As for the SVM, we introduce slack variables and maximize margin:

$$\text{minimize}_{w,b} \sum_y w(y) \cdot w(y) + C \sum_j \xi_j$$
$$w(y_j) \cdot x_j + b(y_j) \geq w(y') \cdot x_j + b(y') + 1 - \xi_j, \forall y' \neq y_j, \forall j$$
$$\xi_j \geq 0, \forall j$$

To predict, we use:

$$\hat{y} \leftarrow \arg \max_k w_k \cdot x + b_k$$

Now can we learn it? →

Multi-class SVM
How to deal with imbalanced data?

- In many practical applications we may have **imbalanced** data sets
- We may want errors to be equally distributed between the positive and negative classes
- A slight modification to the SVM objective does the trick!

\[
\min_{w,b} \frac{1}{2}||w||^2 + \frac{C}{N_+} \sum_{j : y_j = +1} \xi_j + \frac{C}{N_-} \sum_{j : y_j = -1} \xi_j
\]

Class-specific weighting of the slack variables
What’s Next!

• Learn one of the most interesting and exciting recent advancements in machine learning
  – The “kernel trick”
  – High dimensional feature spaces at no extra cost!

• But first, a detour
  – Constrained optimization!
Constrained optimization

No Constraint

\( x^* = 0 \)

\( x \geq -1 \)

\( x^* = 0 \)

\( x \geq 1 \)

\( x^* = 1 \)

How do we solve with constraints?

\( \Rightarrow \) Lagrange Multipliers!!!
Lagrange multipliers – Dual variables

Introduce Lagrangian (objective):
\[ L(x, \alpha) = x^2 - \alpha(x - b) \]

Why is this equivalent?
- \( \min \) is fighting \( \max \)!
  - \( x < b \) → \( (x-b) < 0 \) → \( \max_{\alpha} - \alpha(x-b) = \infty \)
    - \( \min \) won’t let this happen!
  - \( x > b, \ \alpha \geq 0 \) → \( (x-b) > 0 \) → \( \max_{\alpha} - \alpha(x-b) = 0, \ \alpha^* = 0 \)
    - \( \min \) is cool with 0, and \( L(x, \alpha) = x^2 \) (original objective)
  - \( x = b \) → \( \alpha \) can be anything, and \( L(x, \alpha) = x^2 \) (original objective)

The \( \min \) on the outside forces \( \max \) to behave, so constraints will be satisfied.
Dual SVM derivation (1) – the linearly separable case

Original optimization problem:

\[
\begin{align*}
\text{minimize}_{w,b} & \quad \frac{1}{2}w \cdot w \\
\left( w \cdot x_j + b \right)y_j & \geq 1, \quad \forall j
\end{align*}
\]

Rewrite constraints

One Lagrange multiplier per example

Lagrangian:

\[
L(w, \alpha) = \frac{1}{2}w \cdot w - \sum_j \alpha_j \left( \left( w \cdot x_j + b \right)y_j - 1 \right)
\]

\[
\alpha_j \geq 0, \quad \forall j
\]

Our goal now is to solve:

\[
\begin{align*}
\min_{\vec{w}, b} & \quad \max_{\vec{\alpha} \geq 0} L(\vec{w}, \vec{\alpha})
\end{align*}
\]
Dual SVM derivation (2) – the linearly separable case

(Primal) \[
\min_{\vec{w}, b} \max_{\alpha \geq 0} \left[ \frac{1}{2} ||\vec{w}||^2 - \sum_j \alpha_j \left[ (\vec{w} \cdot \vec{x}_j + b) y_j - 1 \right] \right]
\]

Swap min and max

(Dual) \[
\max_{\alpha \geq 0} \min_{\vec{w}, b} \left[ \frac{1}{2} ||\vec{w}||^2 - \sum_j \alpha_j \left[ (\vec{w} \cdot \vec{x}_j + b) y_j - 1 \right] \right]
\]

Slater’s condition from convex optimization guarantees that these two optimization problems are equivalent!
Dual SVM derivation (3) – the linearly separable case

(Dual) \( \max_{\alpha \geq 0} \min_{\vec{w}, b} \frac{1}{2} ||\vec{w}||^2 - \sum_j \alpha_j [(\vec{w} \cdot \vec{x}_j + b) y_j - 1] \)

Can solve for optimal \( \vec{w}, b \) as function of \( \alpha \):

\[ \frac{\partial L}{\partial \vec{w}} = \vec{w} - \sum_j \alpha_j y_j \vec{x}_j \quad \Rightarrow \quad \vec{w} = \sum_j \alpha_j y_j \vec{x}_j \]

\[ \frac{\partial L}{\partial b} = - \sum_j \alpha_j y_j \quad \Rightarrow \quad \sum_j \alpha_j y_j = 0 \]

Substituting these values back in (and simplifying), we obtain:

(Dual) \( \bar{\alpha} \geq 0, \sum_j \alpha_j y_j = 0 \) \( \sum_j \alpha_j - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j (\vec{x}_i \cdot \vec{x}_j) \)

Sums over all training examples scalars dot product
Dual SVM derivation (3) – the linearly separable case

(Dual) \[ \max_{\alpha \geq 0} \min_{\vec{w}, b} \frac{1}{2} \| \vec{w} \|^2 - \sum_j \alpha_j [(\vec{w} \cdot \vec{x}_j + b) y_j - 1] \]

Can solve for optimal \( \vec{w}, b \) as function of \( \alpha \):

\[
\frac{\partial L}{\partial \vec{w}} = \vec{w} - \sum_j \alpha_j y_j \vec{x}_j \quad \Rightarrow \quad \vec{w} = \sum_j \alpha_j y_j \vec{x}_j
\]

\[
\frac{\partial L}{\partial b} = -\sum_j \alpha_j y_j \quad \Rightarrow \quad \sum_j \alpha_j y_j = 0
\]

Substituting these values back in (and simplifying), we obtain:

(Dual) \[ \max_{\vec{\alpha} \geq 0, \sum_j \alpha_j y_j = 0} \sum_j \alpha_j - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j (\vec{x}_i \cdot \vec{x}_j) \]

So, in dual formulation we will solve for \( \vec{\alpha} \) directly!

- \( \vec{w} \) and \( b \) are computed from \( \vec{\alpha} \) (if needed)
Dual SVM derivation (3) – the linearly separable case

Lagrangian:
\[ L(w, \alpha) = \frac{1}{2} w \cdot w - \sum_j \alpha_j \left[ (w \cdot x_j + b) y_j - 1 \right] \]
\[ \alpha_j \geq 0, \quad \forall j \]

\( \alpha_j > 0 \) for some \( j \) implies constraint is tight. We use this to obtain \( b \):

\[ y_j (\tilde{w} \cdot \tilde{x}_j + b) = 1 \] (1)
\[ y_j y_j (\tilde{w} \cdot \tilde{x}_j + b) = y_j \] (2)
\[ (\tilde{w} \cdot \tilde{x}_j + b) = y_j \] (3)

\[ w = \sum_i \alpha_i y_i x_i \]
\[ b = y_k - w \cdot x_k \]
for any \( k \) where \( \alpha_k > 0 \)
Dual for the non-separable case – same basic story (we will skip details)

Primal:  
\[
\begin{align*}
\text{minimize}_{w,b} & \quad \frac{1}{2}w \cdot w + C \sum_j \xi_j \\
(w \cdot x_j + b) y_j & \geq 1 - \xi_j, \quad \forall j \\
\xi_j & \geq 0, \quad \forall j
\end{align*}
\]

Dual:  
\[
\begin{align*}
\text{maximize}_\alpha & \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i x_j \\
\sum_i \alpha_i y_i & = 0 \\
C & \geq \alpha_i \geq 0
\end{align*}
\]

Solve for \( w, b, \alpha \):  
\[
\begin{align*}
w & = \sum_i \alpha_i y_i x_i \\
b & = y_k - w \cdot x_k
\end{align*}
\]

for any \( k \) where \( C > \alpha_k > 0 \)

What changed?  
- Added upper bound of \( C \) on \( \alpha_i \)!
- Intuitive explanation:  
  - Without slack, \( \alpha_i \to \infty \) when constraints are violated (points misclassified)
  - Upper bound of \( C \) limits the \( \alpha_i \), so misclassifications are allowed
Wait a minute: why did we learn about the dual SVM?

• There are some quadratic programming algorithms that can solve the dual faster than the primal
  – At least for small datasets

• But, more importantly, the “kernel trick”!!!
Reminder: What if the data is not linearly separable?

Use features of features of features of features....

Feature space can get really large really quickly!

\[ \phi(x) = \begin{pmatrix} x^{(1)} \\ \vdots \\ x^{(n)} \\ x^{(1)}x^{(2)} \\ x^{(1)}x^{(3)} \\ \vdots \\ e^{x^{(1)}} \\ \vdots \end{pmatrix} \]
Higher order polynomials

\[
\text{num. terms} = \binom{d + m - 1}{d} = \frac{(d + m - 1)!}{d!(m - 1)!}
\]

- \(d\) – degree of polynomial
- \(m\) – input features

\(d = 6, m = 100\)

about 1.6 billion terms

grows fast!
Dual formulation only depends on dot-products, not on \( \mathbf{w} \! \)!

\[
\begin{align*}
\text{maximize}_{\alpha} & \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j \\
\sum_i \alpha_i y_i &= 0 \\
C &\geq \alpha_i \geq 0
\end{align*}
\]

First, we introduce features:

\[ \mathbf{x}_i \mathbf{x}_j \rightarrow \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \]

Next, replace the dot product with a Kernel:

\[
\begin{align*}
\text{maximize}_{\alpha} & \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\
K(\mathbf{x}_i, \mathbf{x}_j) &= \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \\
\sum_i \alpha_i y_i &= 0 \\
C &\geq \alpha_i \geq 0
\end{align*}
\]

Why is this useful???
Efficient dot-product of polynomials

Polynomials of degree exactly $d$

$d=1$

$$\phi(u).\phi(v) = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = u_1 v_1 + u_2 v_2 = u.v$$

$d=2$

$$\phi(u).\phi(v) = \begin{pmatrix} u_1^2 \\ u_1 u_2 \\ u_2 u_1 \\ u_2^2 \end{pmatrix} \cdot \begin{pmatrix} v_1^2 \\ v_1 v_2 \\ v_2 v_1 \\ v_2^2 \end{pmatrix} = u_1^2 v_1^2 + 2u_1 v_1 u_2 v_2 + u_2^2 v_2^2$$

$$= (u_1 v_1 + u_2 v_2)^2$$

$$= (u.v)^2$$

For any $d$ (we will skip proof):

$$\phi(u).\phi(v) = (u.v)^d$$

- Cool! Taking a dot product and exponentiating gives same results as mapping into high dimensional space and then taking dot produce
Finally: the “kernel trick”!

\[
\text{maximize}_\alpha \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j)
\]

\[
K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)
\]

\[
\sum_i \alpha_i y_i = 0
\]

\[
C \geq \alpha_i \geq 0
\]

- Never compute features explicitly!!!
  - Compute dot products in closed form
- Constant-time high-dimensional dot-products for many classes of features
- But, \(O(n^2)\) time in size of dataset to compute objective
  - Naïve implements slow
  - much work on speeding up

\[
w = \sum_i \alpha_i y_i \Phi(x_i)
\]

\[
b = y_k - w \cdot \Phi(x_k)
\]
Common kernels

- Polynomials of degree exactly $d$
  \[ K(u, v) = (u \cdot v)^d \]

- Polynomials of degree up to $d$
  \[ K(u, v) = (u \cdot v + 1)^d \]

- Gaussian kernels
  \[
  K(\vec{u}, \vec{v}) = \exp \left( - \frac{||\vec{u} - \vec{v}||_2^2}{2\sigma^2} \right)
  \]

- Sigmoid
  \[ K(u, v) = \tanh(\eta u \cdot v + \nu) \]

- And many others: very active area of research!