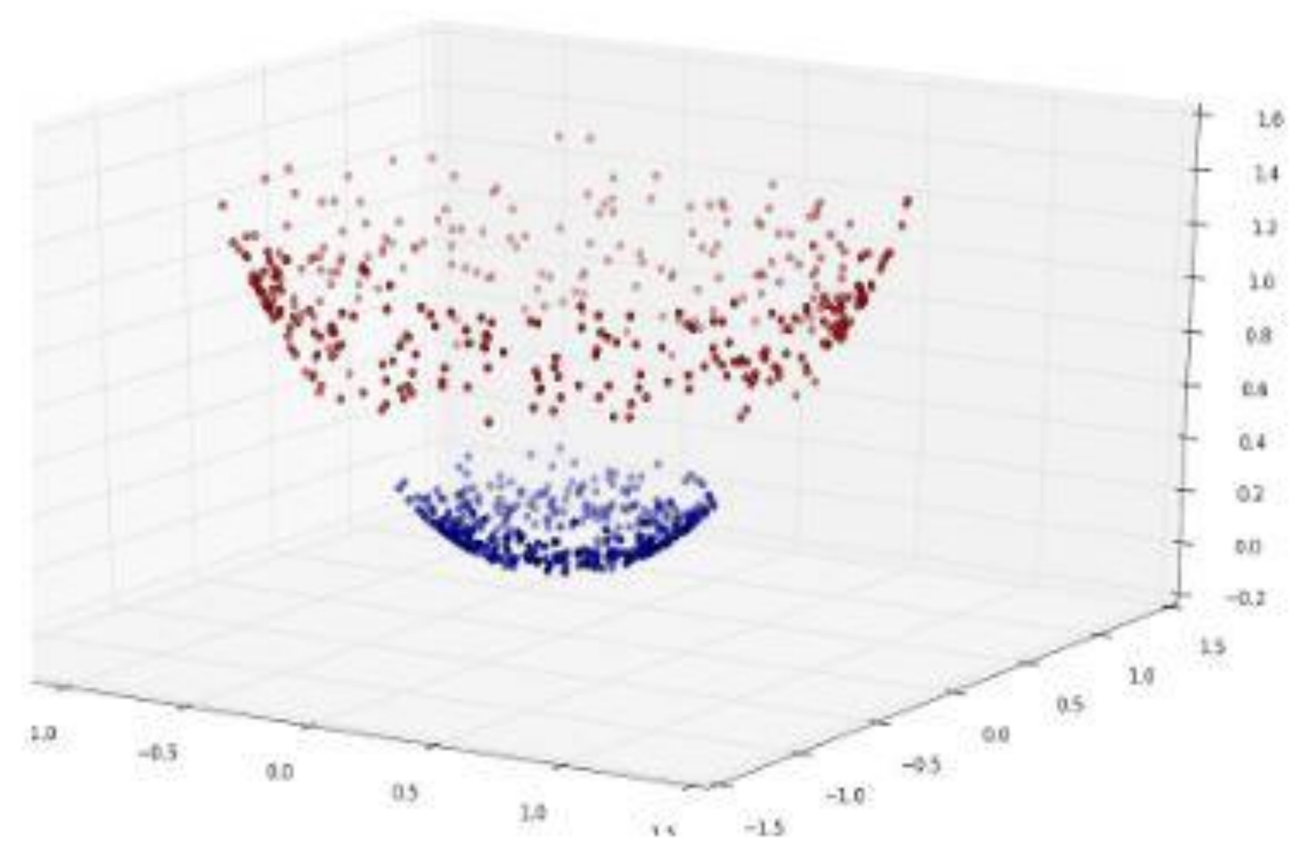
What if the dataset is not linearly separable?

Often the classes can be separated by a curve.

Recall linear regression on polynomial features: even though the model was linear, we were able to get predictions that are not linear in the raw input.

We can apply the same idea to SVMs to make the decision boundary non-linear.





Credit: Suriya Narayanan

Features

Example: Polynomial features up to degree 2

$$\phi(\vec{x}) = (\underbrace{1}_{\text{degree 0}} \quad \underbrace{x_1 \quad \cdots \quad x_n}_{\text{degree 1}} \quad \underbrace{x_1^2 \quad x_1x_2 \quad \cdots \quad x_1x_n \quad x_2^2 \quad x_2x_3 \quad \cdots \quad x_n^2}_{\text{degree 2 (don't forget the cross terms!)}})^\top$$

We can replace the raw input $\vec{x}_i$ with the features of raw inputs $\phi(\vec{x}_i)$ in the SVM:

Primal: $\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2$ s.t. $y_i(\vec{w}^\top \phi(\vec{x}_i) - b) \geq 1 \; \forall i$

Dual: $\max_{\vec{\lambda}} \sum_{i=1}^N \lambda_i - \frac{1}{2}\sum_{i=1}^N \sum_{j=1}^N \lambda_i\lambda_j y_i y_j \phi\left(\vec{x}_j\right)^\top \phi(\vec{x}_i)$ s.t. $\sum_{i=1}^N \lambda_i y_i = 0 \text{ and } \lambda_i \geq 0 \; \forall i$

Kernel Trick

Unfortunately, the dimensionality of the features becomes much higher than the dimensionality of the raw input. As a result, explicitly computing the features becomes computationally inefficient.

Let’s focus on the dual form:

$$\max_{\vec{\lambda}} \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \phi(\vec{x}_j)^\top \phi(\vec{x}_i) \ \text{s.t.} \sum_{i=1}^{N} \lambda_i y_i = 0 \text{ and } \lambda_i \geq 0 \ \forall i$$

It turns out there is a way of computing $\phi(\vec{x}_j)^\top \phi(\vec{x}_i)$ without explicitly computing the features $\phi(\vec{x}_i)$ .This is much more efficient and is known as the **kernel trick**.

Kernel Trick Example

Suppose we choose a variant of polynomial features up to degree 2:

$$\phi(\vec{x}) = \begin{pmatrix} 1 & \sqrt{2}x_1 & \cdots & \sqrt{2}x_n & x_1^2 & \sqrt{2}x_1x_2 & \cdots & \sqrt{2}x_1x_n & x_2^2 & \sqrt{2}x_2x_3 & \cdots & x_n^2 \end{pmatrix}^\top$$

(It differs from the usual polynomial features in that there is a coefficient of before the first-order and second-order cross terms)

$$\begin{aligned}
\phi(\vec{x})^\top\phi(\vec{y}) &= 1 + \sum_{i=1}^{n}\left(\sqrt{2}x_i\right)\left(\sqrt{2}y_i\right) + \sum_{i=1}^{n} x_i^2 y_i^2 + \sum_{i=1}^{n}\sum_{j=i+1}^{n}\left(\sqrt{2}x_ix_j\right)\left(\sqrt{2}y_iy_j\right) \\
&= 1 + \sum_{i=1}^{n}\left(\sqrt{2}x_i\right)\left(\sqrt{2}y_i\right) + \sum_{i=1}^{n} x_i^2 y_i^2 + \sum_{i=1}^{n}\sum_{j=i+1}^{n}\left(\sqrt{2}x_ix_j\right)\left(\sqrt{2}y_iy_j\right) \\
&= 1 + 2\sum_{i=1}^{n} x_iy_i + \sum_{i=1}^{n}(x_iy_i)^2 + 2\sum_{i=1}^{n}\sum_{j=i+1}^{n}(x_iy_i)\left(x_jy_j\right) \\
&= \left(1 + \sum_{i=1}^{n} x_iy_i\right)^2 = (1 + \vec{x}^\top\vec{y})^2 \coloneqq k(\vec{x},\vec{y})
\end{aligned}$$

So, instead of computing $\phi(\vec{x})$ and $\phi(\vec{y})$ explicitly, we can instead compute $k(\vec{x},\vec{y}) = (1 + \vec{x}^\top\vec{y})^2$, which is much more efficient to compute.

Kernels

The function $k(\vec{x}, \vec{y})$ is known as a **kernel function** or a **kernel** for short.

There are many possible kernel functions, each of which corresponds to a different choice of features $\phi(\vec{x})$.

Choosing features is equivalent to choosing kernels.

Computing the kernel allows us to avoid directly computing the features $\phi(\vec{x})$,which can be very high-dimensional and therefore inefficient or intractable to compute.

Kernels can even allow us to use infinite-dimensional features without needing to compute them!

Kernels

Common examples of kernels:

Linear kernel: $k(\vec{x}, \vec{y}) = \vec{x}^\top \vec{y} \Leftrightarrow \phi(\vec{x}) = \vec{x}$

Quadratic kernel: $k(\vec{x}, \vec{y}) = (1 + \vec{x}^\top \vec{y})^2$

$$\Leftrightarrow \phi(\vec{x}) = \begin{pmatrix} 1 & \sqrt{2}x_1 & \cdots & \sqrt{2}x_n & x_1^2 & \sqrt{2}x_1x_2 & \cdots & \sqrt{2}x_1x_n & x_2^2 & \sqrt{2}x_2x_3 & \cdots & x_n^2 \end{pmatrix}^\top$$

Polynomial kernel: $k(\vec{x}, \vec{y}) = (1 + \vec{x}^\top \vec{y})^d$ each dimension of $\phi(\vec{x})$ is a term in the binomial expansion of $(1 + x_1 + \cdots + x_n)^d$, where square root is taken on the coefficient.

Radial basis function (RBF) kernel:

$$k(\vec{x}, \vec{y}) = \exp\left(-\frac{1}{2}\|\vec{x} - \vec{y}\|_2^2\right) = \exp\left(-\frac{1}{2}\|\vec{x}\|_2^2\right)\exp\left(-\frac{1}{2}\|\vec{y}\|_2^2\right)\sum_{i=0}^{\infty}\frac{(\vec{x}^\top \vec{y})^i}{i!}$$

$\Leftrightarrow \phi(\vec{x})$ is the concatenation of an infinite number of vectors, whose dimensions are terms in the binomial expansion of $(x_1 + \cdots + x_n)^i$, where square root is taken on the product of the coefficient and $\exp(-\|\vec{x}\|_2^2)/(i!)$.

Kernels in SVMs

To use kernels in SVMs, we need to solve the dual problem:

$$\max_{\vec{\lambda}} \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \underbrace{\phi(\vec{x}_j)^\top \phi(\vec{x}_i)}_{k(\vec{x}_j, \vec{x}_i)} \text{ s.t. } \sum_{i=1}^{N} \lambda_i y_i = 0 \text{ and } \lambda_i \geq 0 \; \forall i$$

We must also be able to classify unseen test examples without directly computing features.

The predicted class label $\hat{y}_{\text{test}}$ is:

$$\hat{y}_{\text{test}} = \begin{cases} 1 & \vec{w}^{*\top} \phi(\vec{x}_{\text{test}}) - b^* > 0 \\ -1 & \vec{w}^{*\top} \phi(\vec{x}_{\text{test}}) - b^* < 0 \end{cases} := \begin{cases} 1 & g(\vec{x}_{\text{test}}) > 0 \\ -1 & g(\vec{x}_{\text{test}}) < 0 \end{cases}, \text{where } g(\vec{x}_{\text{test}}) = \vec{w}^{*\top} \phi(\vec{x}_{\text{test}}) - b^*$$

We can plug in the following formulas for the primal solution in terms of the dual solution into the above expression:

$$\vec{w}^* = \sum_{i=1}^{N} \lambda_i^* y_i \phi(\vec{x}_i), \qquad b^* = \frac{\max_{i:y=-1} \left\{ \vec{w}^{*\top} \phi(\vec{x}_i) \right\} + \min_{i:y=1} \left\{ \vec{w}^{*\top} \phi(\vec{x}_i) \right\}}{2}$$

Kernels in SVMs

$$g(\vec{x}_{\text{test}}) = \vec{w}^{*\top}\phi(\vec{x}_{\text{test}}) - b^* = \left(\sum_{i=1}^{N}\lambda_i^* y_i \phi(\vec{x}_i)\right)^{\top}\phi(\vec{x}_{\text{test}}) - b^* = \sum_{i=1}^{N}[\lambda_i^* y_i \underbrace{\phi(\vec{x}_i)^{\top}\phi(\vec{x}_{\text{test}})}_{k(\vec{x}_i,\vec{x}_{\text{test}})}] - b^*,$$

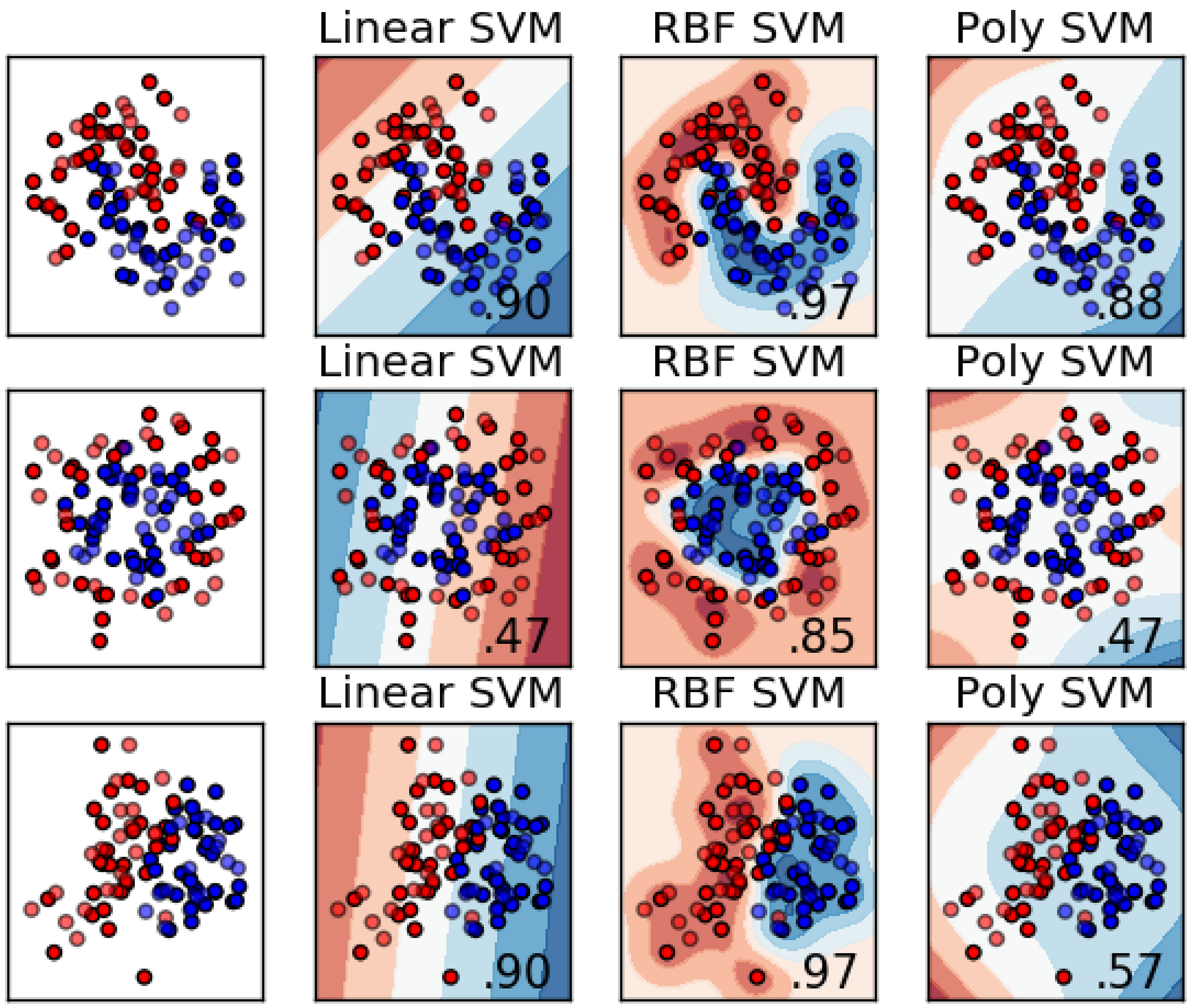
$$\text{where } b^* = \frac{1}{2}\max_{i:y=-1}\left\{\sum_{i=1}^{N}\lambda_i^* y_i \underbrace{\phi(\vec{x}_j)^{\top}\phi(\vec{x}_i)}_{k(\vec{x}_j,\vec{x}_i)}\right\} + \min_{i:y=1}\left\{\sum_{i=1}^{N}\lambda_i^* y_i \underbrace{\phi(\vec{x}_j)^{\top}\phi(\vec{x}_i)}_{k(\vec{x}_j,\vec{x}_i)}\right\}$$

Now, after rewriting, we can train and test without ever directly computing the features.

This is the second reason of why we might use the dual form:

Even if $n \ll N$, if the data is not linearly separable, we can use a non-linear kernel to implicitly replace the raw input with features, which will result in a non-linear decision boundary.

SVM Decision Boundaries



Credit: Fernando Wittmann

Mercer's Theorem

Can we come up with kernels without deriving what features they correspond to?

Yes! Mercer's theorem tells us whether a proposed kernel function actually corresponds to some choice of features. A kernel $k(\cdot,\cdot)$ is only considered **valid** if there is some $\phi(\cdot)$ such that $k(\vec{x},\vec{y}) = \phi(\vec{x})^\top \phi(\vec{y})$ for all $\vec{x}$ and $\vec{y}$.

Mercer's theorem tells us that a function $k(\cdot,\cdot)$ is a valid kernel if and only if:

(1) $k(\cdot,\cdot)$ is symmetric, i.e.: $k(\vec{x},\vec{y}) = k(\vec{y},\vec{x})$ for all $\vec{x},\vec{y}$

(2) for any $\vec{u}_1, \dots, \vec{u}_m$, the Gram matrix $\begin{pmatrix} k(\vec{u}_1,\vec{u}_1) & k(\vec{u}_1,\vec{u}_2) & \cdots & k(\vec{u}_1,\vec{u}_m) \\ k(\vec{u}_2,\vec{u}_1) & k(\vec{u}_2,\vec{u}_2) & \cdots & k(\vec{u}_2,\vec{u}_m) \\ \vdots & \vdots & \ddots & \vdots \\ k(\vec{u}_m,\vec{u}_1) & k(\vec{u}_m,\vec{u}_2) & \cdots & k(\vec{u}_m,\vec{u}_m) \end{pmatrix} \succcurlyeq 0$

Kernelization

This idea of kernels applies more generally beyond the context of SVMs.

Whenever we have a method that only depends on the inner products between different data points and not the data points themselves, we can apply the kernel trick.

- i.e.: We can replace all occurrences of inner products $\vec{x}_i^\top \vec{x}_j$ with $\phi(\vec{x}_i)^\top \phi(\vec{x}_j)$, which is by definition equal to $k(\vec{x}_i, \vec{x}_j)$.

This allows us to convert a linear model to a nonlinear model, while still largely preserving the computational efficiency of a linear model.

This process is known as **kernelization**.

Soft-Margin SVM

Vanilla (hard-margin) SVM:

$$\min_{\vec{w},b} \frac{1}{2} \|\vec{w}\|_2^2$$

subject to

$$y_i(\vec{w}^\top \vec{x}_i - b) \geq 1 \; \forall i$$

Soft-margin SVM:

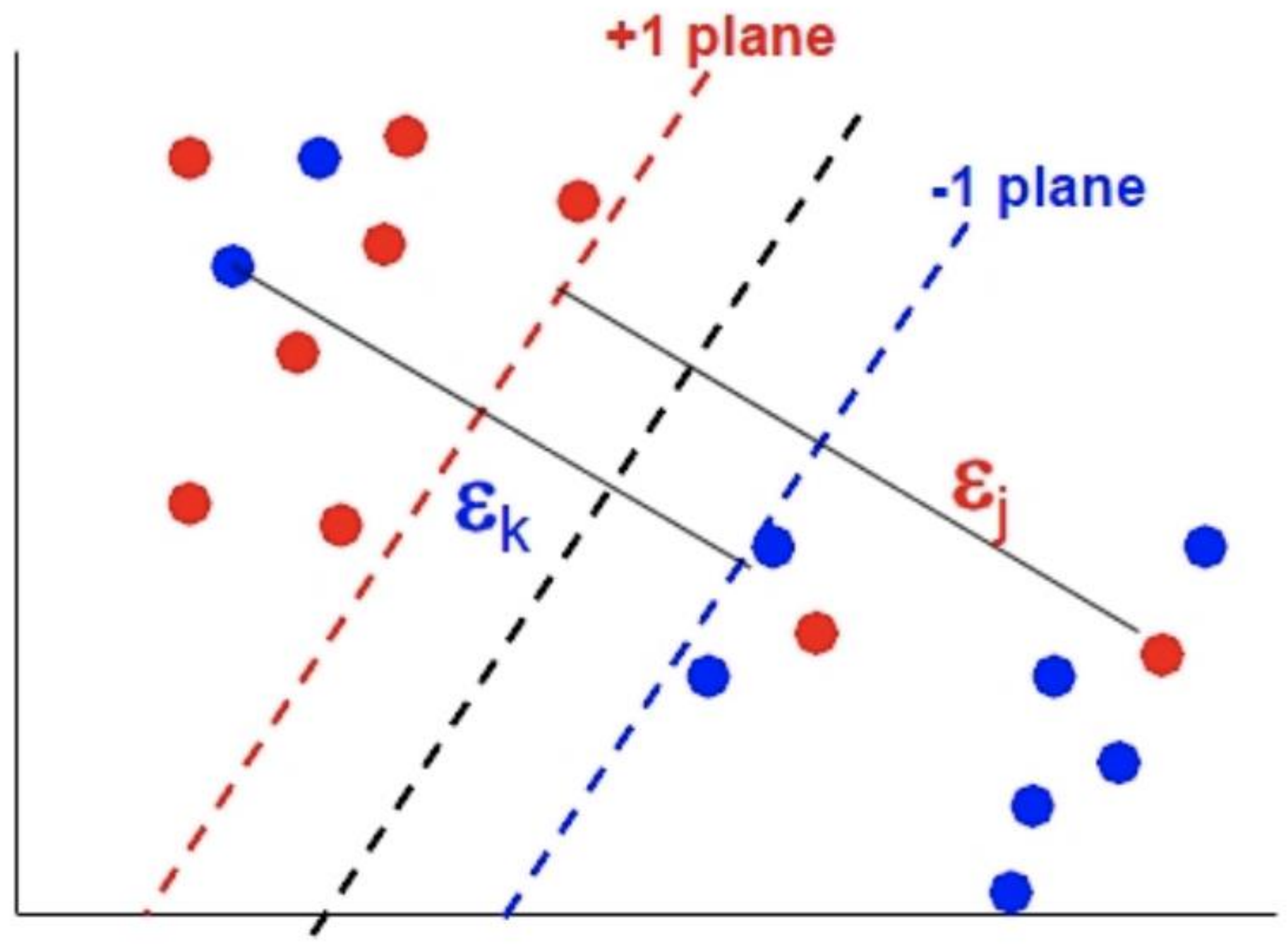
$$\min_{\vec{w},b} \frac{1}{2} \|\vec{w}\|_2^2 + \gamma \sum_{i=1}^{N} \xi_i$$

subject to

$$y_i(\vec{w}^\top \vec{x}_i - b) \geq 1 - \xi_i \;\; \forall i$$
$$\xi_i \geq 0 \;\; \forall i$$

ξ_i's are called **slack variables**.

Soft-Margin SVM



Credit: Rich Zemel, Raquel Urtasun and Sanja Fidler

Soft-margin SVM:

$$\min_{\vec{w},b} \frac{1}{2} \|\vec{w}\|_2^2 + \gamma \sum_{i=1}^{N} \xi_i$$

subject to

$$y_i(\vec{w}^\top \vec{x}_i - b) \geq 1 - \xi_i \ \ \forall i$$
$$\xi_i \geq 0 \ \ \forall i$$

ξ_i's are called **slack variables**.

Soft-Margin SVM

Alternate Formulation:

$$\min_{\vec{w},b} \frac{1}{2} \|\vec{w}\|_2^2 + \gamma \sum_{i=1}^{N} \xi_i$$

subject to

$\xi_i \geq 1 - y_i(\vec{w}^\top \vec{x}_i - b) \ \forall i$

$\xi_i \geq 0 \ \forall i$

Soft-margin SVM:

$$\min_{\vec{w},b} \frac{1}{2} \|\vec{w}\|_2^2 + \gamma \sum_{i=1}^{N} \xi_i$$

subject to

$y_i(\vec{w}^\top \vec{x}_i - b) \geq 1 - \xi_i \ \forall i$

$\xi_i \geq 0 \ \forall i$

ξ_i's are called **slack variables**.

Soft-Margin SVM

Alternate Formulation:

$$\min_{\vec{w},b} \frac{1}{2} \|\vec{w}\|_2^2 + \gamma \sum_{i=1}^{N} \xi_i$$

subject to

$\xi_i \geq \max\big(0, 1 - y_i(\vec{w}^\top \vec{x}_i - b)\big) \; \forall i$

Soft-margin SVM:

$$\min_{\vec{w},b} \frac{1}{2} \|\vec{w}\|_2^2 + \gamma \sum_{i=1}^{N} \xi_i$$

subject to

$y_i(\vec{w}^\top \vec{x}_i - b) \geq 1 - \xi_i \;\; \forall i$

$\xi_i \geq 0 \;\; \forall i$

ξ_i's are called **slack variables**.

Soft-Margin SVM

Alternate Formulation:

$$\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2 + \gamma \sum_{i=1}^{N} \max\left(0, 1 - y_i(\vec{w}^\top \vec{x}_i - b)\right)$$

Becomes an unconstrained objective!

Soft-margin SVM:

$$\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2 + \gamma \sum_{i=1}^{N} \xi_i$$

subject to

$$y_i(\vec{w}^\top \vec{x}_i - b) \geq 1 - \xi_i \ \ \forall i$$
$$\xi_i \geq 0 \ \ \forall i$$

ξ_i's are called **slack variables**.

Soft-Margin SVM

Alternate Formulation:

$$\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2 + \gamma \sum_{i=1}^{N} \max\big(0, 1 - y_i(\vec{w}^\top \vec{x}_i - b)\big)$$

$$\min_{\vec{w},b} \gamma \sum_{i=1}^{N} \max\big(0, 1 - y_i(\vec{w}^\top \vec{x}_i - b)\big) + \frac{1}{2}\|\vec{w}\|_2^2$$

Soft-margin SVM:

$$\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2 + \gamma \sum_{i=1}^{N} \xi_i$$

subject to

$$y_i(\vec{w}^\top \vec{x}_i - b) \geq 1 - \xi_i \;\; \forall i$$
$$\xi_i \geq 0 \;\; \forall i$$

ξ_i's are called **slack variables**.

Soft-Margin SVM

Alternate Formulation:

$$\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2 + \gamma \sum_{i=1}^{N} \max\big(0, 1 - y_i(\vec{w}^\top \vec{x}_i - b)\big)$$

$$\min_{\vec{w},b} \gamma \sum_{i=1}^{N} \max\big(0, 1 - y_i(\vec{w}^\top \vec{x}_i - b)\big) + \frac{1}{2}\|\vec{w}\|_2^2$$

$$\min_{\vec{w},b} \sum_{i=1}^{N} \max\big(0, 1 - y_i(\vec{w}^\top \vec{x}_i - b)\big) + \frac{1}{2\gamma}\|\vec{w}\|_2^2$$

Soft-margin SVM:

$$\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2 + \gamma \sum_{i=1}^{N} \xi_i$$

subject to

$$y_i(\vec{w}^\top \vec{x}_i - b) \geq 1 - \xi_i \ \ \forall i$$

$$\xi_i \geq 0 \ \ \forall i$$

ξ_i's are called **slack variables**.

Soft-Margin SVM

Alternate Formulation:

$$\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2 + \gamma \sum_{i=1}^{N} \max\big(0, 1 - y_i(\vec{w}^\top \vec{x}_i - b)\big)$$

$$\min_{\vec{w},b} \gamma \sum_{i=1}^{N} \max\big(0, 1 - y_i(\vec{w}^\top \vec{x}_i - b)\big) + \frac{1}{2}\|\vec{w}\|_2^2$$

$$\min_{\vec{w},b} \sum_{i=1}^{N} \max\big(0, 1 - y_i(\vec{w}^\top \vec{x}_i - b)\big) + \frac{1}{2\gamma}\|\vec{w}\|_2^2$$

$$\min_{\vec{w},b} \sum_{i=1}^{N} \max\big(0, 1 - y_i(\vec{w}^\top \vec{x}_i - b)\big) + \lambda\|\vec{w}\|_2^2$$

where $\lambda \coloneqq 1/2\gamma$

Soft-margin SVM:

$$\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2 + \gamma \sum_{i=1}^{N} \xi_i$$

subject to

$$y_i(\vec{w}^\top \vec{x}_i - b) \geq 1 - \xi_i \ \ \forall i$$

$$\xi_i \geq 0 \ \ \forall i$$

ξ_i's are called **slack variables**.

Soft-Margin SVM

Alternate Formulation:

$$\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2 + \gamma \sum_{i=1}^{N} \max\big(0, 1 - y_i(\vec{w}^\top \vec{x}_i - b)\big)$$

$$\min_{\vec{w},b} \gamma \sum_{i=1}^{N} \max\big(0, 1 - y_i(\vec{w}^\top \vec{x}_i - b)\big) + \frac{1}{2}\|\vec{w}\|_2^2$$

$$\min_{\vec{w},b} \sum_{i=1}^{N} \max\big(0, 1 - y_i(\vec{w}^\top \vec{x}_i - b)\big) + \frac{1}{2\gamma}\|\vec{w}\|_2^2$$

$$\min_{\vec{w},b} \sum_{i=1}^{N} \max\big(0, 1 - y_i(\vec{w}^\top \vec{x}_i - b)\big) + \lambda\|\vec{w}\|_2^2$$

where $\lambda \coloneqq 1/2\gamma$

$l(y, \hat{y}) = \max(0, 1 - y\hat{y})$is called a **hinge loss**.

We can rewrite the objective as:

$$\min_{\vec{w},b} \sum_{i=1}^{N} l\big(y_i, f(\vec{x}_i; \vec{w}, b)\big) + \lambda\|\vec{w}\|_2^2$$

where $f(\vec{x}; \vec{w}, b) = \vec{w}^\top \vec{x} - b$

Soft-Margin SVM

Alternate Formulation:

$$\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2 + \gamma \sum_{i=1}^{N} \max\big(0, 1 - y_i(\vec{w}^\top \vec{x}_i - b)\big)$$

$$\min_{\vec{w},b} \gamma \sum_{i=1}^{N} \max\big(0, 1 - y_i(\vec{w}^\top \vec{x}_i - b)\big) + \frac{1}{2}\|\vec{w}\|_2^2$$

$$\min_{\vec{w},b} \sum_{i=1}^{N} \max\big(0, 1 - y_i(\vec{w}^\top \vec{x}_i - b)\big) + \frac{1}{2\gamma}\|\vec{w}\|_2^2$$

$$\min_{\vec{w},b} \sum_{i=1}^{N} \max\left(0, 1 - y_i\left({\vec{w}^*}^\top \vec{x}_i - b^*\right)\right) + \lambda\|\vec{w}\|_2^2$$

where $\lambda \coloneqq 1/2\gamma$

$l(y, \hat{y}) = \max(0, 1 - y\hat{y})$is called a **hinge loss**.

We can rewrite the objective as:

$$\min_{\vec{w}} \sum_{i=1}^{N} l\big(y_i, f(\vec{x}_i; \vec{w})\big) + \lambda\|\vec{w}\|_2^2$$

where $f(\vec{x}; \vec{w}) = \vec{w}^\top \vec{x}$

Here we let $\vec{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_n \\ b \end{pmatrix}$ and $\vec{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \\ 1 \end{pmatrix}$

Soft-Margin SVM

Compare to the ridge regression objective:

$$\min_{\vec{w}} \sum_{i=1}^{N} (y_i - \vec{w}^\top \vec{x}_i)^2 + \lambda \|\vec{w}\|_2^2$$

Consider $l(y, \hat{y}) = (y - \hat{y})^2$m which is called the **square loss**.

We can write the objective as

$$\min_{\vec{w}} \sum_{i=1}^{N} l\big(y_i, f(\vec{x}_i; \vec{w})\big) + \lambda \|\vec{w}\|_2^2$$

where $f(\vec{x}; \vec{w}) = \vec{w}^\top \vec{x}$

$l(y, \hat{y}) = \max(0, 1 - y\hat{y})$ is called a **hinge loss**.

We can rewrite the objective as:

$$\min_{\vec{w}} \sum_{i=1}^{N} l\big(y_i, f(\vec{x}_i; \vec{w})\big) + \lambda \|\vec{w}\|_2^2$$

where $f(\vec{x}; \vec{w}) = \vec{w}^\top \vec{x}$

Here we let $\vec{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_n \\ b \end{pmatrix}$ and $\vec{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \\ 1 \end{pmatrix}$

Ridge Regression vs. Soft-Margin SVM

So we can compare ridge regression to soft-margin SVM in a unified framework.

Both optimize an objective function of the following form:

$$\min_{\vec{w}} \sum_{i=1}^{N} \ell\big(y_i, f(\vec{x}_i; \vec{w})\big) + \lambda \|\vec{w}\|_2^2, \text{where } f(\vec{x}; \vec{w}) = \vec{w}^\top \vec{x}$$

In ridge regression (square loss): $\ell(y, \hat{y}) = (y - \hat{y})^2$

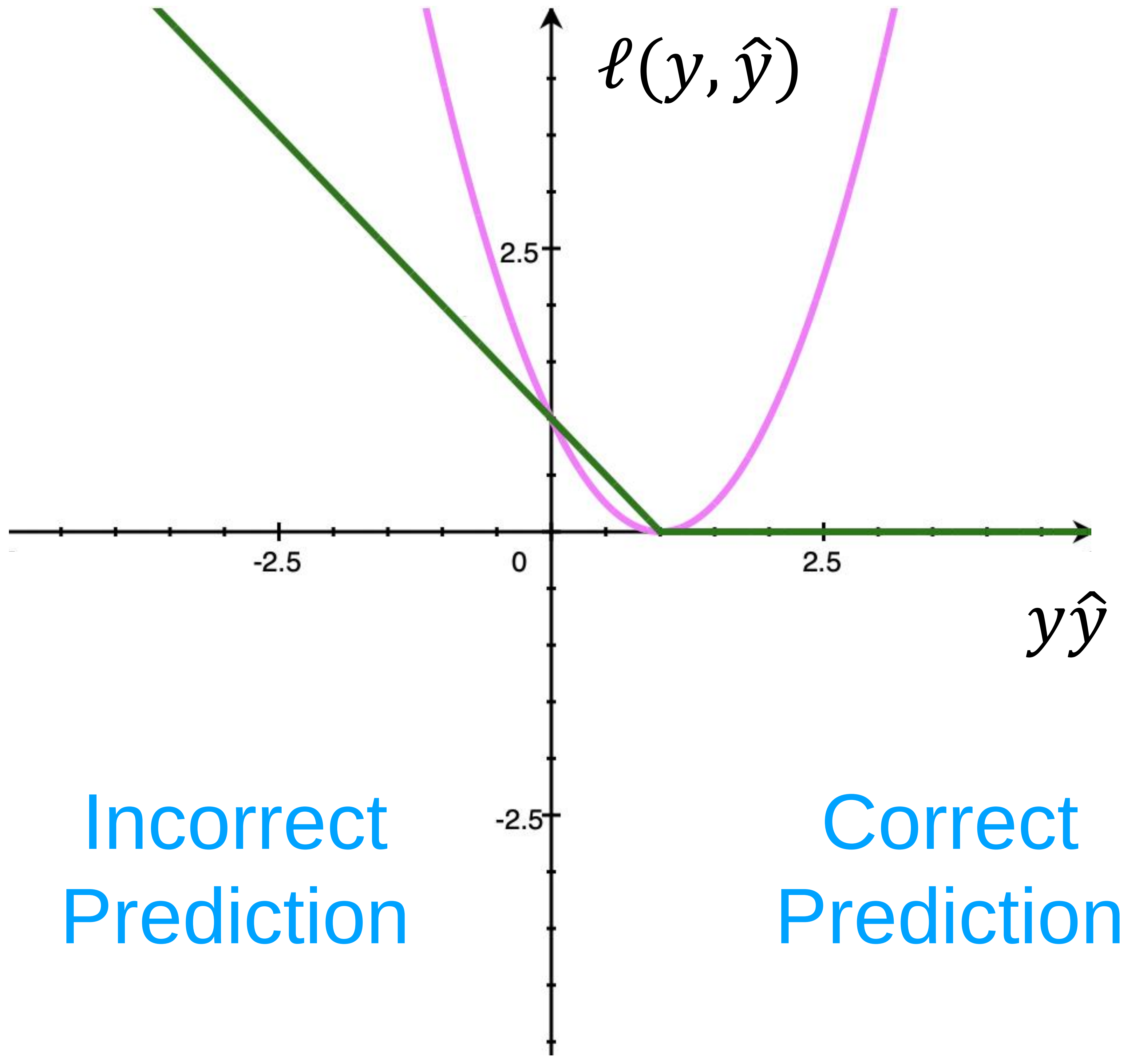
- When used for classification (i.e. when $y = \pm 1$), this is equivalent to

$$\ell(y, \hat{y}) = (1 - y\hat{y})^2 = \begin{cases} (1 - \hat{y})^2 & y = 1 \\ (1 + \hat{y})^2 & y = -1 \end{cases}$$

Since

$$\ell(y, \hat{y}) = (y - \hat{y})^2 = \begin{cases} (1 - \hat{y})^2 & y = 1 \\ (-1 - \hat{y})^2 = (1 + \hat{y})^2 & y = -1 \end{cases}$$

In soft-margin SVM: $\ell(y, \hat{y}) = \max(0, 1 - y\hat{y})$



Ridge Regression vs. Soft-Margin SVM

So we can compare ridge regression to soft-margin SVM in a unified framework.

Both optimize an objective function of the following form:

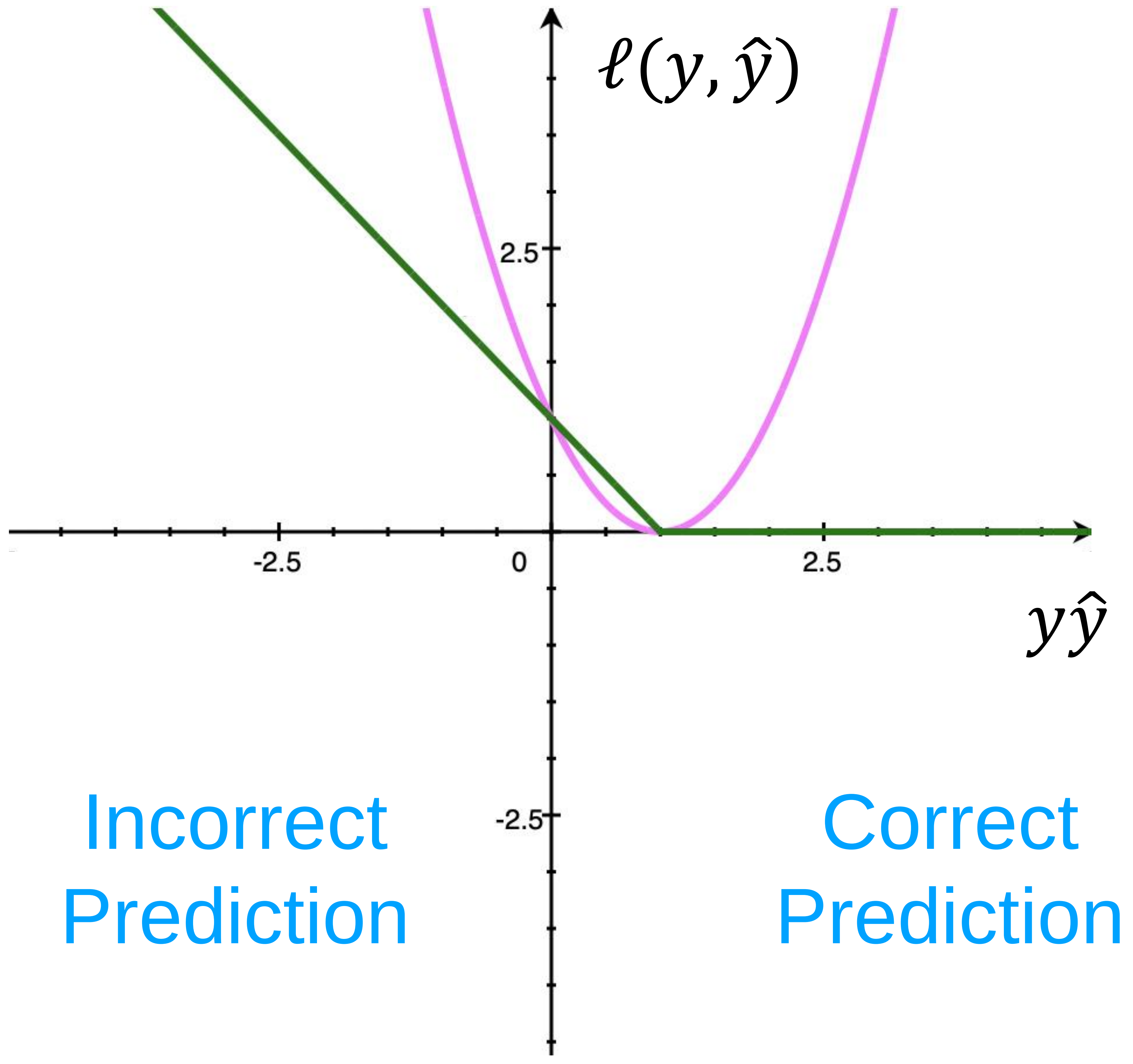
$$\min_{\vec{w}} \sum_{i=1}^{N} l\big(y_i, f(\vec{x}_i; \vec{w})\big) + \lambda \|\vec{w}\|_2^2, \text{where } f(\vec{x}; \vec{w}) = \vec{w}^\top \vec{x}$$

Ridge regression (square loss): $\ell(y, \hat{y}) = (1 - y\hat{y})^2$

Soft-margin SVM (hinge loss): $\ell(y, \hat{y}) = \max(0, 1 - y\hat{y})$

Compared to soft-margin SVM, ridge regression:

- Penalizes predictions that are *too* correct (on the correct side of the decision boundary, but is too far from it). It is therefore sensitive to high intra-class variability.
- Penalizes predictions that are very incorrect much more harshly. It is therefore sensitive to outliers.



Ridge Regression vs. Soft-Margin SVM

So we can compare ridge regression to soft-margin SVM in a unified framework.

Both optimize an objective function of the following form:

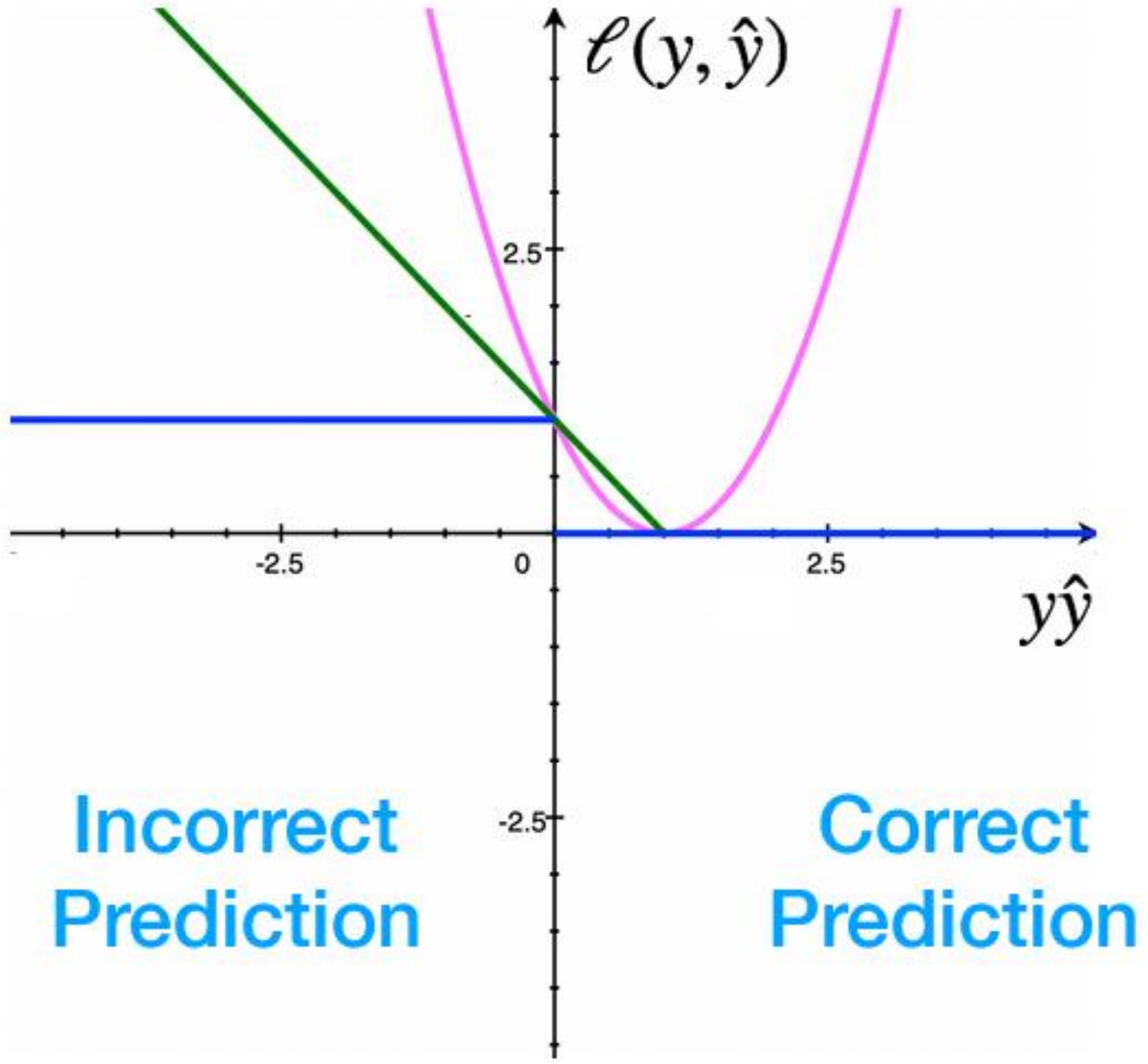
$$\min_{\vec{w}} \sum_{i=1}^{N} \ell\big(y_i, f(\vec{x}_i; \vec{w})\big) + \lambda \|\vec{w}\|_2^2 \text{, where } f(\vec{x}; \vec{w}) = \vec{w}^\top \vec{x}$$

Ridge regression (square loss): $\ell(y, \hat{y}) = (1 - y\hat{y})^2$

Soft-margin SVM (hinge loss): $\ell(y, \hat{y}) = \max(0, 1 - y\hat{y})$

0-1 loss: $\ell(y, \hat{y}) = \begin{cases} 1 & y \neq \operatorname{sign} \hat{y} \\ 0 & y = \operatorname{sign} \hat{y} \end{cases} = \begin{cases} 1 & y\hat{y} < 0 \\ 0 & y\hat{y} > 0 \end{cases}$

This is essentially what we care about, since in that case, $\sum_{i=1}^{N} \ell\big(y_i, f(\vec{x}_i; \vec{w})\big)$ is the number of training examples that the classifier got right. When divided by the total number of training examples, it is the classification accuracy.



Ridge Regression vs. Soft-Margin SVM

So we can compare ridge regression to soft-margin SVM in a unified framework.

Both optimize an objective function of the following form:

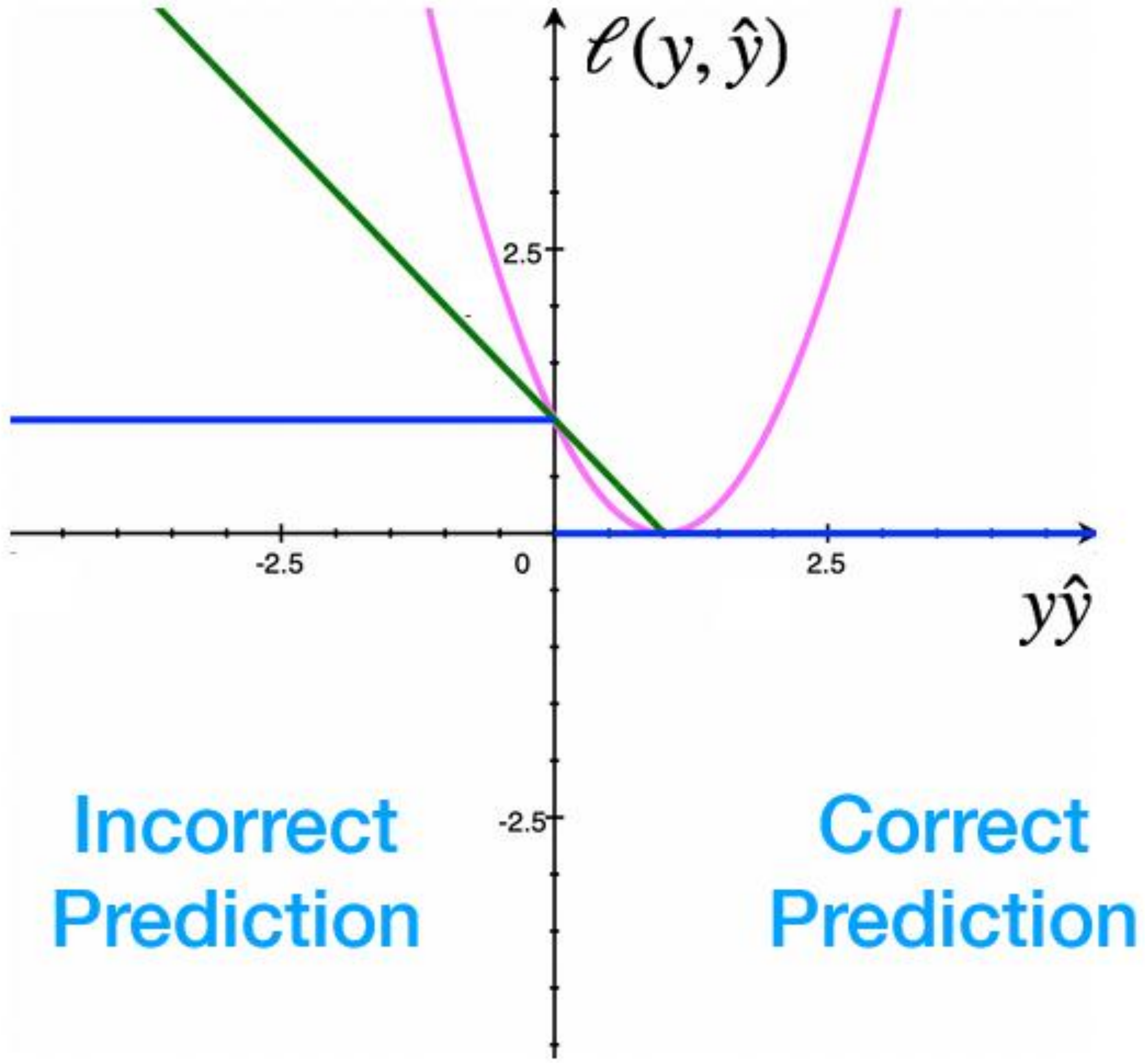
$$\min_{\vec{w}} \sum_{i=1}^{N} \ell\big(y_i, f(\vec{x}_i; \vec{w})\big) + \lambda \|\vec{w}\|_2^2, \text{where } f(\vec{x}; \vec{w}) = \vec{w}^\top \vec{x}$$

Ridge regression (square loss): $\ell(y, \hat{y}) = (1 - y\hat{y})^2$

Soft-margin SVM (hinge loss): $\ell(y, \hat{y}) = \max(0, 1 - y\hat{y})$

0-1 loss: $\ell(y, \hat{y}) = \begin{cases} 1 & y \neq \operatorname{sign} \hat{y} \\ 0 & y = \operatorname{sign} \hat{y} \end{cases} = \begin{cases} 1 & y\hat{y} < 0 \\ 0 & y\hat{y} > 0 \end{cases}$

Unfortunately, the 0-1 loss has zero derivative almost everywhere and is non-convex, so it is hard to optimize the objective with 0-1 loss. (In fact it is NP-hard.)



Ridge Regression vs. Soft-Margin SVM

So we can compare ridge regression to soft-margin SVM in a unified framework.

Both optimize an objective function of the following form:

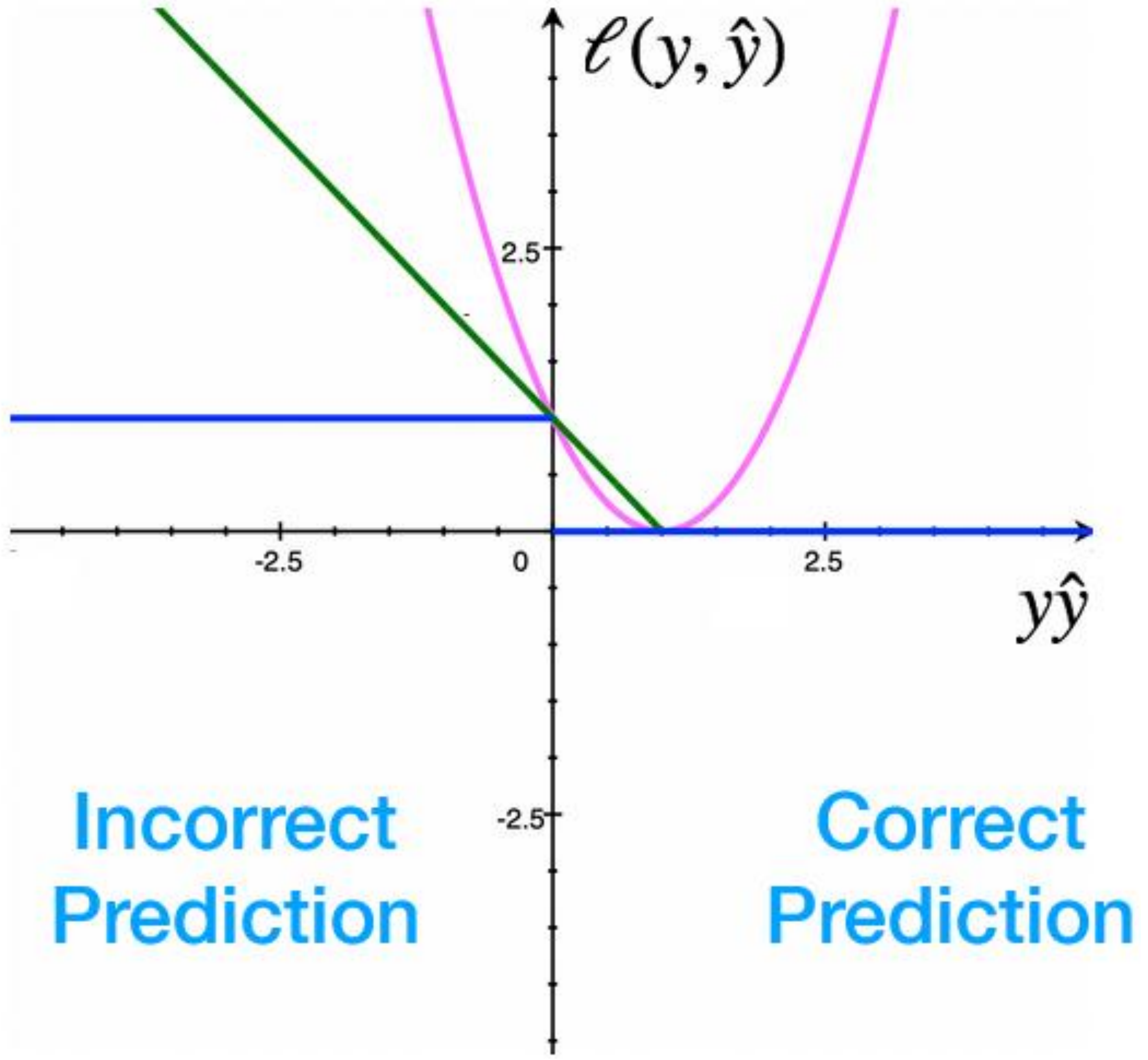
$$\min_{\vec{w}} \sum_{i=1}^{N} \ell\big(y_i, f(\vec{x}_i; \vec{w})\big) + \lambda \|\vec{w}\|_2^2 \text{, where } f(\vec{x}; \vec{w}) = \vec{w}^\top \vec{x}$$

Ridge regression (square loss): $\ell(y, \hat{y}) = (1 - y\hat{y})^2$

Soft-margin SVM (hinge loss): $\ell(y, \hat{y}) = \max(0, 1 - y\hat{y})$

0-1 loss: $\ell(y, \hat{y}) = \begin{cases} 1 & y \neq \operatorname{sign} \hat{y} \\ 0 & y = \operatorname{sign} \hat{y} \end{cases} = \begin{cases} 1 & y\hat{y} < 0 \\ 0 & y\hat{y} > 0 \end{cases}$

As a result, we have to use alternatives, such as the square loss or hinge loss, which upper bound the 0-1 loss. These are known as **surrogate losses** to the 0-1 loss.



Ridge Regression vs. Soft-Margin SVM

So we can compare ridge regression to soft-margin SVM in a unified framework.

Both optimize an objective function of the following form:

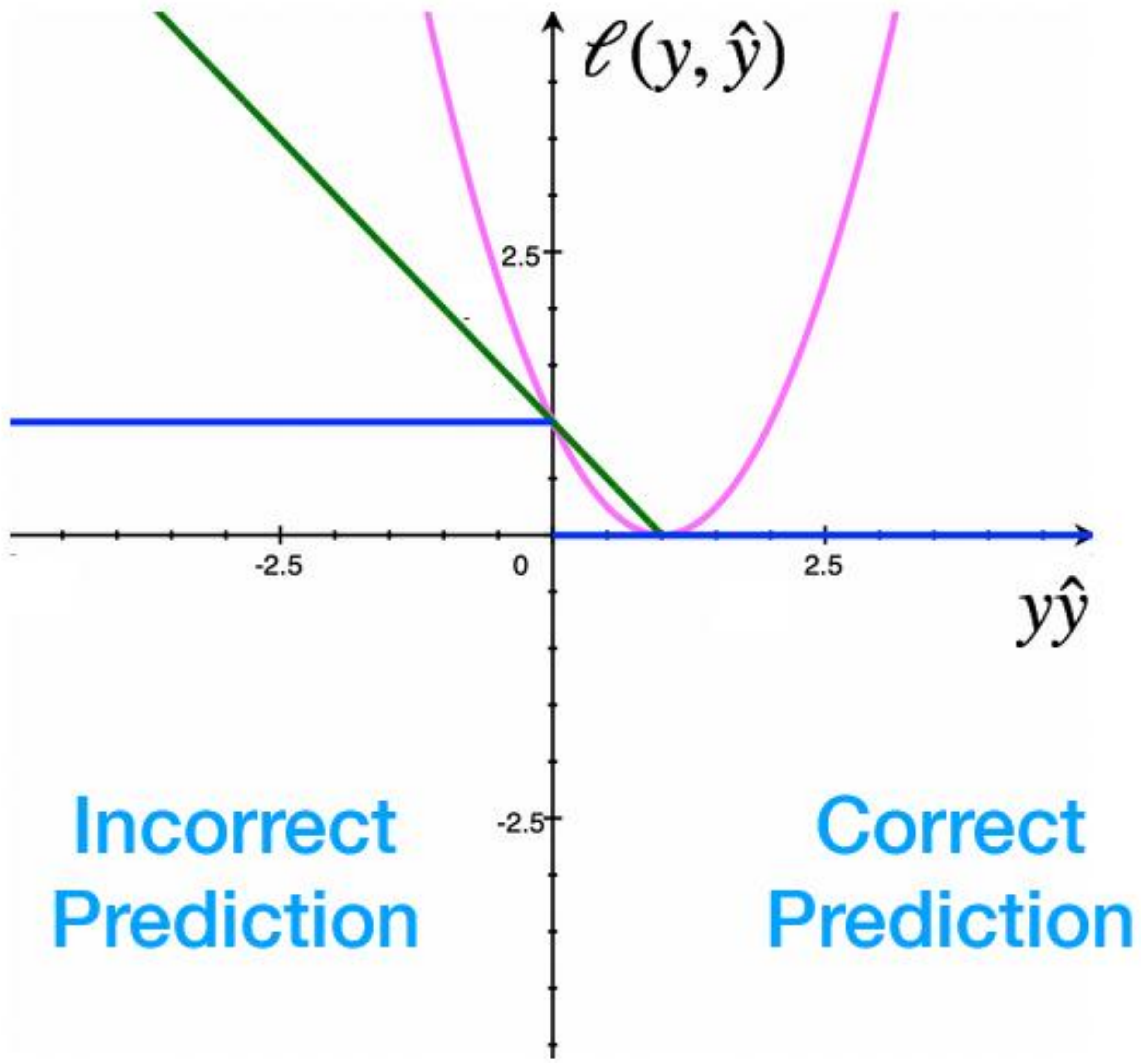
$$\min_{\vec{w}} \sum_{i=1}^{N} \ell\big(y_i, f(\vec{x}_i; \vec{w})\big) + \lambda \|\vec{w}\|_2^2 \text{, where } f(\vec{x}; \vec{w}) = \vec{w}^\top \vec{x}$$

Ridge regression (square loss): $\ell(y, \hat{y}) = (1 - y\hat{y})^2$

Soft-margin SVM (hinge loss): $\ell(y, \hat{y}) = \max(0, 1 - y\hat{y})$

0-1 loss: $\ell(y, \hat{y}) = \begin{cases} 1 & y \neq \operatorname{sign} \hat{y} \\ 0 & y = \operatorname{sign} \hat{y} \end{cases} = \begin{cases} 1 & y\hat{y} < 0 \\ 0 & y\hat{y} > 0 \end{cases}$

Logistic loss: $\ell(y, \hat{y}) = \log\big(1 + e^{-y\hat{y}}\big) / \log 2$



Ridge Regression vs. Soft-Margin SVM

So we can compare ridge regression to soft-margin SVM in a unified framework.

Both optimize an objective function of the following form:

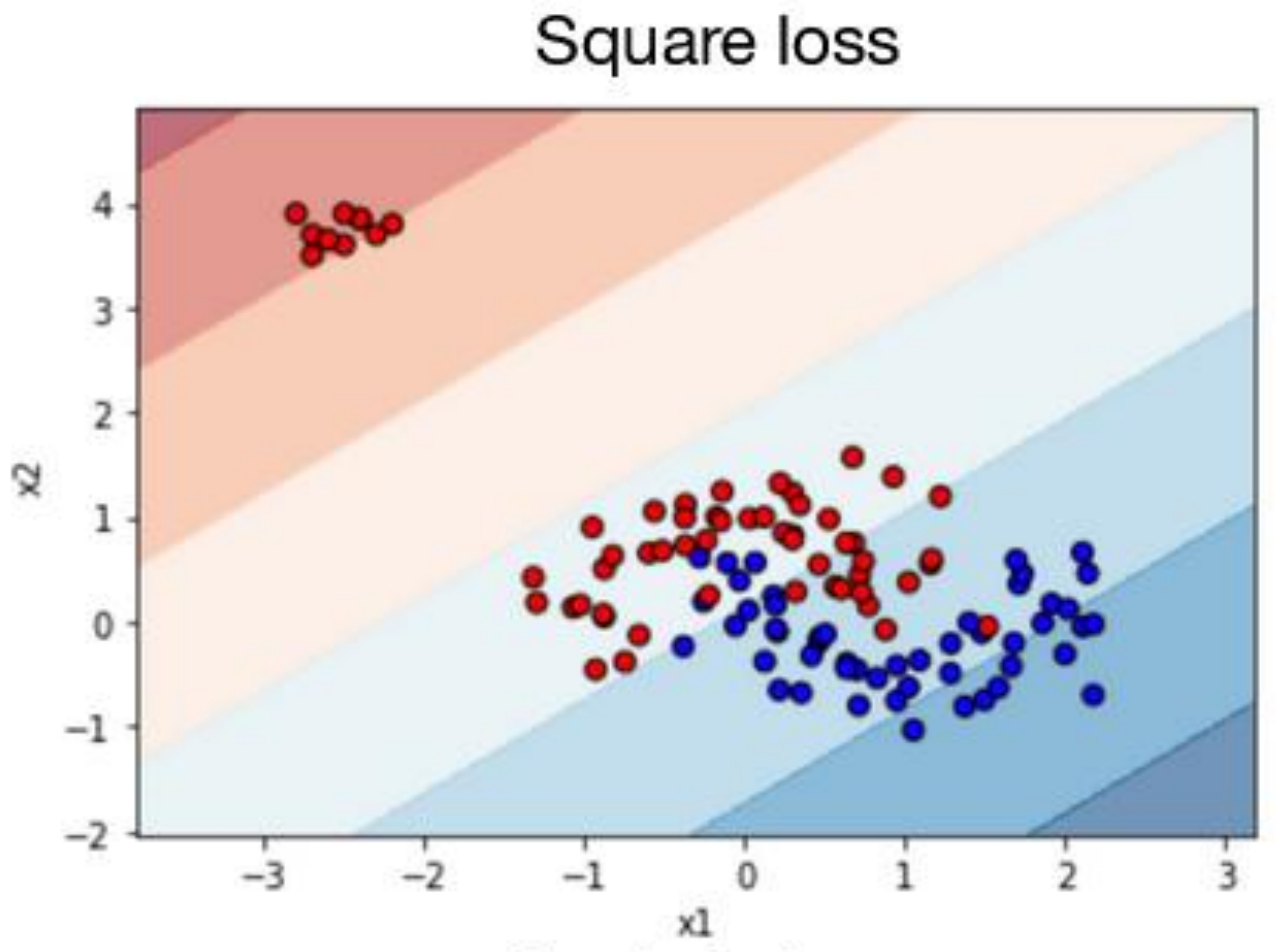
$$\min_{\vec{w}} \sum_{i=1}^{N} \ell\big(y_i, f(\vec{x}_i; \vec{w})\big) + \lambda \|\vec{w}\|_2^2 \text{, where } f(\vec{x}; \vec{w}) = \vec{w}^\top \vec{x}$$

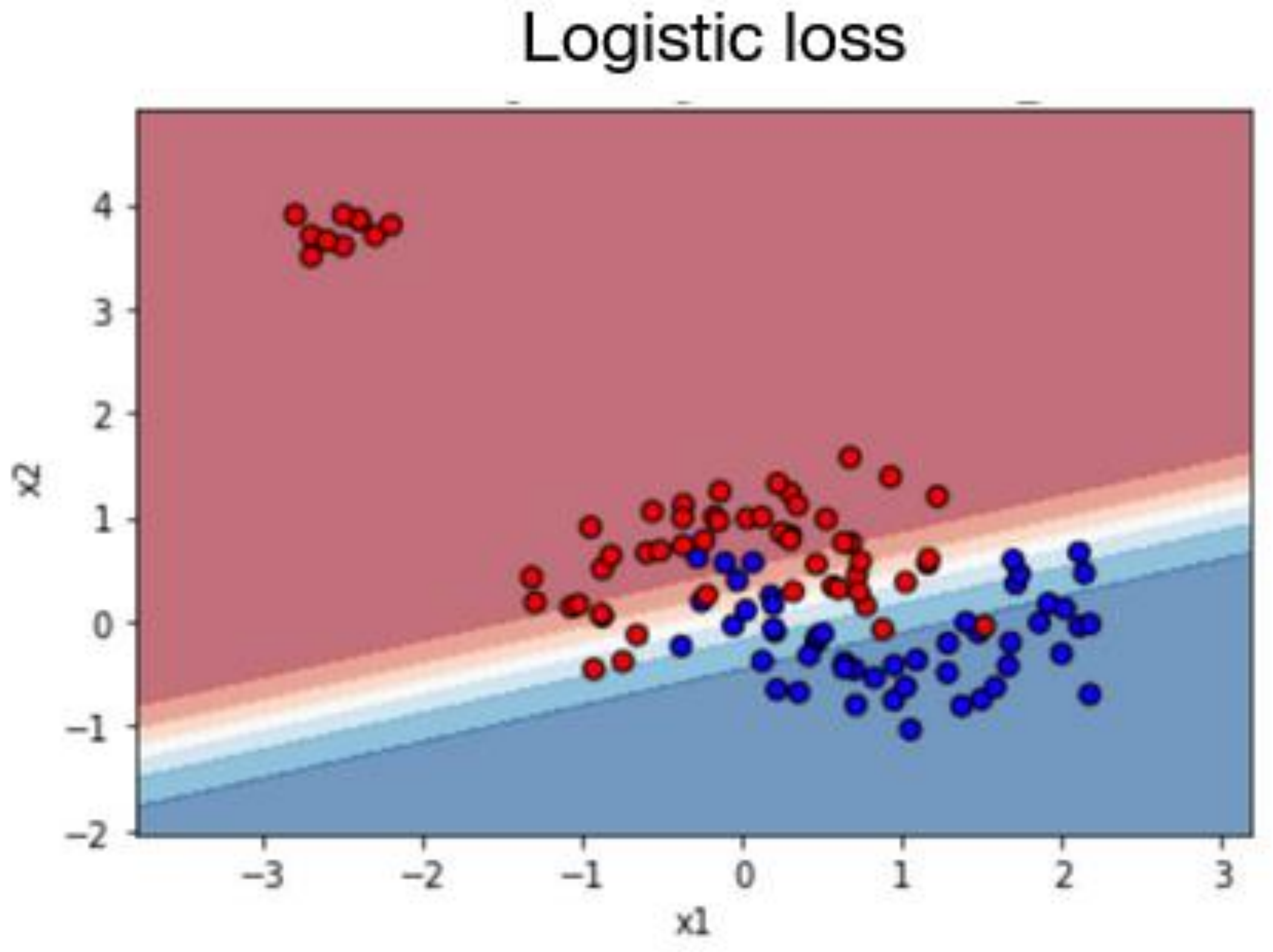
Ridge regression (square loss): $\ell(y, \hat{y}) = (1 - y\hat{y})^2$

Soft-margin SVM (hinge loss): $\ell(y, \hat{y}) = \max(0, 1 - y\hat{y})$

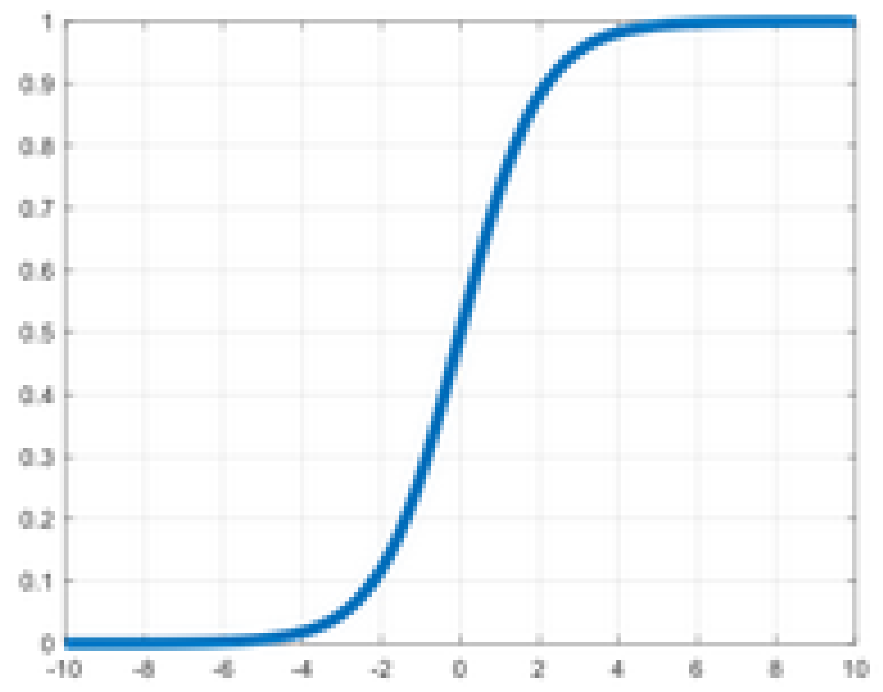
0-1 loss: $\ell(y, \hat{y}) = \begin{cases} 1 & y \neq \operatorname{sign} \hat{y} \\ 0 & y = \operatorname{sign} \hat{y} \end{cases} = \begin{cases} 1 & y\hat{y} < 0 \\ 0 & y\hat{y} > 0 \end{cases}$

Logistic loss: $\ell(y, \hat{y}) = \log\big(1 + e^{-y\hat{y}}\big) / \log 2$





Logistic Regression



Plugging in the logistic loss gives rise to a classifier known as logistic regression (will show this precisely later).

We consider the following model: $f(\vec{x}, \vec{w}) = \sigma(\vec{w}^\top \vec{x})$, where $\sigma(\cdot)$ is the sigmoid function, i.e.: $\sigma(z) = \frac{1}{1+e^{-z}}$ (this is also sometimes known as the **logistic function**). The parameters are $\vec{w}$. Here we use the convention that $\vec{x} = (x_1 \quad \cdots \quad x_n \quad 1)^\top$.

This looks almost like a linear model. If general, a model of the form$f(\vec{x}, \vec{w}) = \psi(\vec{w}^\top \vec{x})$is known as a **generalized linear model**, where $\psi^{-1}(\cdot)$ is known as the link function.

Logistic regression is special case of a generalized linear model, where the link function is $\sigma^{-1}(z) = \log\left(\frac{z}{1-z}\right)$.

This function is known as the **logit function** (not to be confused with the logistic function) or just the **logit**.

Because the model output is between 0 and 1, we can interpret the output of the model as the probability that the label of the input is positive.

Logistic Regression

A bit of a misnomer: logistic regression actually performs classification rather than regression.

Sometimes known as **logit regression**, **logit model** or just the **logit** in other fields, e.g.: econometrics.

The term logit is overloaded; it could mean the logit function, the logit model or the quantity $\log\left(\frac{z}{1-z}\right)$, where z is some probability.

In ML, the logit usually takes on the last meaning, where z is a particular choice, namely the probability of a particular label assigned by the classifier.

$$\log\left(\frac{\sigma(\vec{w}^\top\vec{x})}{1-\sigma(\vec{w}^\top\vec{x})}\right) = \sigma^{-1}(\sigma(\vec{w}^\top\vec{x})) = \vec{w}^\top\vec{x}$$

Logistic Regression

To make a prediction $\hat{y}$, we simply threshold the output of the model:

$$\hat{y} = \begin{cases} 1 & \sigma(\vec{w}^\top \vec{x}) > \frac{1}{2} \\ -1 & \sigma(\vec{w}^\top \vec{x}) < \frac{1}{2} \end{cases} = \begin{cases} 1 & \vec{w}^\top \vec{x} > 0 \\ -1 & \vec{w}^\top \vec{x} < 0 \end{cases}$$

Observe that the decision boundary is $\{\vec{x} \mid \vec{w}^\top \vec{x} = 0\}$.

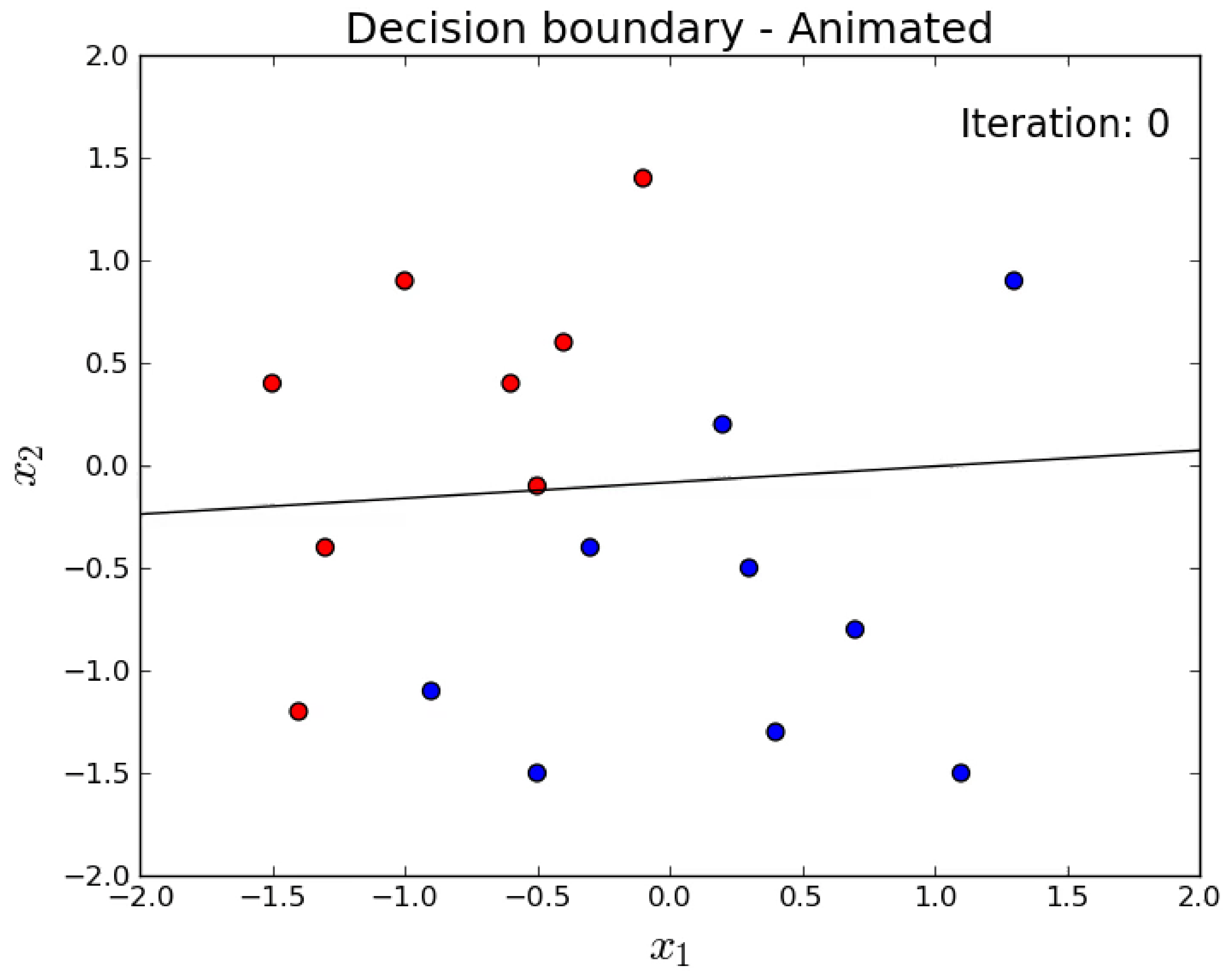
As we saw earlier with SVMs, this is a hyperplane. So, logistic regression is a linear classifier.

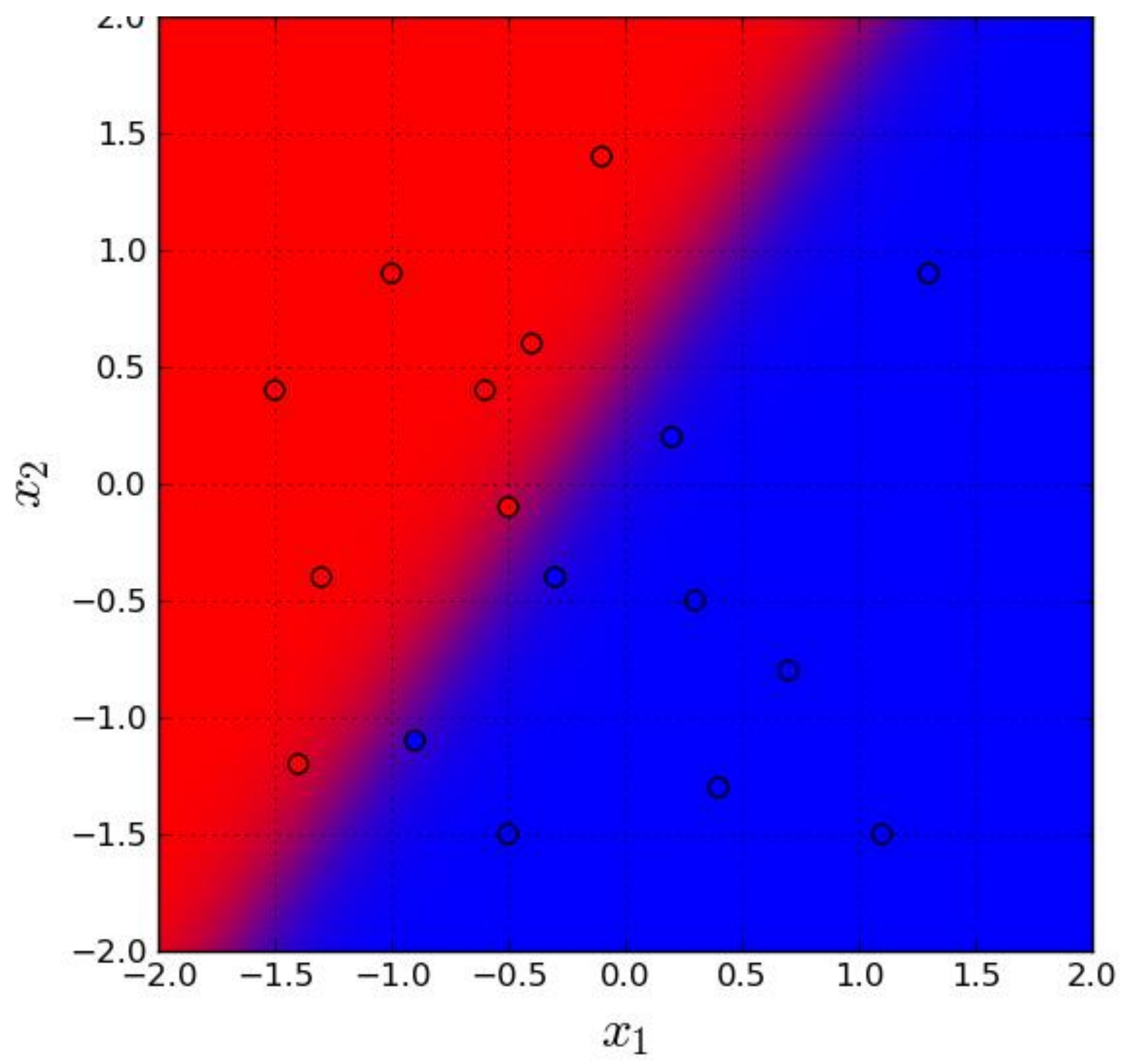
To train the model, we use the (binary**) cross entropy loss**: (shown here with ℓ_2 regularization)

$$L(\vec{w}) = -\sum_{i=1}^{N} ([y_i = 1] \log \Pr(\hat{y}_i = 1) + [y_i = -1] \log \Pr(\hat{y}_i = -1)) + \lambda \|\vec{w}\|_2^2 \,,$$

where $[x = i]$ is the Iverson bracket, i.e.: $[x = i] = \begin{cases} 1 & x = i \\ 0 & x \neq i \end{cases}$

Logistic Regression





Credit: Sait Celebi

Logistic Regression

$$L(\vec{w}) = -\sum_{i=1}^{N}([y_i = 1]\log \Pr(\hat{y}_i = 1) + [y_i = -1]\log \Pr(\hat{y}_i = -1)) + \lambda \|\vec{w}\|_2^2$$

Why is this called cross entropy?

Consider a discrete random variable $\hat{Y}$ that take on the values 1 and -1. The entropy $H(\hat{Y})$) with base e) is

$$H(\hat{Y}) = E_{\hat{Y}\sim p}\left[-\log p(\hat{Y})\right] = -\sum_{y\in\Omega_{\hat{Y}}} p(y)\log p(y) = p(1)\log p(1) + p(-1)\log p(-1)\,.$$

For cross entropy, we take the expectation w.r.t. a different distribution q:

$$E_{\hat{Y}\sim q}\left[-\log p(\hat{Y})\right] = -\sum_{y\in\Omega_{\hat{Y}}} q(y)\log p(y) = q(1)\log p(1) + q(-1)\log p(-1)$$

Each term in the main loss corresponds to setting $q(y) = \begin{cases} 1 & y = 1 \\ 0 & y = -1 \end{cases}$ or $\begin{cases} 0 & y = 1 \\ 1 & y = -1 \end{cases}$.

Put the loss and the model together…

$$L(\vec{w}) = -\sum_{i=1}^{N}([y_i = 1]\log\Pr(\hat{y}_i = 1) + [y_i = -1]\log\Pr(\hat{y}_i = -1)) + \lambda\|\vec{w}\|_2^2$$

$$= -\sum_{i=1}^{N}([y_i = 1]\log\Pr(\hat{y}_i = 1) + [y_i = -1]\log(1 - \Pr(\hat{y}_i = 1))) + \lambda\|\vec{w}\|_2^2$$

$$= -\sum_{i=1}^{N}\big([y_i = 1]\log\sigma(\vec{w}^\top\vec{x}_i) + [y_i = -1]\log\big(1 - \sigma(\vec{w}^\top\vec{x}_i)\big)\big) + \lambda\|\vec{w}\|_2^2$$

$$= -\sum_{i=1}^{N}\left([y_i = 1]\log\left(\frac{1}{1 + e^{-\vec{w}^\top\vec{x}_i}}\right) + [y_i = -1]\log\left(1 - \frac{1}{1 + e^{-\vec{w}^\top\vec{x}_i}}\right)\right) + \lambda\|\vec{w}\|_2^2$$

$$= -\sum_{i=1}^{N}\left([y_i = 1]\log\left(\frac{1}{1 + e^{-\vec{w}^\top\vec{x}_i}}\right) + [y_i = -1]\log\left(\frac{e^{-\vec{w}^\top\vec{x}_i}}{1 + e^{-\vec{w}^\top\vec{x}_i}}\right)\right) + \lambda\|\vec{w}\|_2^2$$

$$= -\sum_{i=1}^{N}\left([y_i = 1]\log\left(\frac{1}{1 + e^{-\vec{w}^\top\vec{x}_i}}\right) + [y_i = -1]\log\left(\frac{e^{\vec{w}^\top\vec{x}_i}}{e^{\vec{w}^\top\vec{x}_i}}\cdot\frac{e^{-\vec{w}^\top\vec{x}_i}}{1 + e^{-\vec{w}^\top\vec{x}_i}}\right)\right) + \lambda\|\vec{w}\|_2^2$$

$$= -\sum_{i=1}^{N}\left([y_i = 1]\log\left(\frac{1}{1 + e^{-\vec{w}^\top\vec{x}_i}}\right) + [y_i = -1]\log\left(\frac{1}{1 + e^{\vec{w}^\top\vec{x}_i}}\right)\right) + \lambda\|\vec{w}\|_2^2$$

Logistic Regression

$$L(\vec{w}) = -\sum_{i=1}^{N}\left([y_i = 1]\log\left(\frac{1}{1+e^{-\vec{w}^\top \vec{x}_i}}\right) + [y_i = -1]\log\left(\frac{1}{1+e^{\vec{w}^\top \vec{x}_i}}\right)\right) + \lambda\|\vec{w}\|_2^2$$

$$= -\sum_{i=1}^{N}\left([y_i = 1]\left(-\log\left(1+e^{-\vec{w}^\top \vec{x}_i}\right)\right) + [y_i = -1]\left(-\log\left(1+e^{\vec{w}^\top \vec{x}_i}\right)\right)\right) + \lambda\|\vec{w}\|_2^2$$

$$= \sum_{i=1}^{N}\underbrace{\left([y_i = 1]\left(\log\left(1+e^{-\vec{w}^\top \vec{x}_i}\right)\right) + [y_i = -1]\left(\log\left(1+e^{\vec{w}^\top \vec{x}_i}\right)\right)\right)}_{\begin{cases}\log\left(1+e^{-\vec{w}^\top \vec{x}_i}\right) & y_i = 1\\ \log\left(1+e^{\vec{w}^\top \vec{x}_i}\right) & y_i = -1\end{cases}} + \lambda\|\vec{w}\|_2^2$$

Compare to the objective function on the right: only differs by a constant factor of $1/\log 2$. So the two objective functions are equivalent.

This shows that plugging in the logistic loss gives rise to logistic regression.

Recall the generic form of binary classification objective:

$$\min_{\vec{w}} \sum_{i=1}^{N} \ell\left(y_i, f(\vec{x}_i; \vec{w})\right) + \lambda\|\vec{w}\|_2^2, \text{where } f(\vec{x}; \vec{w}) = \vec{w}^\top \vec{x}$$

Recall the logistic loss:

$$\ell(y, \hat{y}) = \frac{\log\left(1+e^{-y\hat{y}}\right)}{\log 2} = \frac{1}{\log 2}\begin{cases}\log\left(1+e^{-\hat{y}}\right) & y_i = 1\\ \log\left(1+e^{\hat{y}}\right) & y_i = -1\end{cases}$$

So,

$$\ell\left(y_i, f(\vec{x}_i; \vec{w})\right) = \ell(y_i, \vec{w}^\top \vec{x})$$

$$= \frac{1}{\log 2}\begin{cases}\log\left(1+e^{-\vec{w}^\top \vec{x}_i}\right) & y_i = 1\\ \log\left(1+e^{\vec{w}^\top \vec{x}_i}\right) & y_i = -1\end{cases}$$

Logistic Regression

There is no closed form solution for the optimal parameters $\vec{w}$, but it turns out the objective function is both convex and Lipschitz continuous.

So we can use iterative gradient-based optimization methods like gradient descent to find the optimal parameters.

Non-Linear Classification

Recall the decision boundary of logistic regression is a hyperplane.

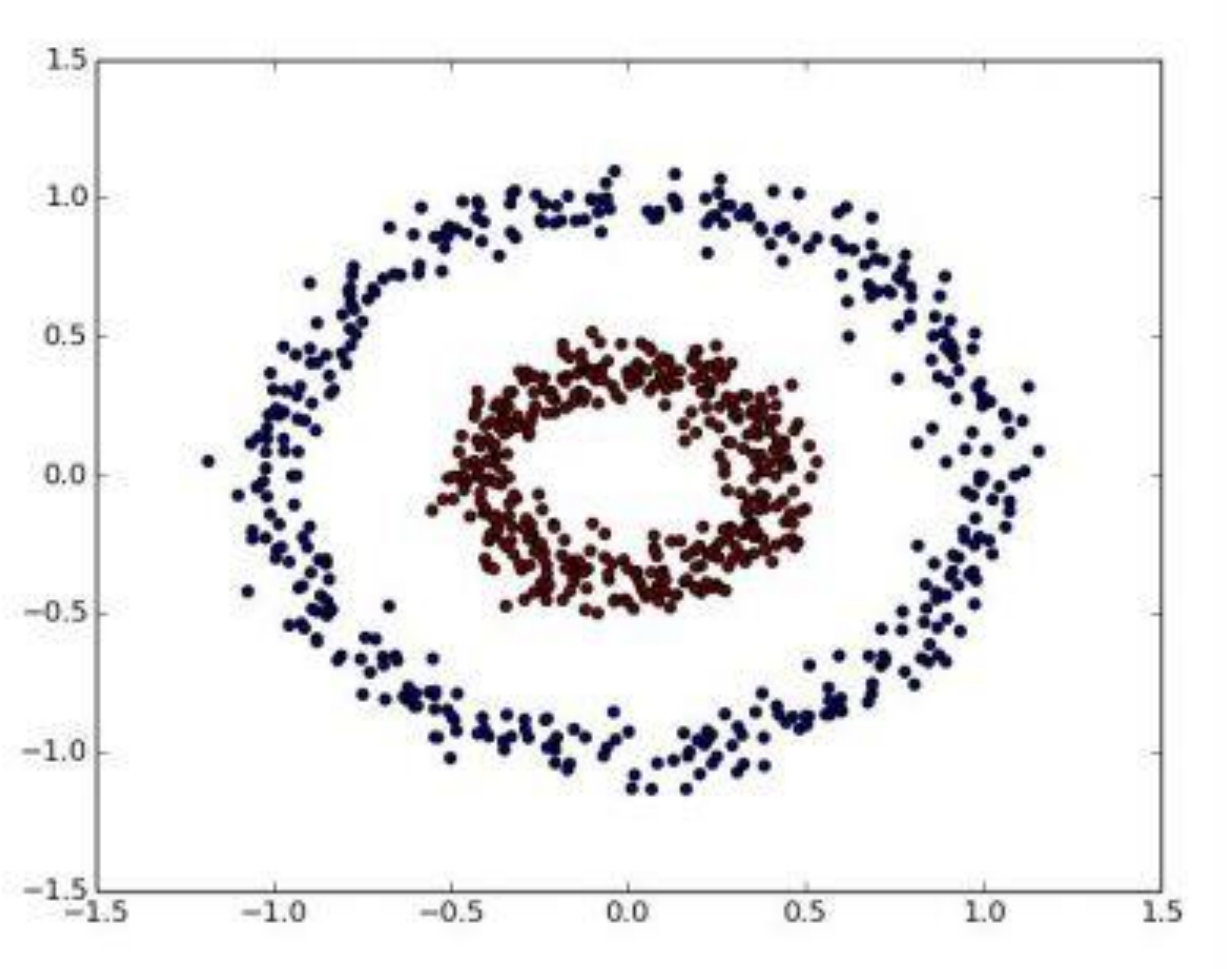
What if we'd like the decision boundary to be non-linear?

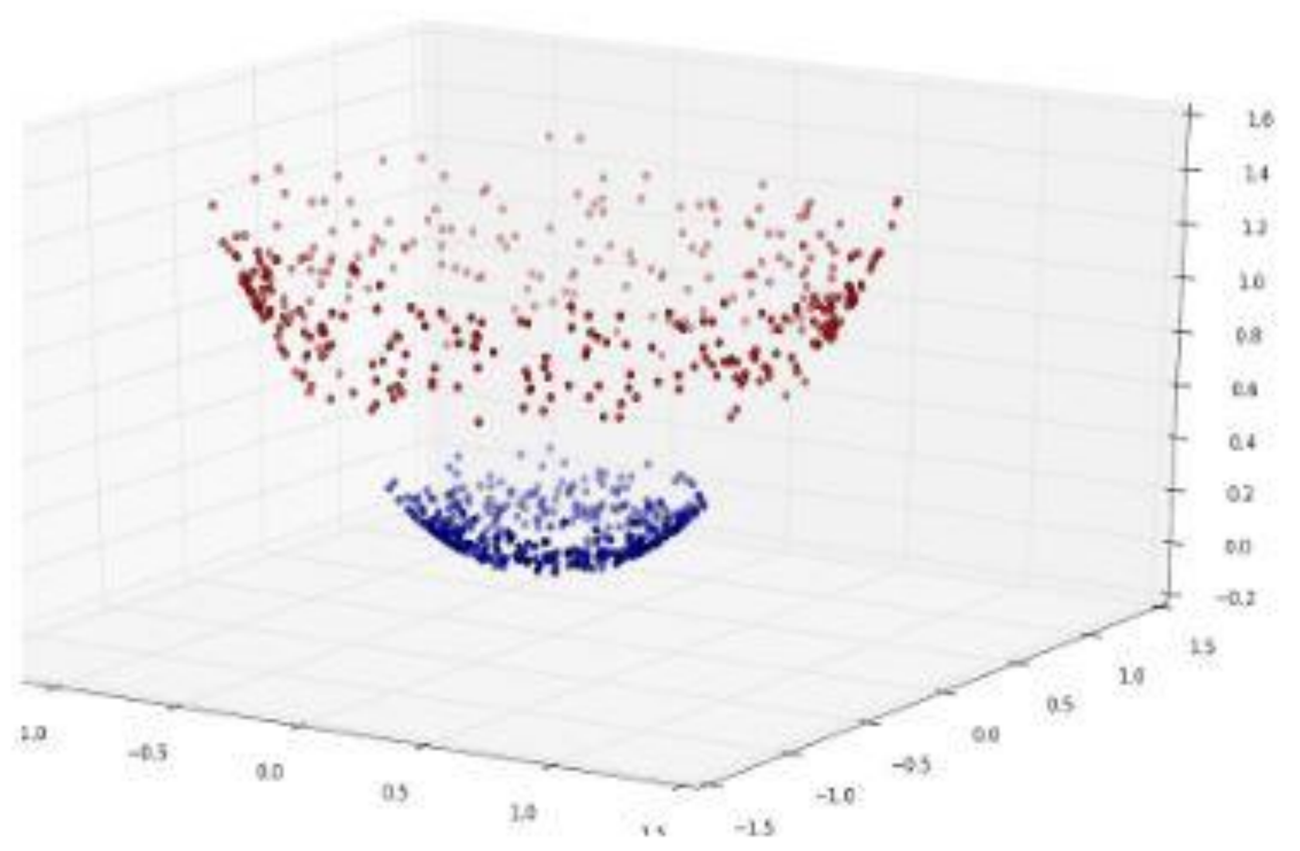
We can replace the raw input with features.

However, the kernel trick no longer applies because we don't have a constrained optimization problem and so cannot derive the dual. As a result, the features cannot be very high dimensional.

However, we can learn the features – we can replace the model with a neural network whose output layer has a sigmoid activation function. (Though the objective is no longer convex after doing so.)

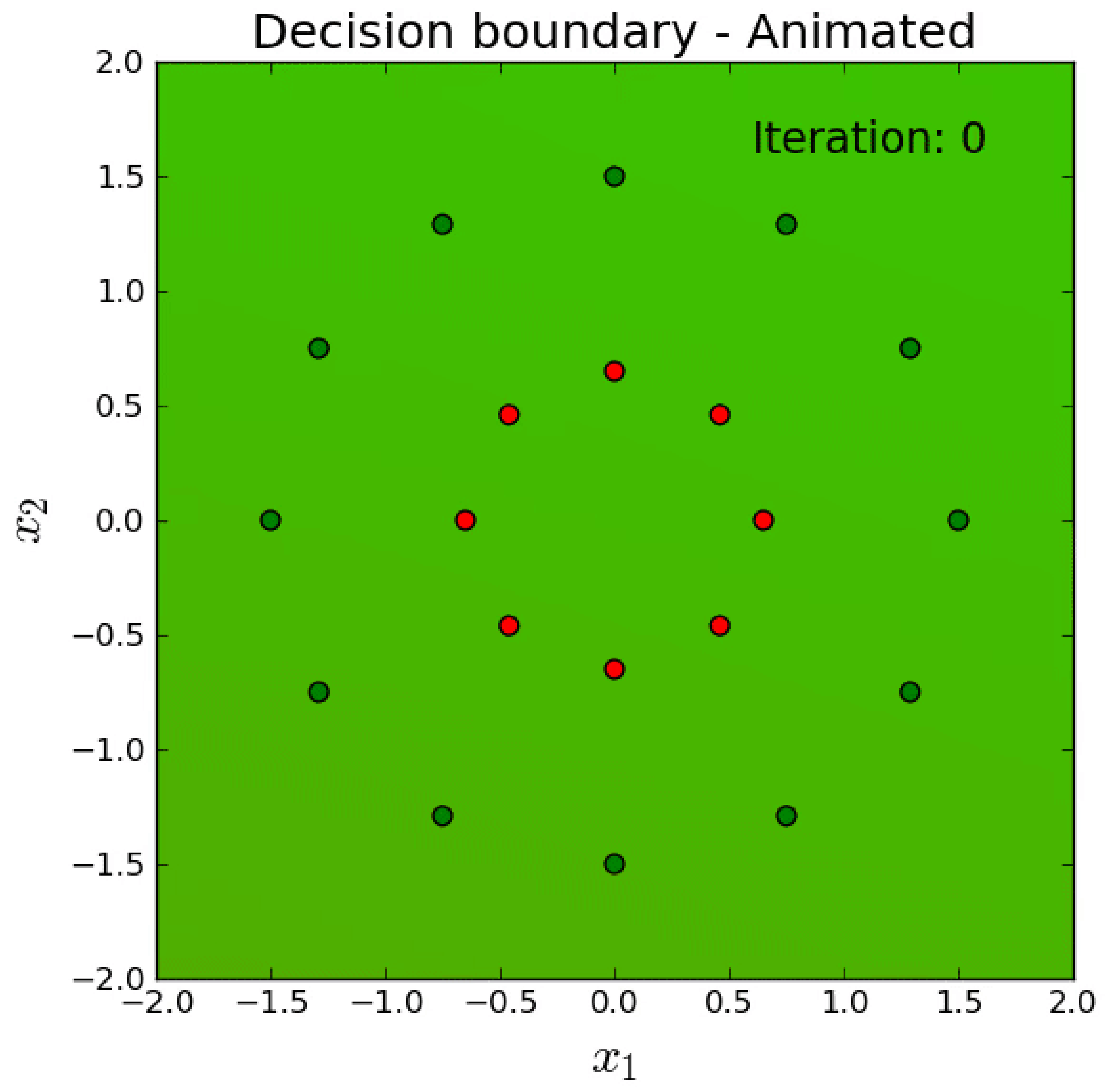
- Because the optimization problem is unconstrained, we can optimize it with iterative gradient-based methods. This is compatible with neural networks, because they are also trained with gradient-based methods.

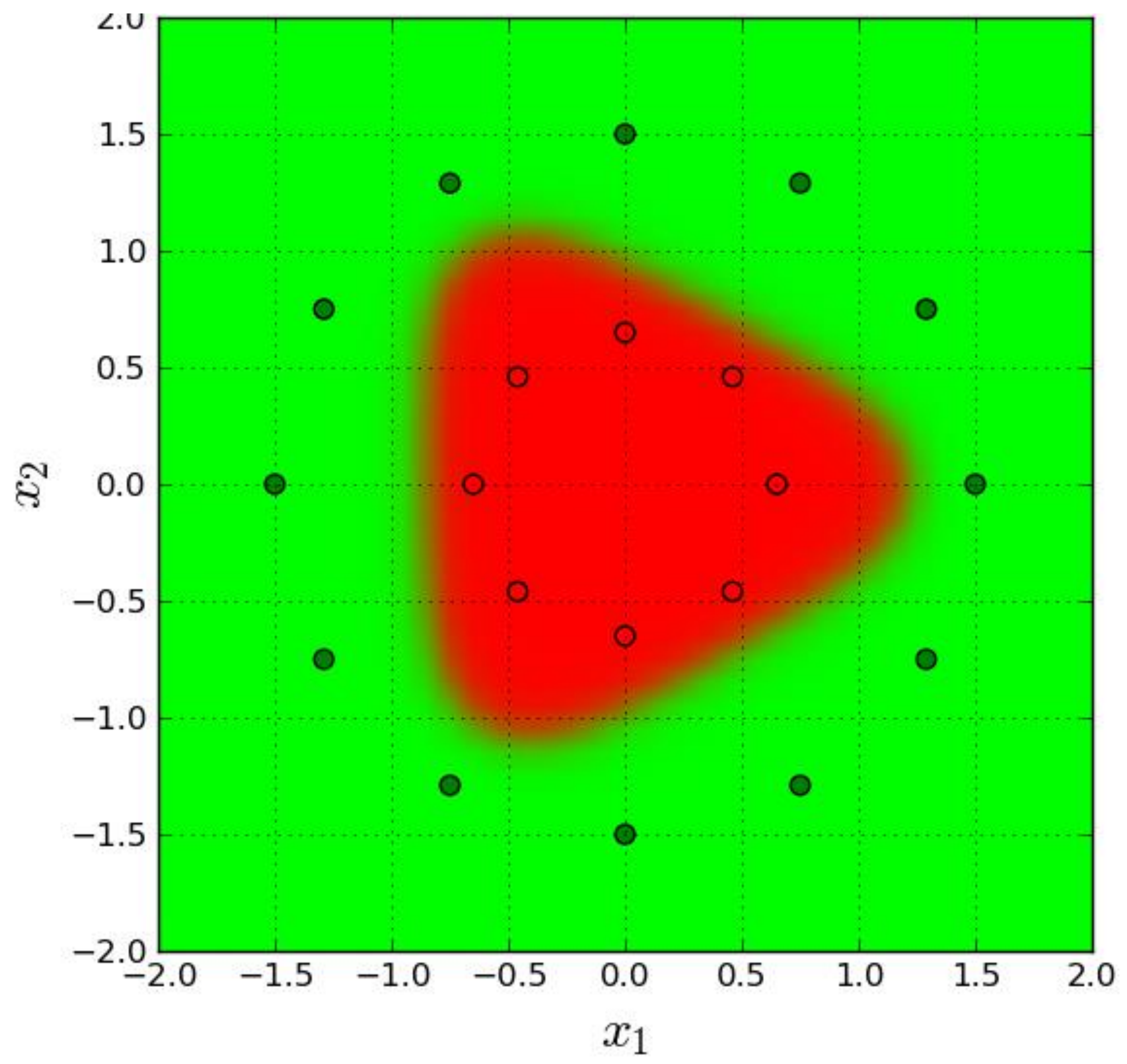




Credit: Suriya Narayanan

Logistic Regression with Neural Networks





Credit: Sait Celebi

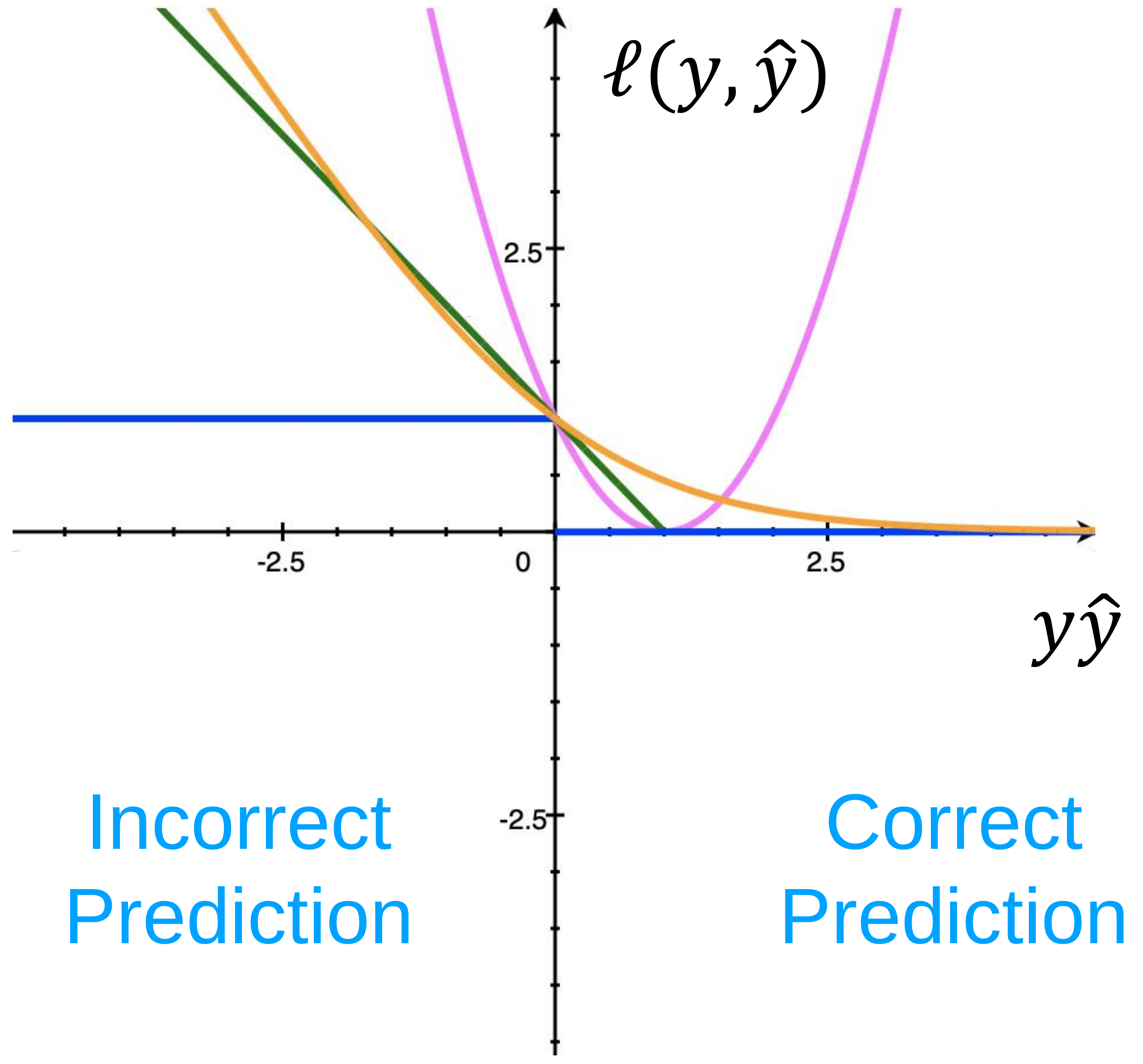
Logistic Regression vs. Soft-Margin SVM

Recall: The key difference between logistic regression and soft-margin SVM is the choice of the loss. Logistic regression uses the logistic loss, whereas soft-margin SVM uses the hinge loss.

They have similar behaviour – when should we choose one over the other?

- Hinge loss: Can be formulated as a constrained problem. So can derive the dual which only involves inner products between data points. As a result, can apply the kernel trick. When formulated as unconstrained problem, gradient-based methods do not work as well because the derivative is discontinuous.

- Logistic loss: Can only be formulated as an unconstrained problem, so must use iterative gradient-based methods to optimize it. Cannot apply the kernel trick, but works well with models that are trained with gradient-based methods anyway, i.e.: neural networks.



Softmax Classification

What if we have more than two classes?

Softmax classification is a generalization of logistic regression to multiple classes. It is also known as **softmax regression**, **multinomial logistic regression** and **multinomial logit**.

Suppose we have K classes. The model for softmax classification is

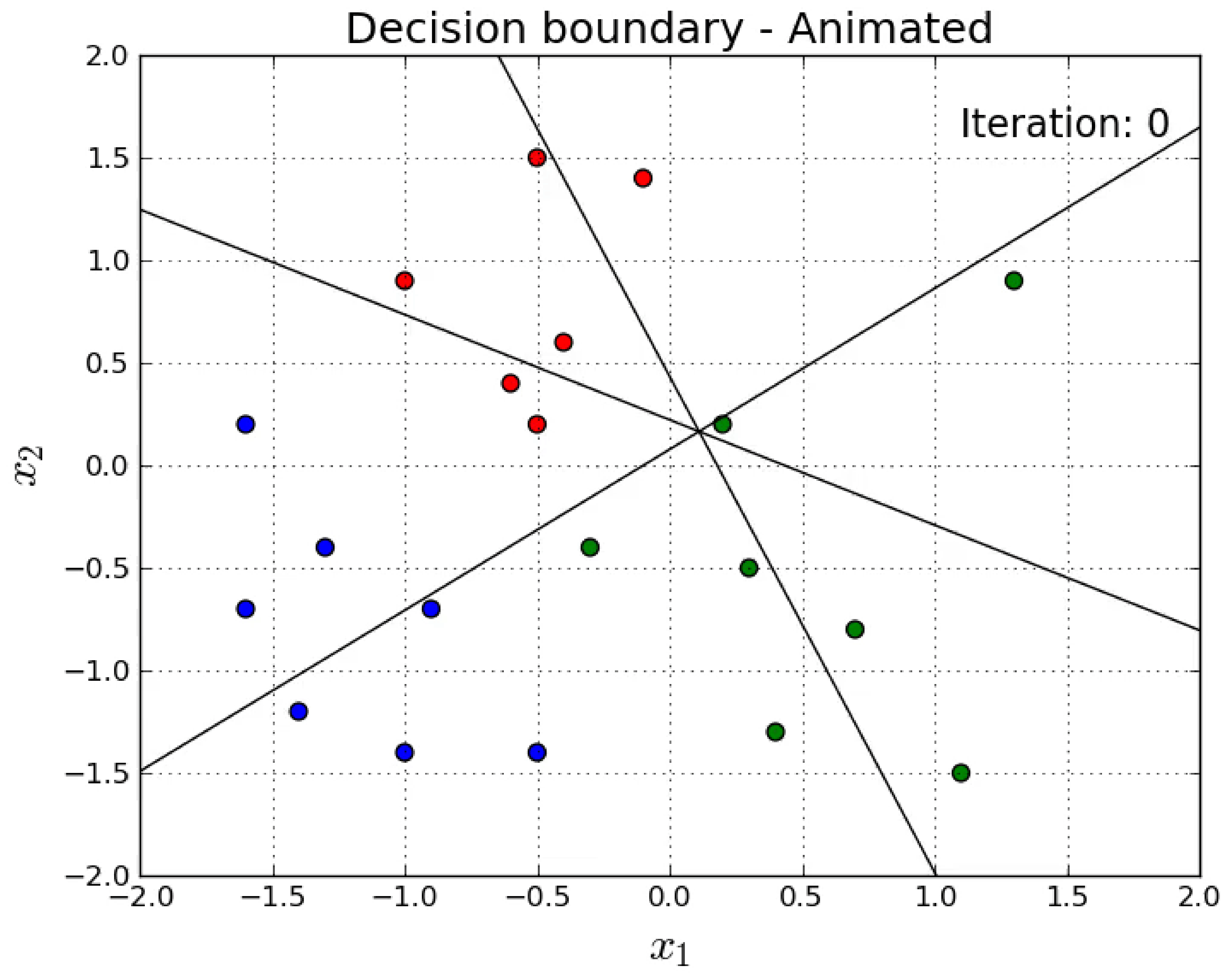
$$f\left(\vec{x}; \{\vec{w}_j\}_{j=1}^{K}\right) = \left(\underbrace{f_1\left(\vec{x}; \{\vec{w}_j\}_{j=1}^{K}\right)}_{\text{Class 1}} \quad \cdots \quad \underbrace{f_K\left(\vec{x}; \{\vec{w}_j\}_{j=1}^{K}\right)}_{\text{Class } K}\right)^{\top},$$

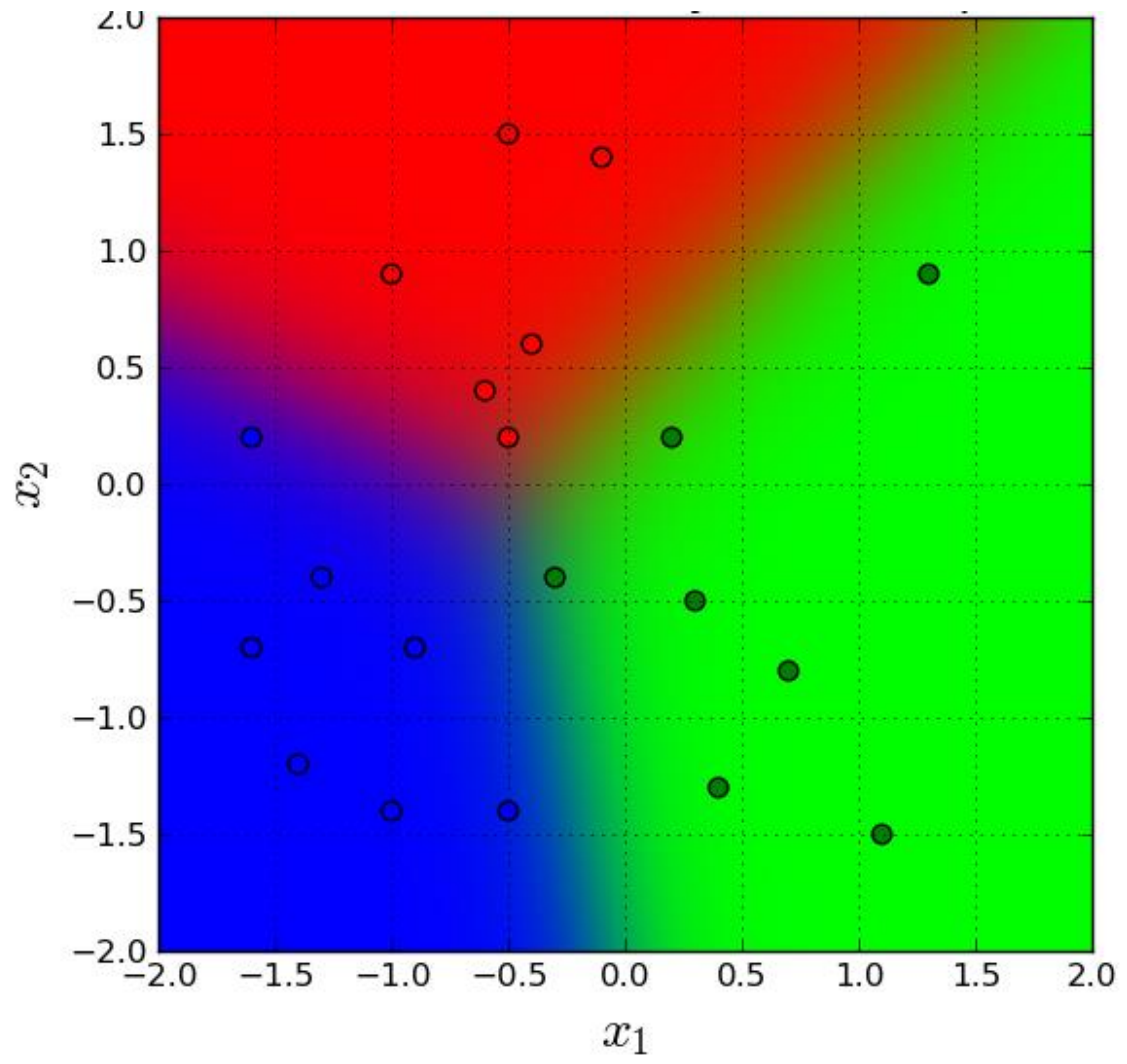
$$\text{where } f_k\left(\vec{x}; \{\vec{w}_j\}_{j=1}^{K}\right) = \frac{\exp\left(\vec{w}_k^{\top} \vec{x}\right)}{\sum_{j=1}^{K} \exp\left(\vec{w}_j^{\top} \vec{x}\right)}$$

The parameters are $\{\vec{w}_j\}_{j=1}^{K}$.

The function $\vec{\sigma}\left(\{z_j\}_{j=1}^{K}\right) = \left(\vec{\sigma}_1\left(\{z_j\}_{j=1}^{K}\right) \quad \cdots \quad \vec{\sigma}_K\left(\{z_j\}_{j=1}^{K}\right)\right)^{\top}$, where $\sigma_k\left(\{z_j\}_{j=1}^{K}\right) = \frac{\exp(z_k)}{\sum_{j=1}^{K} \exp(z_j)}$ is called the **softmax function**.

Softmax Classification





Credit: Sait Celebi

Softmax Classification

To train the model, we use the (multinomial) **cross entropy loss**: (shown here with ℓ_2 regularization)

$$L\left(\{\vec{w}_j\}_{j=1}^{K}\right) = -\sum_{i=1}^{N}\sum_{k=1}^{K}[y_i = k]\log\Pr(\hat{y}_i = k) + \lambda\sum_{j=1}^{K}\left\|\vec{w}_j\right\|_2^2, \text{where } [x = i] \text{ is the Iverson bracket.}$$

Let's put the loss and the model together:

$$L\left(\{\vec{w}_j\}_{j=1}^{K}\right) = -\sum_{i=1}^{N}\sum_{k=1}^{K}[y_i = k]\log\left(\frac{\exp\left(\vec{w}_k^{\top}\vec{x}\right)}{\sum_{j=1}^{K}\exp\left(\vec{w}_j^{\top}\vec{x}\right)}\right) + \lambda\sum_{j=1}^{K}\left\|\vec{w}_j\right\|_2^2$$

$$= -\sum_{i=1}^{N}\sum_{k=1}^{K}[y_i = k]\left(\vec{w}_k^{\top}\vec{x} - \underbrace{\log\left(\sum_{j=1}^{K}\exp\left(\vec{w}_j^{\top}\vec{x}\right)\right)}_{\text{logsumexp}\left(\{\vec{w}_j^{\top}\vec{x}\}_{j=1}^{K}\right)}\right) + \lambda\sum_{j=1}^{K}\left\|\vec{w}_j\right\|_2^2$$

This is more numerically stable to compute than the line above.
Many libraries have built-in routines for computing logsumexp.

Softmax Classification

When $K = 2$, softmax classification is equivalent to logistic regression. So logistic regression is a special case of softmax regression.

There is no closed form solution for the optimal parameters $\{\vec{w}_j\}_{j=1}^{K}$,but it turns out the objective function is both convex and Lipschitz continuous.

So we can use iterative gradient-based optimization methods like gradient descent to find the optimal parameters.

As with logistic regression, softmax classification is compatible with neural networks – we can replace the model with a neural network whose output layer has a softmax activation function.

Class Imbalance

Sometimes there are many more examples in one class than in others. This problem is known as **class imbalance**.

In such cases, classification accuracy and losses can be misleading. Example, if 99% of examples are positive and 1% of examples are negative, a classifier that always predicts the positive label will achieve 99% accuracy.

Using a vanilla objective function will lead to the classifier ignoring the negative examples because they contribute very little to the loss.

If you care about all classes equally, you should re-weight the classes in proportion to their inverse frequencies. E.g.:

$$L(\{\vec{w}_j\}_{j=1}^{K}) = -\sum_{i=1}^{N}\sum_{k=1}^{K}[y_i = k]\frac{N}{K\sum_{i'=1}^{N}[y_i' = k]}\log\Pr(\hat{y}_i = k) + \lambda\sum_{j=1}^{K}\left\|\vec{w}_j\right\|_2^2$$