Machine Learning
A Geometric Approach

Linear Classification: Support Vector Machines (SVM)
Professor Liang Huang
some slides from Alex Smola (CMU)
From Perceptron to SVM

From 1959 to 1964, the concept of the perceptron was invented by Rosenblatt. In 1962, Novikoff proved the perceptron's classification ability. The perceptron was later revived in 1999 by Freund and Schapire, who proposed the voted/average: revived approach.

In 1997, Cortes and Vapnik introduced SVM, which added kernels and soft-margin to the perceptron. This was a significant advancement as it allowed for the handling of inseparable cases.

In 1999, Collins introduced structured SVM, which was further refined in 2002 by McDonald, Crammer, and Pereira, who added conservative updates.

In 2003, Crammer and Singer introduced MIRA, which was a significant improvement in handling online approximations and max margin.

In 2005, Singer and colleagues introduced Pegasos, which used subgradient descent and minibatch methods to improve performance.

In 2006, Singer and colleagues continued to develop aggressive methods.

In 2007-2010, Singer and colleagues continued to refine their approaches, with Pegasos being a key component.


*mentioned in lectures but optional (others papers all covered in detail)
Large Margin Classifier

\[ \langle w, x \rangle + b \leq -1 \]

\[ \langle w, x \rangle + b \geq 1 \]

linear function

\[ f(x) = \langle w, x \rangle + b \]
Large Margin Classifier

\[ \langle w, x \rangle + b \leq -1 \]

\[ \langle w, x \rangle + b \geq 1 \]

linear function

\[ f(x) = \langle w, x \rangle + b \]
Why large margins?

- Maximum robustness relative to uncertainty
- Symmetry breaking
- Independent of correctly classified instances
- Easy to find for easy problems

\[
\Delta x \in H \text{ is bounded in norm by some } r > 0. \text{ Clearly, if we manage to separate the training set with a margin } \rho > r, \text{ we will correctly classify all test points: Since all training points have a distance of at least } \rho \text{ to the hyperplane, the test patterns will still be on the correct side (Figure 7.3, cf. also [146]).}
\]

![Figure 7.3](image)

Two-dimensional toy example of a classification problem: Separate 'o' from '+' using a hyperplane. Suppose that we add bounded noise to each pattern. If the optimal margin hyperplane has margin \( \rho \), and then \( \rho \) is bounded by \( r < \rho \), then the hyperplane will correctly separate even the noisy patterns. Conversely, if we run the perceptron algorithm (which finds some separating hyperplane, but not necessarily the optimal one) on the noisy data, then we would recover the optimal hyperplane in the limit \( r \to \rho \).

If we knew \( \rho \) beforehand, then this could actually be turned into an optimal margin classifier training algorithm, as follows. If we use a \( r \) which is slightly smaller than \( \rho \), the event the patterns will be separable with a nonzero margin. In this case, the standard perceptron algorithm can be shown to converge.

Rosenblatt's perceptron algorithm [423] is one of the simplest conceivable iterative procedures for computing a separating hyperplane. In its simplest form, it proceeds as follows. We start with an arbitrary weight vector \( w_0 \). At step \( n \in \mathbb{N} \), we consider the training example \((x_n, y_n)\). If it is classified correctly using the current weight vector (i.e., if \( \text{sgn} \langle x_n, w_n \rangle = y_n \)), we set \( w_n : = w_n - y_n x_n \); otherwise, we set \( w_n : = w_n - y_n x_n + \eta y_n x_i (\text{here, } \eta > 0 \text{ is a learning rate}) \). We then loop over patterns repeatedly, until we can complete one full pass through the training set without a single error. The result in weight vector will thus classify all points correctly. Novikoff [369] proved that this procedure terminates, provided that the training set is separable with a nonzero margin.
Feature Map $\Phi$

- SVM is often used with kernels
Large Margin Classifier

Functional margin: $y_i (\mathbf{w} \cdot \mathbf{x}_i) 

Geometric margin: \[
\frac{y_i (\mathbf{w} \cdot \mathbf{x}_i)}{||\mathbf{w}||} = \frac{1}{||\mathbf{w}||} 
\]
Large Margin Classifier

Q1: what if we want functional margin of 2?
Q2: what if we want geometric margin of 1?

SVM objective (max version):

$$\max_w \frac{1}{\|w\|} \quad \text{s.t.} \quad \forall (x, y) \in D, y(w \cdot x) \geq 1$$

max. geometric margin
s.t. functional margin
is at least 1
Large Margin Classifier

SVM objective (min version):

$$\min_w \|w\| \quad \text{s.t.} \quad \forall (x, y) \in D, y(w \cdot x) \geq 1$$

interplation: small models generalize better

$$\langle w, x \rangle \geq 1$$
Large Margin Classifier

SVM objective (min version):

$$\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 \text{ s.t. } \forall (\mathbf{x}, y) \in D, y(\mathbf{w} \cdot \mathbf{x}) \geq 1$$

- $|\mathbf{w}|$ not differentiable, $|\mathbf{w}|^2$ is.
- min. weight vector s.t. functional margin is at least 1
SVM vs. MIRA

• SVM: min weight vector to enforce functional margin of at least 1 on ALL EXAMPLES
• MIRA: min weight change to enforce functional margin of at least 1 on THIS EXAMPLE
• MIRA is 1-step or online approximation of SVM
• Aggressive MIRA → SVM as \( p \to 1 \)

\[
\begin{align*}
\min_w & \frac{1}{2} \|w\|^2 & \text{s.t. } \forall (x, y) \in D, y(w \cdot x) \geq 1 \\
\min_{w'} & \|w' - w\|^2 & \text{s.t. } w' \cdot x \geq 1
\end{align*}
\]
Convex Hull Interpretation

max. distance between convex hulls

Class $B$ | Class $A$

weight vector is determined by the support vectors alone

c.f. perceptron: $w = \sum_{(x,y) \in \text{errors}} y \cdot x$

how many support vectors in 2D?

why don’t use convex hulls for SVMs in practice??

what about MIRA?
Convex set. A point set $C \in \mathbb{R}^d$ is convex if the line segment $[x, y]$ connecting any two points $x$ and $y$ in $C$ lies entirely in $C$.

Convex hull. Smallest convex set containing $C$.

$$\text{ch}(C) := \left\{ \sum \alpha_i x_i : x_i \in C, \alpha_i \geq 0, \sum \alpha_i = 1 \right\}.$$
Optimization

- Primal optimization problem
\[ \min_w \frac{1}{2} \| w \|^2 \quad \text{s.t. } \forall (x, y) \in D, y(w \cdot x) \geq 1 \]

- Convex optimization: convex function over convex set!

- Quadratic prog.: quadratic function w/ linear constraints
MIRA as QP

- MIRA is a trivial QP; can be solved geometrically
- what about multiple constraints (e.g. minibatch)?

\[
\min_{w'} \|w' - w\|^2 \\
\text{s.t. } w' \cdot x \geq 1
\]
Optimization

- Primal optimization problem
\[ \min_{\mathbf{w}} \frac{1}{2} \| \mathbf{w} \|^2 \quad \text{s.t.} \quad \forall (\mathbf{x}, y) \in D, y(\mathbf{w} \cdot \mathbf{x}) \geq 1 \]

- Convex optimization: convex function over convex set!

- Lagrange function
\[ L(\mathbf{w}, \alpha) = \frac{1}{2} \| \mathbf{w} \|^2 - \sum_{i} \alpha_i [y_i \langle \mathbf{x}_i, \mathbf{w} \rangle + b] - 1 \]

Derivatives in \( \mathbf{w} \) need to vanish
\[ \partial_{\mathbf{w}} L(\mathbf{w}, \alpha) = \mathbf{w} - \sum_{i} \alpha_i y_i \mathbf{x}_i = 0 \]
\[ \mathbf{w} = \sum_{i} y_i \alpha_i \mathbf{x}_i \]

Model is a linear combo of a small subset of input (the support vectors) i.e., those with \( \alpha_i > 0 \)
Lagrangian & Saddle Point

- equality: \( \min x^2 \) s.t. \( x = 1 \)
- inequality: \( \min x^2 \) s.t. \( x \geq 1 \)
- Lagrangian: \( L(x, \alpha) = x^2 - \alpha(x-1) \)
- derivative in \( x \) need to vanish
- optimality is at saddle point with \( \alpha \)
- \( \min x \) in primal => \( \max \alpha \) in dual
Constrained Optimization

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \|w\|^2 \\
\text{subject to} & \quad y_i \left[ \langle x_i, w \rangle + b \right] \geq 1
\end{align*}
\]

• Quadratic Programming
  • Quadratic Objective
  • Linear Constraints

KKT condition (complementary slackness)

optimal point is achieved at active constraints

where \( \alpha_i > 0 \) (\( \alpha_i = 0 \) => inactive)

\[
\alpha_i \left[ y_i \left[ \langle w, x_i \rangle + b \right] - 1 \right] = 0
\]
KKT implies Support Vectors

\[
\begin{align*}
\text{minimize} \quad & \frac{1}{2} \|w\|^2 \\
\text{subject to} \quad & y_i [\langle x_i, w \rangle + b] \geq 1
\end{align*}
\]

\[w = \sum_i y_i \alpha_i x_i\]

Karush Kuhn Tucker (KKT) Optimality Condition

\[\alpha_i [y_i [\langle w, x_i \rangle + b] - 1] = 0\]

\[\alpha_i > 0 \implies y_i [\langle w, x_i \rangle + b] = 1\]
Properties

\[ w = \sum_{i} y_i \alpha_i x_i \]

• Weight vector \( w \) as weighted linear combination of instances
• Only points on margin matter (ignore the rest and get same solution)
• Only inner products matter
  • Quadratic program
  • We can replace the inner product by a kernel
• Keeps instances away from the margin
Alternative: Primal $\Rightarrow$ Dual

- Lagrange function
  \[ L(w, \alpha) = \frac{1}{2} \|w\|^2 - \sum \alpha_i [y_i \langle x_i, w \rangle - 1] \]

- Derivatives in \( w \) need to vanish
  \[ \partial_w L(w, \alpha) = w - \sum \alpha_i y_i x_i = 0 \]

  \[ w = \sum_i y_i \alpha_i x_i \]

- Plugging \( w \) back into \( L \) yields
  \[ \max_{\alpha} -\frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle + \sum \alpha_i \]

  subject to dual variables \( \alpha_i \geq 0 \)
Primal vs. Dual

**Primal**

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \|w\|^2 \\
\text{subject to} & \quad y_i \left[\langle x_i, w \rangle + b \right] \geq 1
\end{align*}
\]

\[w = \sum_i y_i \alpha_i x_i\]

**Dual**

\[
\begin{align*}
\text{maximize} & \quad -\frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle + \sum_i \alpha_i \\
\text{subject to} & \quad \text{dual variables} \quad \alpha_i \geq 0
\end{align*}
\]
Solving the optimization problem

• Dual problem

\[
\text{maximize } \alpha \rightarrow -\frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle + \sum_i \alpha_i \\
\text{subject to dual variables } \alpha_i \geq 0
\]

• If problem is small enough (1000s of variables) we can use off-the-shelf solver (CVXOPT, CPLEX, OOQP, LOQO)

• For larger problem use fact that only SVs matter and solve in blocks (active set method).
Quadratic Program in Dual

• Dual problem

\[
\max_{\alpha} - \frac{1}{2} \alpha^T Q \alpha - \alpha^T b
\]

subject to \( \alpha \geq 0 \)

Q: what’s the Q in SVM primal? how about Q in SVM dual?

• Quadratic Programming
  • Objective: Quadratic function
  • Q is positive semidefinite
  • Constraints: Linear functions

• Methods
  • Gradient Descent
  • Coordinate Descent
    • aka., Hildreth Algorithm
  • Sequential Minimal Optimization (SMO)
Convex QP

- if $Q$ is positive (semi)definite, i.e., $x^TQx \geq 0$ for all $x$, then convex QP $\Rightarrow$ local min/max is global min/max
- if $Q = 0$, it reduces to linear programming
- if $Q$ is indefinite $\Rightarrow$ saddlepoint
- general QP is NP-hard: convex QP is polynomial-time
QP: Hildreth Algorithm

• idea 1:
  • update one coordinate while fixing all other coordinates
  • e.g., update coordinate $i$ is to solve:
    $\arg\max_{\alpha_i} - \frac{1}{2} \alpha^T Q \alpha - \alpha^T b$
    subject to $\alpha \geq 0$

Quadratic function with only one variable
Maximum $\Rightarrow$ first-order derivative is 0
QP: Hildreth Algorithm

• idea 2:
  • choose another coordinate and repeat until meet stopping criterion
  • reach maximum or
  • increase between 2 consecutive iterations is very small or
  • after some # of iterations
• how to choose coordinate: sweep pattern
  • Sequential:
    • 1, 2, ..., n, 1, 2, ..., n, ...
    • 1, 2, ..., n, n-1, n-2, ..., 1, 2, ...
  • Random: permutation of 1,2, ..., n
  • Maximal Descent
    • choose i with maximal descent in objecti
QP: Hildreth Algorithm

initialize $\alpha_i = 0$ for all $i$
repeat
pick $i$ following sweep pattern
solve
$\alpha_i \leftarrow \arg\max_{\alpha_i} \left( -\frac{1}{2} \alpha^T Q \alpha - \alpha^T b \right)$
subject to $\alpha \geq 0$
until meet stopping criterion
QP: Hildreth Algorithm

\[
\begin{align*}
\text{maximize} & \quad -\frac{1}{2} \alpha^T \begin{pmatrix} 4 & 1 \\ 1 & 2 \end{pmatrix} \alpha - \alpha^T \begin{pmatrix} -6 \\ -4 \end{pmatrix} \\
\text{subject to} & \quad \alpha \geq 0
\end{align*}
\]

- choose coordinates
- 1, 2, 1, 2, ...
QP: Hildreth Algorithm

• **pros:**
  • extremely simple
  • no gradient calculation
  • easy to implement

• **cons:**
  • converges slow, compared to other methods
  • can’t deal with too many constraints
  • works for minibatch MIRA but not SVM
Linear Separator

Ham

Spam
Large Margin Classifier

$$\langle w, x \rangle \leq -1$$

$$\langle w, x \rangle \geq 1$$

linear function

$$f(x) = \langle w, x \rangle + b$$

linear separator

is impossible
Theorem (Minsky & Papert)
Finding the minimum error separating hyperplane is NP hard
Adding slack variables

Convex optimization problem

\[ \langle w, x \rangle \leq -1 + \xi \]

\[ \langle w, x \rangle \geq 1 - \xi \]

minimize amount of slack
margin violation vs. misclassification

\[ \xi_i \geq 0 \]

- for \( 0 < \xi \leq 1 \) point is between margin and correct side of hyperplane. This is a **margin violation**
- for \( \xi > 1 \) point is **misclassified**

misclassification is also margin violation (\( \xi > 0 \))

\[ \frac{\xi_i}{\|w\|} > \frac{2}{\|w\|} \]

\[ \text{Margin} = \frac{2}{\|w\|} \]
Adding slack variables

• Hard margin problem

\[
\text{minimize } \frac{1}{2} \|w\|^2 \text{ subject to } y_i \langle w, x_i \rangle \geq 1
\]

• With slack variables \( C=0? \) \( C=+\infty? \)

\[
\text{minimize } \frac{1}{2} \|w\|^2 + C \sum_i \xi_i
\]
subject to \( y_i \langle w, x_i \rangle \geq 1 - \xi_i \) and \( \xi_i \geq 0 \)

Problem is always feasible. Proof:

\( w = 0 \) and \( \xi_i = 1 \) (also yields upper bound)

\( C=+\infty \Rightarrow \) not tolerant on violation \( \Rightarrow \) hard margin
• Primal optimization problem
  \[ \minimize_{x} f(x) \text{ subject to } c_i(x) \leq 0 \]

• Lagrange function
  \[ L(x, \alpha) = f(x) + \sum_{i} \alpha_i c_i(x) \]

• First order optimality conditions in \( x \)
  \[ \partial_x L(x, \alpha) = \partial_x f(x) + \sum_{i} \alpha_i \partial_x c_i(x) = 0 \]

• Solve for \( x \) and plug it back into \( L \)
  \[ \maximize_{\alpha} L(x(\alpha), \alpha) \]
  (keep explicit constraints)
• Primal optimization problem

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \|w\|^2 + C \sum_i \xi_i \\
\text{subject to} & \quad y_i \langle w, x_i \rangle + b \geq 1 - \xi_i \text{ and } \xi_i \geq 0
\end{align*}
\]

• Lagrange function

\[
L(w, \xi, \alpha, \eta) = \frac{1}{2} \|w\|^2 + C \sum_i \xi_i - \sum_i \alpha_i [y_i \langle x_i, w \rangle + b] + \xi_i - 1 - \sum_i \eta_i \xi_i
\]

optimality in \((w, \xi)\) is at saddle point with \(\alpha, \eta\)

• Derivatives in \((w, \xi)\) need to vanish
Dual Problem

- **Lagrange function**
  \[
  L(w, \xi, \alpha, \eta) = \frac{1}{2} \|w\|^2 + C \sum_i \xi_i - \sum_i \alpha_i [y_i \langle x_i, w \rangle + b] + \xi_i - 1 - \sum \eta_i \xi_i
  \]

- **Derivatives in \( w \) need to vanish**
  \[
  \partial_w L(w, b, \xi, \alpha, \eta) = w - \sum \alpha_i y_i x_i = 0
  \]

- **Plugging terms back into \( L \) yields**
  \[
  \partial_{\xi_i} L(w, b, \xi, \alpha, \eta) = C - \alpha_i - \eta_i = 0
  \]

- **Bound influence**
  \[
  \max \alpha \quad \text{subject to} \quad \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle + \sum \alpha_i \leq C
  \]
Karush Kuhn Tucker Conditions

\[ L(w, \xi, \alpha, \eta) = \frac{1}{2} \|w\|^2 + C \sum_i \xi_i - \sum_i \alpha_i [y_i \langle x_i, w \rangle + b] + \xi_i - 1] - \sum_i \eta_i \xi_i \]

\[ \partial_w L(w, \xi, \alpha, \eta) = w - \sum_i \alpha_i y_i x_i = 0 \]

\[ w = \sum_i y_i \alpha_i x_i \]

\[ \alpha = 0 \]

\[ \alpha_i [y_i \langle w, x_i \rangle + b] + \xi_i - 1] = 0 \]

\[ \eta_i \xi_i = 0 \]

\[ 0 \leq \alpha_i = C - \eta_i \leq C \]

\[ 0 < \alpha_i < C \implies y_i \langle w, x_i \rangle + b \geq 1 \]

\[ \alpha_i = 0 \implies y_i \langle w, x_i \rangle + b \geq 1 \]

\[ 0 < \alpha_i < C \implies y_i \langle w, x_i \rangle + b = 1 \]

\[ \alpha_i = C \implies y_i \langle w, x_i \rangle + b \leq 1 \]

\[ \alpha_i = C \implies y_i \langle w, x_i \rangle + b \leq 1 \]

why these are not disjoint?
all circled and squared examples are support vectors ($\alpha > 0$) 
yield include $\xi = 0$ (hard-margin SVs), $\xi > 0$ (margin violations), 
and $\xi > 1$ (misclassifications)
Support Vectors and Violations

non-support vectors ($\alpha=0, \xi=0$)

(0<$\alpha$<C, $\xi$$\geq$0)

support vectors

($\alpha$=C, $\xi$>0)

margin violations

($\alpha$=C, $\xi$>1)

misclassifications

all circled and squared examples are support vectors ($\alpha$>0)
they include $\xi$=0 (hard-margin SVs), $\xi$>0 (margin violations),
and $\xi$>1 (misclassifications)
SVM with sklearn

python demo.py 1e10

python demo.py 1e10
SVM with sklearn

C > 0 is a scalar regularization hyperparameter that trades off:

<table>
<thead>
<tr>
<th></th>
<th>small C</th>
<th>big C</th>
</tr>
</thead>
<tbody>
<tr>
<td>desire</td>
<td>maximize margin $1/</td>
<td>w</td>
</tr>
<tr>
<td>danger</td>
<td>underfitting (misclassifies much training data)</td>
<td>overfitting (awesome training, awful test)</td>
</tr>
<tr>
<td>outliers</td>
<td>less sensitive</td>
<td>very sensitive</td>
</tr>
<tr>
<td>boundary</td>
<td>more “flat”</td>
<td>more sinuous</td>
</tr>
</tbody>
</table>
SVM with sklearn

In [2]: clf = svm.SVC(kernel='linear', C=1e10)
In [3]: X = [[1,1], [1,-1], [-1,1], [-1,-1]]
In [4]: Y = [1,1,-1,-1]
In [5]: clf.fit(X, Y)
In [6]: clf.support_
Out[6]: array([3, 1], dtype=int32)
In [7]: clf.dual_coef_
Out[7]: array([[-0.5,  0.5]])
In [8]: clf.coef_
Out[8]: array([[ 1.,  0.]])
In [9]: clf.intercept_
Out[9]: array([-0.])
In [10]: clf.support_vectors_
Out[10]: array([[-1., -1.], [ 1., -1.], [-0.1, -0.1]])
In [11]: clf.n_support_
Out[11]: array([1, 1], dtype=int32)
SVM with sklearn

In [2]: clf = svm.SVC(kernel='linear', C=1e10)
In [3]: X = [[1,1], [1,-1], [-1,1], [-1,-1]]
In [4]: Y = [1,1,-1,-1]
In [5]: clf.fit(X, Y)
In [6]: clf.support_
Out[6]: array([3, 1], dtype=int32)
In [7]: clf.dual_coef_
Out[7]: array([[-0.5,  0.5]])
In [8]: clf.coef_
Out[8]: array([[ 1.,  0.]])
In [9]: clf.intercept_
Out[9]: array([-0.])
In [10]: clf.support_vectors_
Out[10]: array([[ 1., -1.], [-1., -1.], [-0.1, -0.1]])
In [11]: clf.n_support_
Out[11]: array([1, 1], dtype=int32)

In [2]: X = [[1,1], [1,-1], [-1,1], [-1,-1], [-0.1,-0.1]]
In [3]: Y = [1,1,-1,-1, 1]
In [12]: clf = svm.SVC(kernel='linear', C=1e10)
In [13]: clf.fit(X, Y)
In [14]: clf.coef_
Out[14]: array([[ 2.02010102e+00,   1.00999543e-04]])
In [15]: clf.intercept_
Out[15]: array([ 1.02013469])
In [16]: clf.support_vectors_
Out[16]: array([[ 1. ,  1. ],
             [-1. , -1. ],
             [-0.01, -0.01]])
In [17]: clf.dual_coef_
Out[17]: array([[-1.01000001, -1.03050607,  2.04050608]])
In [2]: clf = svm.SVC(kernel='linear', C=1e10)
In [3]: X = [[1,1], [1,-1], [-1,1], [-1,-1]]
In [4]: Y = [1,1,-1,-1]
In [5]: clf.fit(X, Y)
In [6]: clf.support_
Out[6]: array([3, 1], dtype=int32)
In [7]: clf.dual_coef_
Out[7]: array([[-0.5,  0.5]])
In [8]: clf.coef_
Out[8]: array([[ 1.,  0.]])
In [9]: clf.intercept_
Out[9]: array([-0.])
In [10]: clf.support_vectors_
Out[10]: array([[-1., -1.],
             [ 1., -1.],
             [-2.,  0.]])
In [11]: clf.n_support_
Out[11]: array([1, 1], dtype=int32)

In [2]: X = [[1,1], [1,-1], [-1,1], [-1,-1], [-2,0]]
In [3]: Y = [1,1,-1,-1, 1]
In [4]: clf = svm.SVC(kernel='linear', C=1)
In [5]: clf.fit(X, Y)
In [6]: clf.coef_
Out[6]: array([[ 1.,  0.]])
In [7]: clf.support_vectors_
Out[7]: array([[[-1.,  1.],
               [-1., -1.],
               [ 1.,  1.],
               [ 1., -1.],
               [-2.,  0.]]])
In [8]: clf.dual_coef_
Out[8]: array([[-1. , -1. ,  0.5,  0.5,  1. ]])
In [9]: clf.intercept_
Out[9]: array([-0.])

what if C=1e10?
hard vs. soft margins
hard vs. soft margins
From Constrained Optimization to Unconstrained Optimization (back to Primal)

Learning an SVM has been formulated as a constrained optimization problem over \( w \) and \( \xi \)

\[
\min_{w \in \mathbb{R}^d, \xi_i \in \mathbb{R}^+} ||w||^2 + C \sum_{i=1}^{N} \xi_i \text{ subject to } y_i \left( w^T x_i \right) \geq 1 - \xi_i \text{ for } i = 1 \ldots N
\]

The constraint \( y_i \left( w^T x_i \right) \geq 1 - \xi_i \), can be written more concisely as

\[
y_i f(x_i) \geq 1 - \xi_i
\]

which, together with \( \xi_i \geq 0 \), is equivalent to

\[
\xi_i = \max(0, 1 - y_i f(x_i))
\]

Hence the learning problem is equivalent to the unconstrained optimization problem over \( w \)

\[
\min_{w \in \mathbb{R}^d} ||w||^2 + C \sum_{i=1}^{N} \max(0, 1 - y_i f(x_i))
\]

regularization

loss function
Loss function

\[
\min_{w \in \mathbb{R}^d} ||w||^2 + C \sum_{i} \max(0, 1 - y_i f(x_i))
\]

Points are in three categories:

1. \(y_i f(x_i) > 1\)
   - Point is outside margin.
   - No contribution to loss

2. \(y_i f(x_i) = 1\)
   - Point is on margin.
   - No contribution to loss.
   - As in hard margin case.

3. \(y_i f(x_i) < 1\)
   - Point violates margin constraint.
   - Contributes to loss

\((\text{margin violation } \xi > 0, \text{ including misclassification } \xi > 1)\)
Loss functions

- SVM uses “hinge” loss $\max(0, 1 - y_i f(x_i))$
- an approximation to the 0-1 loss

(perceptron uses a shifted hinge-loss touching the origin)
SVM

\[ \min_{\mathbf{w} \in \mathbb{R}^d} C \sum_{i=1}^{N} \max(0, 1 - y_i f(\mathbf{x}_i)) + \|\mathbf{w}\|^2 \quad \text{convex} \]

\[ \text{convex} + \text{convex} = \text{convex!} \]
Gradient (or steepest) descent algorithm for SVM

To minimize a cost function $C(w)$ use the iterative update

$$w_{t+1} \leftarrow w_t - \eta_t \nabla w C(w_t)$$

where $\eta$ is the learning rate.

First, rewrite the optimization problem as an average

$$\min_w C(w) = \frac{\lambda}{2}||w||^2 + \frac{1}{N} \sum_i \max(0, 1 - y_i f(x_i))$$

$$= \frac{1}{N} \sum_i \left( \frac{\lambda}{2}||w||^2 + \max(0, 1 - y_i f(x_i)) \right)$$

(with $\lambda = 2/(NC)$ up to an overall scale of the problem) and $f(x) = w^\top x$.

Because the hinge loss is not differentiable, a sub-gradient is computed.
Sub-gradient for hinge loss

\[
\mathcal{L}(x_i, y_i; w) = \max(0, 1 - y_i f(x_i)) \quad f(x_i) = w^\top x_i
\]

\[
\frac{\partial \mathcal{L}}{\partial w} = -y_i x_i \\
\frac{\partial \mathcal{L}}{\partial w} = 0
\]
Sub-gradient descent algorithm for SVM

\[ C(w) = \frac{1}{N} \sum_{i}^{N} \left( \frac{\lambda}{2} ||w||^2 + \mathcal{L}(x_i, y_i; w) \right) \]

The iterative update is

\[ \begin{align*}
    w_{t+1} & \leftarrow w_t - \eta \nabla w_t C(w_t) \\
    & \leftarrow w_t - \eta \frac{1}{N} \sum_{i}^{N} (\lambda w_t + \nabla_w \mathcal{L}(x_i, y_i; w_t))
\end{align*} \]

where \( \eta \) is the learning rate.

Then each iteration \( t \) involves cycling through the training data with the updates:

\[ \begin{align*}
    w_{t+1} & \leftarrow w_t - \eta (\lambda w_t - y_i x_i) & \text{if } y_i f(x_i) < 1 \\
    & \leftarrow w_t - \eta \lambda w_t & \text{otherwise}
\end{align*} \]

just like perceptron! \( \text{perc: } \leq 0 \)

In the Pegasos algorithm the learning rate is set at \( \eta_t = \frac{1}{\lambda t} \) \( \lambda = 2/(NC) \)
**INPUT:** \( S, \lambda, T \)

**INITIALIZE:** Set \( w_1 = 0 \)

**FOR** \( t = 1, 2, \ldots, T \)

Choose \( i_t \in \{1, \ldots, |S|\} \) uniformly at random.

Set \( \eta_t = \frac{1}{\lambda_t} \)

If \( y_{i_t} \langle w_t, x_{i_t} \rangle < 1 \), then:

Set \( w_{t+1} \leftarrow (1 - \eta_t \lambda)w_t + \eta_t y_{i_t} x_{i_t} \)

Else (if \( y_{i_t} \langle w_t, x_{i_t} \rangle \geq 1 \)):

Set \( w_{t+1} \leftarrow (1 - \eta_t \lambda)w_t \)

**[Optional:]** \( w_{t+1} \leftarrow \min \left\{ 1, \frac{1/\sqrt{\lambda}}{\|w_{t+1}\|} \right\} w_{t+1} \)

**OUTPUT:** \( w_{T+1} \)

\[ \lambda = \frac{2}{(NC)} \]
Pegasos – Stochastic Gradient Descent Algorithm (SGD)

Randomly sample from the training data

SGD is online update: gradient on one example (unbiasedly) approximates the gradient on the whole training data