From the Perceptron to SVMs

Warning: This file contains dynamic content which will be lost in PDF format.
Please use TeXmacs to view the original file

Miguel de Benito Delgado

Contents

The background example: CIFAR10

- Image classification: **10000** images, **10** categories.

The background example: CIFAR10

- Image classification: **10000** images, **10** categories.
- Each image: 32x32 pixels x24 bits = 3072 bytes

- Image classification: **10000** images, **10** categories.
- Each image: 32x32 pixels x24 bits = 3072 bytes

  $\Rightarrow$ Images are points $x \in \mathbb{R}^D, D = 3072$.

- Image classification: **10000** images, **10** categories.
- Each image: 32x32 pixels x24 bits = 3072 bytes

  $\Rightarrow$ Images are points $x \in \mathbb{R}^D, D = 3072$.
- 50000 images already labeled:

- Image classification: **10000** images, **10** categories.
- Each image: 32x32 pixels x24 bits = 3072 bytes

  $\Rightarrow$ Images are points $x \in \mathbb{R}^D, D = 3072$.
- 50000 images already labeled:

  Training set: $\{x_i, y_i\}_{i=1,\ldots,N}, N = 50000, y_i \in \{1, \ldots, 10\}$.

- Image classification: **10000** images, **10** categories.
- Each image: 32x32 pixels x24 bits = 3072 bytes

  $\Rightarrow$ Images are points $x \in \mathbb{R}^D, D = 3072$.
- 50000 images already labeled:

  Training set: $\{x_i, y_i\}_{i=1,\dots,N}, N = 50000, y_i \in \{1, \dots, 10\}$.

  Test set: $\{x_i, ??\}_{i=1,\dots,10000}$.

- Image classification: **10000** images, **10** categories.
- Each image: 32x32 pixels x24 bits = 3072 bytes

  $\Rightarrow$ Images are points $x \in \mathbb{R}^D, D = 3072$.
- 50000 images already labeled:

  Training set: $\{x_i, y_i\}_{i=1,\dots,N}, N = 50000, y_i \in \{1, \dots, 10\}$.

  Test set: $\{x_i, ??\}_{i=1,\dots,10000}$.
- Goal: find map $x \mapsto y(x) \in \{1, \dots, 10\}$, optimal in some sense.

- Image classification: **10000** images, **10** categories.
- Each image: 32x32 pixels x24 bits = 3072 bytes

  $\Rightarrow$ Images are points $x \in \mathbb{R}^D, D = 3072$.
- 50000 images already labeled:

  Training set: $\{x_i, y_i\}_{i=1,\ldots,N}, N = 50000, y_i \in \{1, \ldots, 10\}$.

  Test set: $\{x_i, ??\}_{i=1,\ldots,10000}$.
- Goal: find map $x \mapsto y(x) \in \{1, \ldots, 10\}$, optimal in some sense.

  Training stage, test stage.

Roadmap: Four attempts at classification

- Round 1: Naive attempts

- Round 1: Naive attempts

  l^p distance. Least squares.

Roadmap: Four attempts at classification

- Round 1: Naive attempts

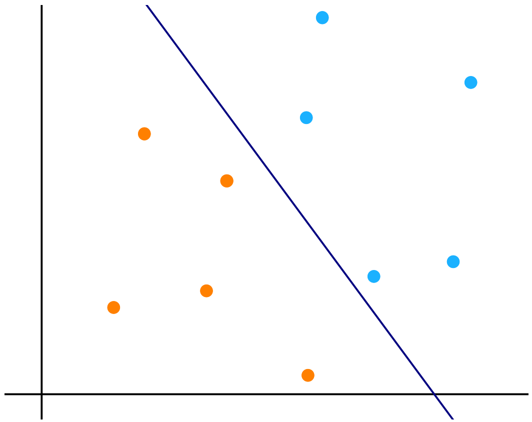
  l^p distance. Least squares.

- Round 2: Linear classifiers

- Round 1: Naive attempts

  l^p distance. Least squares.

- Round 2: Linear classifiers

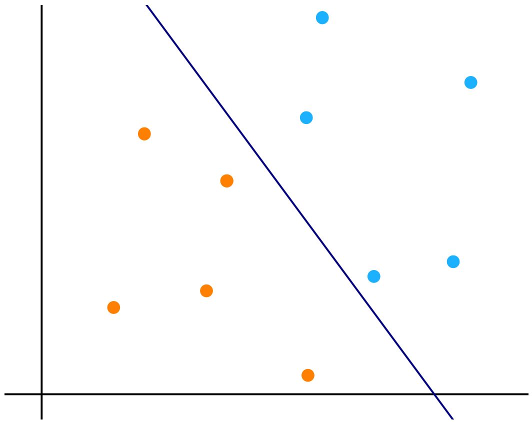


Roadmap: Four attempts at classification

- Round 1: Naive attempts

  l^p distance. Least squares.

- Round 2: Linear classifiers

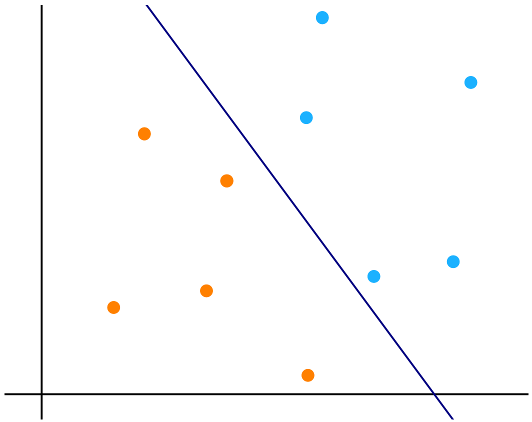


- Round 3: Sloppy linear classifiers

- Round 1: Naive attempts

  l^p distance. Least squares.

- Round 2: Linear classifiers



- Round 3: Sloppy linear classifiers
- Round 4: (Kind of) Non linear classifiers

- Easiest technique.

- Easiest technique.
- Training: “Nothing” to do! ✓

- Easiest technique.
- Training: “Nothing” to do! ✓
- Testing:

- Easiest technique.
- Training: "Nothing" to do! ✓
- Testing:
  1. Take a test sample x.
  2. Compute $d(x, x_i)$ **for every** training sample x_i. Set: $i^\star = \operatorname{argmin}_{i=1,\dots,N} d(x, x_i)$
  3. Assign $x \mapsto y(x_{i^\star})$ (label of closest match).

- Easiest technique.
- Training: “Nothing” to do! ✓
- Testing:
  1. Take a test sample x.
  2. Compute $d(x, x_i)$ **for every** training sample x_i. Set: $i^\star = \underset{i=1,\dots,N}{\operatorname{argmin}}\, d(x, x_i)$
  3. Assign $x \mapsto y(x_{i^\star})$ (label of closest match).
- This sucks. Badly. $\sim 60\%$ error rate for K-nearest neighbours. Why?

- Easiest technique.
- Training: "Nothing" to do! ✓
- Testing:
  1. Take a test sample x.
  2. Compute $d(x, x_i)$ **for every** training sample x_i. Set: $i^\star = \operatorname{argmin}_{i=1,\dots,N} d(x, x_i)$
  3. Assign $x \mapsto y(x_{i^\star})$ (label of closest match).
- This sucks. Badly. $\sim 60\%$ error rate for K-nearest neighbours. Why?

Take some random guy:  And do some nice modification:

- Easiest technique.
- Training: "Nothing" to do! ✓
- Testing:
  1. Take a test sample x.
  2. Compute $d(x, x_i)$ **for every** training sample x_i. Set: $i^\star = \operatorname*{argmin}_{i=1,\dots,N} d(x, x_i)$
  3. Assign $x \mapsto y(x_{i^\star})$ (label of closest match).
- This sucks. Badly. $\sim 60\%$ error rate for K-nearest neighbours. Why?

Take some random guy:  And do some nice modification: 

- Easiest technique.
- Training: "Nothing" to do! ✓
- Testing:
  1. Take a test sample x.
  2. Compute $d(x, x_i)$ **for every** training sample x_i. Set: $i^\star = \underset{i=1,\dots,N}{\operatorname{argmin}}\, d(x, x_i)$
  3. Assign $x \mapsto y(x_{i^\star})$ (label of closest match).
- This sucks. Badly. $\sim 60\%$ error rate for K-nearest neighbours. Why?

  Take some random guy:  And do some nice modification: 
- Fix it or forget about it?

- Easiest technique.
- Training: "Nothing" to do! ✓
- Testing:
  1. Take a test sample x.
  2. Compute $d(x, x_i)$ **for every** training sample x_i. Set: $i^\star = \underset{i=1,\ldots,N}{\operatorname{argmin}} \, d(x, x_i)$
  3. Assign $x \mapsto y(x_{i^\star})$ (label of closest match).
- This sucks. Badly. $\sim 60\%$ error rate for K-nearest neighbours. Why?

  Take some random guy:  And do some nice modification: 
- Fix it or forget about it?

  High testing time, poor performance, lots of memory required.

- 1-of-K encoding for labels: $y = (0, ..., 1, ..., 0) \in \{0, 1\}^K$. All training labels: $Y \in \mathbb{R}^{N \times K}$

- 1-of-K encoding for labels: $y = (0, ..., 1, ..., 0) \in \{0, 1\}^K$. All training labels: $Y \in \mathbb{R}^{N \times K}$
- Each class has a linear model

$$y_k(\overline{x}) = w_k \cdot x + b = \overline{w}_k \cdot \overline{x}, \text{ where } \overline{x} = (x, 1), \overline{w} = (w, b) \in \mathbb{R}^{D+1}.$$

- 1-of-K encoding for labels: $y = (0, ..., 1, ..., 0) \in \{0, 1\}^K$. All training labels: $Y \in \mathbb{R}^{N \times K}$
- Each class has a linear model

$$y_k(\overline{x}) = w_k \cdot x + b = \overline{w}_k \cdot \overline{x}, \text{ where } \overline{x} = (x, 1), \overline{w} = (w, b) \in \mathbb{R}^{D+1}.$$

- Bringing all of them together, after training the parameters $W \in \mathbb{R}^{(D+1) \times K}$:

$$y(\overline{x}) = W^T \overline{x} = \text{“class scores” for datum } \overline{x} \in \mathbb{R}^{D+1}.$$

The class predicted is $\text{argmax}\{y_k(\overline{x}) : k = 1, ..., K\}$.

- 1-of-K encoding for labels: $y = (0, ..., 1, ..., 0) \in \{0, 1\}^K$. All training labels: $Y \in \mathbb{R}^{N \times K}$
- Each class has a linear model

$$y_k(\overline{x}) = w_k \cdot x + b = \overline{w}_k \cdot \overline{x}, \text{ where } \overline{x} = (x, 1), \overline{w} = (w, b) \in \mathbb{R}^{D+1}.$$

- Bringing all of them together, after training the parameters $W \in \mathbb{R}^{(D+1) \times K}$:

$$y(\overline{x}) = W^T \overline{x} = \text{“class scores” for datum } \overline{x} \in \mathbb{R}^{D+1}.$$

  The class predicted is $\operatorname{argmax}\{y_k(\overline{x}) : k = 1, ..., K\}$.
- To train, minimise $E(W) = \frac{1}{2} |X W - Y|^2$, $X \in \mathbb{R}^{N \times K}$ all training samples,

- 1-of-K encoding for labels: $y = (0, \ldots, 1, \ldots, 0) \in \{0, 1\}^K$. All training labels: $Y \in \mathbb{R}^{N \times K}$
- Each class has a linear model

$$y_k(\overline{x}) = w_k \cdot x + b = \overline{w}_k \cdot \overline{x}, \text{ where } \overline{x} = (x, 1), \overline{w} = (w, b) \in \mathbb{R}^{D+1}.$$

- Bringing all of them together, after training the parameters $W \in \mathbb{R}^{(D+1) \times K}$:

$$y(\overline{x}) = W^T \overline{x} = \text{“class scores” for datum } \overline{x} \in \mathbb{R}^{D+1}.$$

  The class predicted is $\operatorname{argmax}\{y_k(\overline{x}) : k = 1, \ldots, K\}$.

- To train, minimise $E(W) = \frac{1}{2} |X W - Y|^2$, $X \in \mathbb{R}^{N \times K}$ all training samples,

$$\rightsquigarrow \quad W^* = (X^\top X)^{-1} X^\top Y = X^\dagger Y.$$

- 1-of-K encoding for labels: $y = (0, \ldots, 1, \ldots, 0) \in \{0, 1\}^K$. All training labels: $Y \in \mathbb{R}^{N \times K}$
- Each class has a linear model

$$y_k(\overline{x}) = w_k \cdot x + b = \overline{w}_k \cdot \overline{x}, \text{ where } \overline{x} = (x, 1), \overline{w} = (w, b) \in \mathbb{R}^{D+1}.$$

- Bringing all of them together, after training the parameters $W \in \mathbb{R}^{(D+1) \times K}$:

$$y(\overline{x}) = W^T \overline{x} = \text{"class scores" for datum } \overline{x} \in \mathbb{R}^{D+1}.$$

  The class predicted is $\operatorname{argmax}\{y_k(\overline{x}) : k = 1, \ldots, K\}$.
- To train, minimise $E(W) = \frac{1}{2} |X W - Y|^2$, $X \in \mathbb{R}^{N \times K}$ all training samples,

$$\rightsquigarrow \quad W^* = (X^\top X)^{-1} X^\top Y = X^\dagger Y.$$

- Prediction: $y(x) = Y^\top X^{\dagger^\top} \overline{x}$. Easy but mostly wrong.

- Start easy

- Start easy

  Only $K = 2$ classes. Labels $y \in \{-1, +1\}$.

- Start easy

  Only $K = 2$ classes. Labels $y \in \{-1, +1\}$.

  Assume **linearly separable** data:

  There exists $\overline{w}^{\star} = (w^{\star}, b^{\star}) \in \mathbb{R}^{D+1}$ *s.t. for every training sample* $x_i \in \mathcal{T}$ *with label* y_i*:*

$$\overline{w}^{\star} \cdot \overline{x}_i > 0 \Leftrightarrow y_i = +1 \ \textit{and} \ \overline{w}^{\star} \cdot \overline{x}_i < 0 \Leftrightarrow y_i = -1.$$

  Put another way, we assume that:

$$y_i\left(\overline{w}^{\star} \cdot \overline{x}_i\right) > 0 \text{ for every } i = 1, ..., N.$$

- Start easy

  Only $K = 2$ classes. Labels $y \in \{-1, +1\}$.

  Assume **linearly separable** data:

  There exists $\overline{w}^\star = (w^\star, b^\star) \in \mathbb{R}^{D+1}$ *s.t. for every training sample* $x_i \in \mathcal{T}$ *with label* y_i*:*

$$\overline{w}^\star \cdot \overline{x}_i > 0 \Leftrightarrow y_i = +1 \; and \; \overline{w}^\star \cdot \overline{x}_i < 0 \Leftrightarrow y_i = -1.$$

  Put another way, we assume that:

$$y_i \left(\overline{w}^\star \cdot \overline{x}_i\right) > 0 \text{ for every } i = 1, ..., N.$$

- The model is: $y_w(x) = h(\overline{w} \cdot \overline{x})$, with $h = \chi_{[0,\infty)} - \chi_{(-\infty,0)}$.

- Start easy

  Only $K=2$ classes. Labels $y \in \{-1,+1\}$.

  Assume **linearly separable** data:

  There exists $\overline{w}^\star = (w^\star, b^\star) \in \mathbb{R}^{D+1}$ *s.t. for every training sample* $x_i \in \mathcal{T}$ *with label* y_i*:*

$$\overline{w}^\star \cdot \overline{x}_i > 0 \Leftrightarrow y_i = +1 \; and \; \overline{w}^\star \cdot \overline{x}_i < 0 \Leftrightarrow y_i = -1.$$

  Put another way, we assume that:

$$y_i \left(\overline{w}^\star \cdot \overline{x}_i\right) > 0 \text{ for every } i = 1, \dots, N.$$

- The model is: $y_w(x) = h(\overline{w} \cdot \overline{x})$, with $h = \chi_{[0,\infty)} - \chi_{(-\infty,0)}$.
- How to determine w?

- Start easy

  Only $K=2$ classes. Labels $y \in \{-1,+1\}$.

  Assume **linearly separable** data:

  There exists $\overline{w}^\star = (w^\star, b^\star) \in \mathbb{R}^{D+1}$ *s.t. for every training sample* $x_i \in \mathcal{T}$ *with label* y_i*:*

  $$\overline{w}^\star \cdot \overline{x}_i > 0 \Leftrightarrow y_i = +1 \ \textit{and} \ \overline{w}^\star \cdot \overline{x}_i < 0 \Leftrightarrow y_i = -1.$$

  Put another way, we assume that:

  $$y_i \left(\overline{w}^\star \cdot \overline{x}_i\right) > 0 \text{ for every } i = 1, \ldots, N.$$

- The model is: $y_w(x) = h(\overline{w} \cdot \overline{x})$, with $h = \chi_{[0,\infty)} - \chi_{(-\infty,0)}$.
- How to determine w? Define $\mathcal{M} := \{x_i \in \mathcal{T} : y_w(x_i) \neq y_i\}$, minimise $|\mathcal{M}|$?

- Start easy

  Only $K=2$ classes. Labels $y \in \{-1,+1\}$.

  Assume **linearly separable** data:

  There exists $\overline{w}^\star = (w^\star, b^\star) \in \mathbb{R}^{D+1}$ *s.t. for every training sample* $x_i \in \mathcal{T}$ *with label* y_i*:*

  $$\overline{w}^\star \cdot \overline{x}_i > 0 \Leftrightarrow y_i = +1 \ \textit{and} \ \overline{w}^\star \cdot \overline{x}_i < 0 \Leftrightarrow y_i = -1.$$

  Put another way, we assume that:

  $$y_i \left(\overline{w}^\star \cdot \overline{x}_i\right) > 0 \text{ for every } i = 1, \ldots, N.$$

- The model is: $y_w(x) = h(\overline{w} \cdot \overline{x})$, with $h = \chi_{[0,\infty)} - \chi_{(-\infty,0)}$.
- How to determine w? Define $\mathcal{M} := \{x_i \in \mathcal{T} : y_w(x_i) \neq y_i\}$, minimise $|\mathcal{M}|$? No!

- **Perceptron criterion**

- **Perceptron criterion**

  For some $\overline{w} \in \mathbb{R}^{D+1}$ and any sample x:

$$x \text{ is correctly classified by } \overline{w} \Leftrightarrow y_i\,\overline{w} \cdot \overline{x} > 0.$$

- **Perceptron criterion**

  For some $\overline{w} \in \mathbb{R}^{D+1}$ and any sample x:

$$x \text{ is correctly classified by } \overline{w} \Leftrightarrow y_i \, \overline{w} \cdot \overline{x} > 0.$$

  The error is $\sum_{x_i \in \mathcal{M}} y_i \, \overline{w} \cdot \overline{x} < 0$, so we want to **minimise**

$$E_{\text{per}}(\overline{w}) := -\sum_{x_i \in \mathcal{M}} y_i \, \overline{w} \cdot \overline{x}_i.$$

- **Perceptron criterion**

For some $\overline{w} \in \mathbb{R}^{D+1}$ and any sample x:

$$x \text{ is correctly classified by } \overline{w} \Leftrightarrow y_i\, \overline{w} \cdot \overline{x} > 0.$$

The error is $\sum_{x_i \in \mathcal{M}} y_i\, \overline{w} \cdot \overline{x} < 0$, so we want to **minimise**

$$E_{\text{per}}(\overline{w}) := -\sum_{x_i \in \mathcal{M}} y_i\, \overline{w} \cdot \overline{x}_i.$$

Alternatively:

$$E_{\text{per}}(\overline{w}) = \sum_{i=1}^{N} \max\{0, -y_i\, \overline{w} \cdot \overline{x}_i\}$$

- **Optimization** of the Perceptron criterion.

- **Optimization** of the Perceptron criterion.

  With **S**tochastic **G**radient **D**escent (more later): Pick $x_i \in \mathcal{M}$ randomly, update:

$$\overline{w}_{t+1} = \overline{w}_t - \lambda_t \nabla E^i_{\text{per}}(\overline{w}_t) = \overline{w}_t + \lambda_t y_i \overline{x}_i.$$

- **Optimization** of the Perceptron criterion.

  With **S**tochastic **G**radient **D**escent (more later): Pick $x_i \in \mathcal{M}$ randomly, update:

$$\overline{w}_{t+1} = \overline{w}_t - \lambda_t \nabla E^i_{\text{per}}(\overline{w}_t) = \overline{w}_t + \lambda_t\, y_i\, \overline{x}_i.$$

- **Interpretation:** $\overline{w}$ is adjusted to account for misclassifications only.

- **Optimization** of the Perceptron criterion.

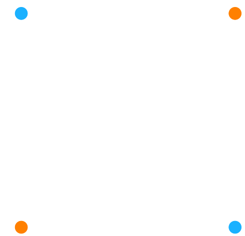
  With **S**tochastic **G**radient **D**escent (more later): Pick $x_i \in \mathcal{M}$ randomly, update:

$$\overline{w}_{t+1} = \overline{w}_t - \lambda_t \nabla E^i_{\text{per}}(\overline{w}_t) = \overline{w}_t + \lambda_t \, y_i \, \overline{x}_i.$$

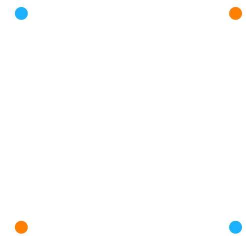
- **Interpretation:** $\overline{w}$ is adjusted to account for misclassifications only.
- **Properties:**
  a) Each step **not** guaranteed to reduce *overall* error.
  b) Convergence guaranteed to *some* solution **if** data linearly separable.
  c) Solution will depend on w_0, b_0.
  d) Doesn't minimise generalisation error $\Rightarrow$ worse generalisation.

- Essential assumption very easy to violate:

- Essential assumption very easy to violate:

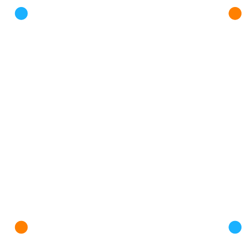


- Essential assumption very easy to violate:



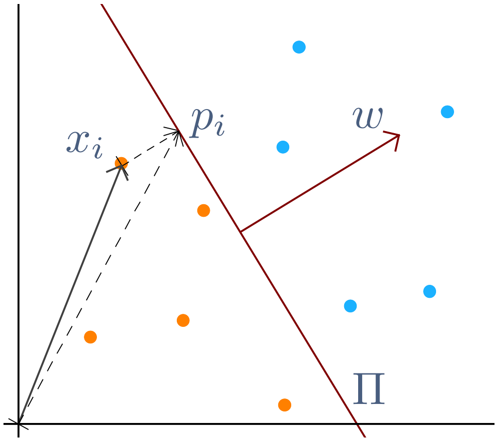
- The perceptron will fail miserably.

- Essential assumption very easy to violate:



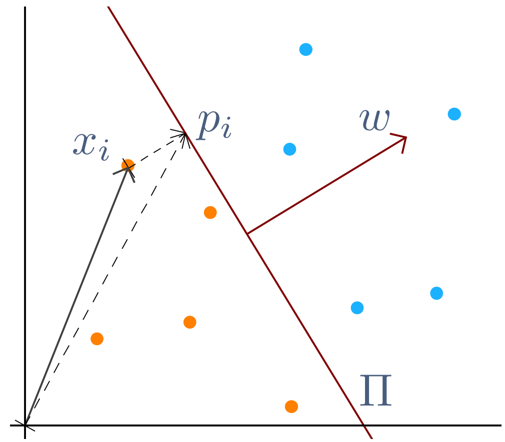
- The perceptron will fail miserably.
- Let's try to fix it.

Let's compute the distance to a hyperplane $\Pi := \{x \in \mathbb{R}^D : w \cdot x + b = 0\}$.



Projection of x_i onto Π: $p_i = x_i \pm \tilde{\gamma}_i \frac{w}{|w|}$, $\tilde{\gamma}_i := |p_i - x_i|$.

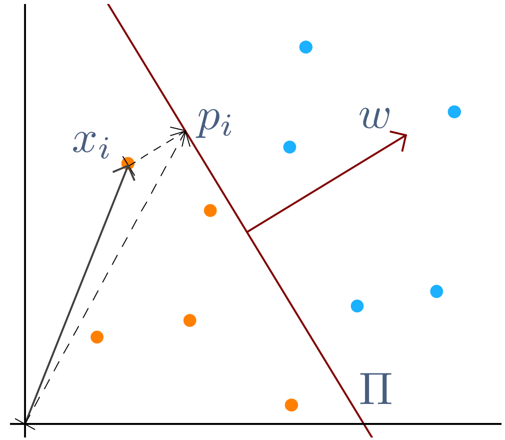
Let's compute the distance to a hyperplane $\Pi := \{x \in \mathbb{R}^D \colon w \cdot x + b = 0\}$.



Projection of x_i onto Π: $p_i = x_i \pm \tilde{\gamma}_i \frac{w}{|w|}$, $\tilde{\gamma}_i := |p_i - x_i|$.

- $p_i \in \Pi \Rightarrow w \cdot p_i + b = 0 \Rightarrow w \cdot x_i \pm \tilde{\gamma}_i \, |w| + b = 0 \Rightarrow \tilde{\gamma}_i = \mp \left(\frac{w}{|w|} \cdot x_i + \frac{b}{|w|} \right) \geqslant 0$.

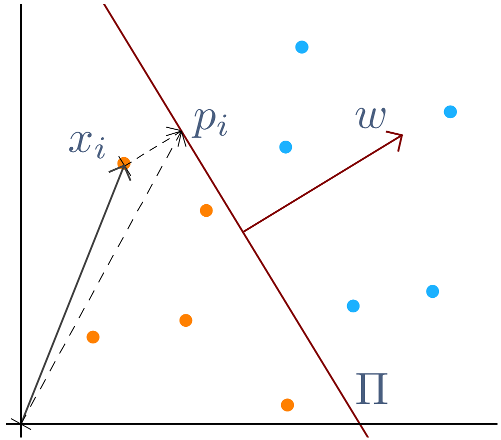
Let's compute the distance to a hyperplane $\Pi := \{x \in \mathbb{R}^D : w \cdot x + b = 0\}$.



Projection of x_i onto Π: $p_i = x_i \pm \tilde{\gamma}_i \frac{w}{|w|}$, $\tilde{\gamma}_i := |p_i - x_i|$.

- $p_i \in \Pi \Rightarrow w \cdot p_i + b = 0 \Rightarrow w \cdot x_i \pm \tilde{\gamma}_i\, |w| + b = 0 \Rightarrow \tilde{\gamma}_i = \mp\left(\frac{w}{|w|} \cdot x_i + \frac{b}{|w|}\right) \geqslant 0$.

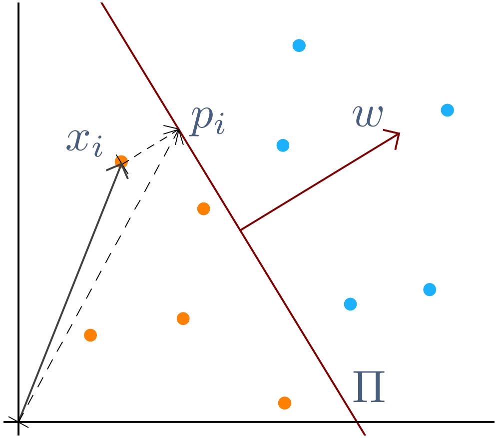
Let's compute the distance to a hyperplane $\Pi := \{x \in \mathbb{R}^D : w \cdot x + b = 0\}$.



Projection of x_i onto Π: $p_i = x_i \pm \tilde{\gamma}_i \frac{w}{|w|}$, $\tilde{\gamma}_i := |p_i - x_i|$.

- $p_i \in \Pi \Rightarrow w \cdot p_i + b = 0 \Rightarrow w \cdot x_i \pm \tilde{\gamma}_i\,|w| + b = 0 \Rightarrow \tilde{\gamma}_i = \mp\left(\frac{w}{|w|} \cdot x_i + \frac{b}{|w|}\right) \geqslant 0$.

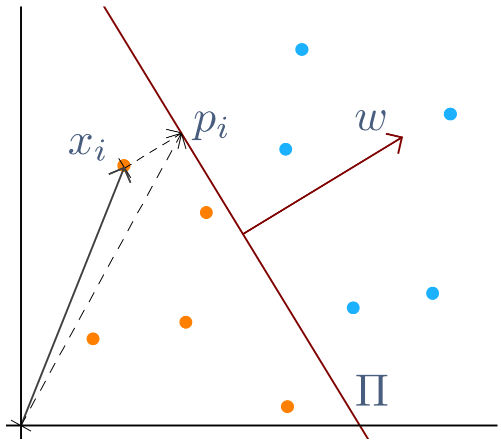
Let's compute the distance to a hyperplane $\Pi := \{x \in \mathbb{R}^D \colon w \cdot x + b = 0\}$.



Projection of x_i onto Π: $p_i = x_i \pm \tilde{\gamma}_i \frac{w}{|w|}$, $\tilde{\gamma}_i := |p_i - x_i|$.

- $p_i \in \Pi \Rightarrow w \cdot p_i + b = 0 \Rightarrow w \cdot x_i \pm \tilde{\gamma}_i\, |w| + b = 0 \Rightarrow \tilde{\gamma}_i = \mp\left(\frac{w}{|w|} \cdot x_i + \frac{b}{|w|}\right) \geqslant 0$.

Let's compute the distance to a hyperplane $\Pi := \{x \in \mathbb{R}^D : w \cdot x + b = 0\}$.

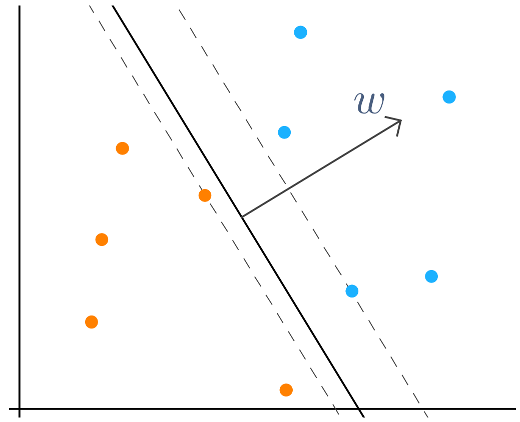


Projection of x_i onto Π: $p_i = x_i \pm \tilde{\gamma}_i \frac{w}{|w|}$, $\tilde{\gamma}_i := |p_i - x_i|$.

- $p_i \in \Pi \Rightarrow w \cdot p_i + b = 0 \Rightarrow w \cdot x_i \pm \tilde{\gamma}_i \, |w| + b = 0 \Rightarrow \tilde{\gamma}_i = \mp\left(\frac{w}{|w|} \cdot x_i + \frac{b}{|w|}\right) \geqslant 0$.
- Define the **geometric margin of x_i** as $\gamma_i := y_i\left(\frac{w}{|w|} \cdot x_i + \frac{b}{|w|}\right)$.

We want to maximize the margin to all points in the training set:

$$\underset{w,b}{\operatorname{argmax}} \min_{i=1,\dots,N} \gamma_i(x_i, w, b) = \underset{w,b}{\operatorname{argmax}} \min_{i=1,\dots,N} y_i\left(\frac{w}{|w|} \cdot x_i + \frac{b}{|w|}\right).$$



Maximal margins and two closest points

The maximal margin is attained at:

$$
\begin{aligned}
(w^\star, b^\star) &= \operatorname*{argmax}_{w,b} \min_{i=1,\ldots,N} \gamma_i(x_i, w, b) \\
&= \operatorname*{argmax}_{w,b} \left\{ \gamma \in \mathbb{R}_+ \colon \gamma_i = y_i \left(\frac{w}{|w|} \cdot x_i + \frac{b}{|w|} \right) \geqslant \gamma, i = 1, \ldots, N \right\} \\
&= \operatorname*{argmax}_{w,b} \left\{ \frac{\hat{\gamma}}{|w|} \in \mathbb{R}_+ \colon y_i \left(w \cdot x_i + b \right) \geqslant \hat{\gamma}, i = 1, \ldots, N, \hat{\gamma} = \gamma \, |w| \right\} \\
&= \operatorname*{argmax}_{w,b} \left\{ \frac{1}{|w|} \in \mathbb{R}_+ \colon y_i \left(w \cdot x_i + b \right) \geqslant 1, i = 1, \ldots, N \right\} \\
&= \operatorname*{argmin}_{w,b} \left\{ \frac{1}{2} |w|^2 \colon y_i \, \overline{w} \cdot \overline{x}_i \geqslant 1, i = 1, \ldots, N \right\}.
\end{aligned}
$$

- We have the cost function

$$f(w) = \frac{1}{2}|w|^2 \quad \text{subject to} \quad y_i\, \overline{w} \cdot \overline{x}_i \geqslant 1, \forall i.$$

- We have the cost function

$$f(w) = \frac{1}{2}|w|^2 \quad \text{subject to} \quad y_i\,\overline{w}\cdot\overline{x}_i \geqslant 1, \forall i.$$

- Optimisation (possible) using off-the-shelf Quadratic Programming routines / software.

- We have the cost function

$$f(w)=\frac{1}{2}|w|^2 \quad \text{subject to} \quad y_i\,\overline{w}\cdot\overline{x}_i \geqslant 1, \forall i.$$

- Optimisation (possible) using off-the-shelf Quadratic Programming routines / software.
- But... Infamous XOR! (Later: in the dual formulation, target function $-\infty$).

- Who's ever seen linearly separable data?

- Who's ever seen linearly separable data?
- **Relax** the constraints. Instead of $y_i\,\overline{w}\cdot\overline{x}_i \geqslant 1$, require:

$$y_i\,\overline{w}\cdot\overline{x}_i \geqslant 1-\xi_i, \quad \xi_i \geqslant 0, \quad i=1,\ldots,N, \tag{$\mathcal{P}_c$}$$

with a new cost function to minimise:

$$f(\overline{w}) = \frac{1}{2}\,|w|^2 + C\sum_{i=1}^{N}\xi_i, \quad C>0. \tag{$\mathcal{P}_f$}$$

- Who's ever seen linearly separable data?
- **Relax** the constraints. Instead of $y_i\,\overline{w}\cdot\overline{x}_i \geqslant 1$, require:

$$y_i\,\overline{w}\cdot\overline{x}_i \geqslant 1-\xi_i, \quad \xi_i \geqslant 0, \quad i=1,\ldots,N, \tag{$\mathcal{P}_c$}$$

with a new cost function to minimise:

$$f(\overline{w}) = \frac{1}{2}|w|^2 + C\sum_{i=1}^{N}\xi_i, \quad C>0. \tag{$\mathcal{P}_f$}$$

The ξ_i are called **slack variables**. One per training sample!

- Who's ever seen linearly separable data?
- **Relax** the constraints. Instead of $y_i\,\overline{w}\cdot\overline{x}_i \geqslant 1$, require:

$$y_i\,\overline{w}\cdot\overline{x}_i \geqslant 1-\xi_i, \quad \xi_i \geqslant 0, \quad i=1,\ldots,N, \qquad (\mathcal{P}_c)$$

with a new cost function to minimise:

$$f(\overline{w}) = \frac{1}{2}\,|w|^2 + C\sum_{i=1}^{N}\xi_i, \quad C>0. \qquad (\mathcal{P}_f)$$

The ξ_i are called **slack variables**. One per training sample!

- The greater ξ_i are, the more the margin constraints may be violated, but this is penalized in the cost. "$C=\infty$" means strict margins. Lower C allows for more slack.

- Who's ever seen linearly separable data?
- **Relax** the constraints. Instead of $y_i\,\overline{w}\cdot\overline{x}_i \geqslant 1$, require:

$$y_i\,\overline{w}\cdot\overline{x}_i \geqslant 1-\xi_i, \quad \xi_i \geqslant 0, \quad i=1,\ldots,N, \tag{$\mathcal{P}_c$}$$

with a new cost function to minimise:

$$f(\overline{w}) = \frac{1}{2}\,|w|^2 + C\sum_{i=1}^{N}\xi_i, \quad C>0. \tag{$\mathcal{P}_f$}$$

The ξ_i are called **slack variables**. One per training sample!

- The greater ξ_i are, the more the margin constraints may be violated, but this is penalized in the cost. "$C=\infty$" means strict margins. Lower C allows for more slack.
- Better generalisation performance.

What *are* these slack variables?

$$y_i \overline{w} \cdot \overline{x}_i \geqslant 1 - \xi_i, \quad \xi_i \geqslant 0, \quad i = 1, \dots, N,$$

- The ξ_i fulfill

What *are* these slack variables?

$$y_i \overline{w} \cdot \overline{x}_i \geqslant 1 - \xi_i, \quad \xi_i \geqslant 0, \quad i = 1, \ldots, N,$$

- The ξ_i fulfill

$$\xi_i = \begin{cases} 0 & \text{if } x_i \text{ is } \textit{on or inside} \text{ the correct margin.} \\ |y_i - \overline{w} \cdot \overline{x}_i| & \text{otherwise.} \end{cases}$$

What *are* these slack variables?

$$y_i\,\overline{w}\cdot\overline{x}_i \geqslant 1-\xi_i, \quad \xi_i \geqslant 0, \quad i=1,\ldots,N,$$

- The ξ_i fulfill

$$\xi_i = \begin{cases} 0 & \text{if } x_i \text{ is } \textit{on or inside} \text{ the correct margin.} \\ |y_i - \overline{w}\cdot\overline{x}_i| & \text{otherwise.} \end{cases}$$

- Therefore:

What *are* these slack variables?

$$y_i\,\overline{w}\cdot\overline{x}_i \geqslant 1-\xi_i, \quad \xi_i \geqslant 0, \quad i=1,\ldots,N,$$

- The ξ_i fulfill

$$\xi_i = \begin{cases} 0 & \text{if } x_i \text{ is } \textit{on or inside} \text{ the correct margin.} \\ |y_i - \overline{w}\cdot\overline{x}_i| & \text{otherwise.} \end{cases}$$

- Therefore:

$$\begin{cases} \xi_i = 1 & \text{if } x_i \text{ is on the decision boundary } (y(x_i) = \overline{w}\cdot\overline{x}_i = 0), \\ \xi_i > 1 & \text{if } x_i \text{ is misclassified.} \end{cases}$$

- We will optimize the **(unconstrained) primal problem**

$$\underset{w,b}{\operatorname{argmin}} \frac{1}{2}|w|^2 + C\sum_{i=1}^{N} \max\{0, 1 - y_i\,\overline{w}\cdot\overline{x}_i\}. \qquad (\mathcal{P})$$

- We will optimize the **(unconstrained) primal problem**

$$\underset{w,b}{\operatorname{argmin}} \frac{1}{2} |w|^2 + C \sum_{i=1}^{N} \max\{0, 1 - y_i \, \overline{w} \cdot \overline{x}_i\}. \qquad (\mathcal{P})$$

- Which we **optimise** using

- We will optimize the **(unconstrained) primal problem**

$$\underset{w,b}{\operatorname{argmin}} \frac{1}{2} |w|^2 + C \sum_{i=1}^{N} \max \{0, 1 - y_i \, \overline{w} \cdot \overline{x}_i\}. \tag{$\mathcal{P}$}$$

- Which we **optimise** using
  - **Gradient Descent**, sort of (costly).

- We will optimize the **(unconstrained) primal problem**

$$\operatorname*{argmin}_{w,b} \frac{1}{2} |w|^2 + C \sum_{i=1}^{N} \max \{0, 1 - y_i \, \overline{w} \cdot \overline{x}_i\}. \tag{$\mathcal{P}$}$$

- Which we **optimise** using
  - **Gradient Descent**, sort of (costly).
  - **Stochastic Gradient Descent** (fast).

- We will optimize the **(unconstrained) primal problem**

$$\operatorname*{argmin}_{w,b} \frac{1}{2}|w|^2 + C\sum_{i=1}^{N} \max\{0, 1 - y_i\, \overline{w} \cdot \overline{x}_i\}. \tag{$\mathcal{P}$}$$

- Which we **optimise** using
  - **Gradient Descent**, sort of (costly).
  - **Stochastic Gradient Descent** (fast).
- We will also write down a *dual problem* and optimize it using

- We will optimize the **(unconstrained) primal problem**

$$\operatorname*{argmin}_{w,b} \frac{1}{2} |w|^2 + C \sum_{i=1}^{N} \max\{0, 1 - y_i \, \overline{w} \cdot \overline{x}_i\}. \qquad (\mathcal{P})$$

- Which we **optimise** using
  - **Gradient Descent**, sort of (costly).
  - **Stochastic Gradient Descent** (fast).
- We will also write down a *dual problem* and optimize it using
  - **Sequential Minimal Optimization**.

- We will optimize the **(unconstrained) primal problem**

$$\underset{w,b}{\operatorname{argmin}} \frac{1}{2}|w|^2 + C\sum_{i=1}^{N} \max\{0, 1 - y_i\,\overline{w}\cdot\overline{x}_i\}. \qquad (\mathcal{P})$$

- Which we **optimise** using
  - **Gradient Descent**, sort of (costly).
  - **Stochastic Gradient Descent** (fast).
- We will also write down a *dual problem* and optimize it using
  - **Sequential Minimal Optimization**.
  - **Stochastic Coordinate Descent**.

- First compute the gradient of ($\mathcal{P}$)

$$\nabla_{\overline{w}} f(\overline{w}) = (w, 0) + C \sum_{i=1}^{N} \nabla_{\overline{w}} \max\{0, 1 - y_i \, \overline{w} \cdot \overline{x}_i\},$$

where:

- First compute the gradient of $(\mathcal{P})$

$$\nabla_{\overline{w}} f(\overline{w}) = (w, 0) + C \sum_{i=1}^{N} \nabla_{\overline{w}} \max\{0, 1 - y_i \overline{w} \cdot \overline{x}_i\},$$

where:

$$\nabla_{\overline{w}} \max\{\ldots\} = \begin{cases} 0 & \text{if } 1 - y_i \overline{w} \cdot \overline{x}_i < 0, \\ [0, 1] & \text{if } 1 - y_i \overline{w} \cdot \overline{x}_i = 0, \\ -y_i \overline{x}_i & \text{otherwise.} \end{cases}$$

- First compute the gradient of ($\mathcal{P}$)

$$\nabla_{\overline{w}} f(\overline{w}) = (w, 0) + C \sum_{i=1}^{N} \nabla_{\overline{w}} \max\{0, 1 - y_i \overline{w} \cdot \overline{x}_i\},$$

where:

$$\nabla_{\overline{w}} \max\{\ldots\} = \begin{cases} 0 & \text{if } 1 - y_i \overline{w} \cdot \overline{x}_i < 0, \\ [0, 1] & \text{if } 1 - y_i \overline{w} \cdot \overline{x}_i = 0, \\ -y_i \overline{x}_i & \text{otherwise.} \end{cases}$$

- First compute the gradient of ($\mathcal{P}$)

$$\nabla_{\overline{w}} f(\overline{w}) = (w, 0) + C \sum_{i=1}^{N} \nabla_{\overline{w}} \max\{0, 1 - y_i\, \overline{w} \cdot \overline{x}_i\},$$

where:

$$\nabla_{\overline{w}} \max\{\ldots\} = \begin{cases} 0 & \text{if } 1 - y_i\, \overline{w} \cdot \overline{x}_i < 0, \\ [0, 1] & \text{if } 1 - y_i\, \overline{w} \cdot \overline{x}_i = 0, \\ -y_i\, \overline{x}_i & \text{otherwise.} \end{cases}$$

- We actually have a **Subgradient Method**, with update rule

$$\overline{w}_{t+1} = ((1 - \lambda_t)\, w_t, b_t) + \lambda_t\, C \sum_{i=1}^{N} \chi_{(0,\infty)}(1 - y_i\, \overline{w}_t \cdot \overline{x}_i)\, y_i\, \overline{x}_i.$$

- SubGrad is very costly: at each step, evaluate on all N samples.

- SubGrad is very costly: at each step, evaluate on all N samples.
- Enter: **Stochastic Gradient Descent**. Convergence theory hard, black-box use "easy":

- SubGrad is very costly: at each step, evaluate on all N samples.
- Enter: **Stochastic Gradient Descent**. Convergence theory hard, black-box use "easy":

Algorithm SGD

1. Pick x_i at random.
2. Update the parameters according to

$$w_{t+1} = w_t - \lambda_t \nabla_w l(x_i, w_t, b_t) \quad \text{and} \quad b_{t+1} = b_t - \lambda_t \partial_b l(x_i, w_t, b_t).$$

   where $\lambda_t \to 0$ for $t \to \infty$ and $\sum \lambda_t^2 < \infty, \sum \lambda_t = \infty$. [because...]
3. Go to 1. until ...? (ε-acc. sol, validation)

Alternatively:

2'. *Mini-batch update:* for some $r \in \mathbb{N}$, pick $x_i, ..., x_{i_r}$ at random and do

$$w_{t+1} = w_t - \lambda_t \sum_{j=1}^{r} \nabla_w l(x_{i_j}, w_t).$$

After round 3:

After round 3:

- Optimal Margin Classifier for $K = 2$ classes.

After round 3:

- Optimal Margin Classifier for $K = 2$ classes.
- Data can be linearly separable or not.

After round 3:

- Optimal Margin Classifier for $K = 2$ classes.
- Data can be linearly separable or not.
- Two optimisation algorithms for the *primal problem*: SubGrad and SGD.

After round 3:

- Optimal Margin Classifier for $K = 2$ classes.
- Data can be linearly separable or not.
- Two optimisation algorithms for the *primal problem*: SubGrad and SGD.

Coming up:

After round 3:

- Optimal Margin Classifier for $K = 2$ classes.
- Data can be linearly separable or not.
- Two optimisation algorithms for the *primal problem*: SubGrad and SGD.

Coming up:

- Round 4: The dual formulation of the Optimal Margin Classifier.

After round 3:

- Optimal Margin Classifier for $K = 2$ classes.
- Data can be linearly separable or not.
- Two optimisation algorithms for the *primal problem*: SubGrad and SGD.

Coming up:

- Round 4: The dual formulation of the Optimal Margin Classifier.
- Two efficient algorithms for the dual: **Sequential Minimal Optimization** and **Stochastic Coordinate Ascent**.

After round 3:

- Optimal Margin Classifier for $K = 2$ classes.
- Data can be linearly separable or not.
- Two optimisation algorithms for the *primal problem*: SubGrad and SGD.

Coming up:

- Round 4: The dual formulation of the Optimal Margin Classifier.
- Two efficient algorithms for the dual: **Sequential Minimal Optimization** and **Stochastic Coordinate Ascent**.
- Finale: Handling multiple classes. Examples.

- Use Lagrange multipliers α_i, β_i to incorporate the constraints $(\mathcal{P}_c)$ into the cost $(\mathcal{P}_f)$

$$\mathcal{L}(w, b, \xi, \alpha, \beta) := \frac{1}{2}|w|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\left(y_i\,\overline{w}\cdot\overline{x}_i - 1 + \xi_i\right) - \sum_{i=1}^{N}\beta_i\,\xi_i.$$

- Use Lagrange multipliers α_i, β_i to incorporate the constraints $(\mathcal{P}_c)$ into the cost $(\mathcal{P}_f)$

$$\mathcal{L}(w, b, \xi, \alpha, \beta) := \frac{1}{2}|w|^2 + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i\,(y_i\,\overline{w}\cdot\overline{x}_i - 1 + \xi_i) - \sum_{i=1}^{N} \beta_i\,\xi_i.$$

Then solve the **dual problem**

$$\max_{\alpha,\beta;\alpha_i \geqslant 0} \underbrace{\min_{w,b,\xi} \mathcal{L}(w, b, \xi, \alpha, \beta)}_{=:-g(\alpha,\beta)}.$$

- Use Lagrange multipliers α_i, β_i to incorporate the constraints $(\mathcal{P}_c)$ into the cost $(\mathcal{P}_f)$

$$\mathcal{L}(w, b, \xi, \alpha, \beta) := \frac{1}{2}|w|^2 + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i\,(y_i\,\overline{w}\cdot\overline{x}_i - 1 + \xi_i) - \sum_{i=1}^{N} \beta_i\,\xi_i.$$

Then solve the **dual problem**

$$\max_{\alpha,\beta;\alpha_i\geqslant 0} \underbrace{\min_{w,b,\xi} \mathcal{L}(w, b, \xi, \alpha, \beta)}_{=:-g(\alpha,\beta)}.$$

- We can explicitly compute $g(\alpha,\beta) = g(\alpha)$, then switch to a $\min$ problem:

$$\min_{\alpha_i\geqslant 0} \underbrace{\frac{1}{2}\sum_{i,j=1}^{N} \alpha_i\,\alpha_j\,y_i\,\overline{x}_i\cdot\overline{x}_j - \sum_{i=1}^{N}\alpha_i,}_{g(\alpha)} \quad \text{s.t.} \sum_{i=1}^{N}\alpha_i\,y_i = 0, \text{ and } 0 \leqslant \alpha_i \leqslant C.$$

- Use Lagrange multipliers α_i, β_i to incorporate the constraints $(\mathcal{P}_c)$ into the cost $(\mathcal{P}_f)$

$$\mathcal{L}(w, b, \xi, \alpha, \beta) := \frac{1}{2}|w|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\,(y_i\,\overline{w}\cdot\overline{x}_i - 1 + \xi_i) - \sum_{i=1}^{N}\beta_i\,\xi_i.$$

Then solve the **dual problem**

$$\max_{\alpha,\beta;\alpha_i\geqslant 0}\ \underbrace{\min_{w,b,\xi}\mathcal{L}(w, b, \xi, \alpha, \beta)}_{=:-g(\alpha,\beta)}.$$

- We can explicitly compute $g(\alpha, \beta) = g(\alpha)$, then switch to a $\min$ problem:

$$\min_{\alpha_i\geqslant 0}\underbrace{\frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\,\alpha_j\,y_i\,\overline{x}_i\cdot\overline{x}_j - \sum_{i=1}^{N}\alpha_i}_{g(\alpha)}, \quad \text{s.t.} \sum_{i=1}^{N}\alpha_i\,y_i = 0, \text{ and } 0 \leqslant \alpha_i \leqslant C.$$

- Optimisation later...

- **Strong duality** $\Rightarrow$ Recover $w^\star, b^\star$ from $\alpha_i^\star$ through

$$w^\star = \sum \alpha_i^\star y_i x_i, \quad b^\star = \ldots$$

The output of the classifier is:

- **Strong duality** $\Rightarrow$ Recover $w^\star, b^\star$ from $\alpha_i^\star$ through

$$w^\star = \sum \alpha_i^\star y_i x_i, \quad b^\star = \ldots$$

The output of the classifier is:

$$w^\star \cdot x + b^\star = \sum_{i=1}^{N} \alpha_i y_i \overbrace{x_i \cdot x}^{(!)} + b^\star.$$

- **Strong duality** $\Rightarrow$ Recover $w^\star, b^\star$ from $\alpha_i^\star$ through

$$w^\star = \sum \alpha_i^\star \, y_i \, x_i, \quad b^\star = \ldots$$

The output of the classifier is:

$$w^\star \cdot x + b^\star = \sum_{i=1}^{N} \alpha_i \, y_i \, \overbrace{x_i \cdot x}^{(!)} + b^\star.$$

- What did we win? Enter Messrs. **Karush-Kuhn-Tucker**:

$$\begin{cases} \alpha_i^\star = 0 & \Leftrightarrow \quad y_i \, \overline{w}^\star \cdot \overline{x}_i > 1 \quad \Leftrightarrow \quad x_i \text{ is “away”}, \\ 0 < \alpha_i^\star < C & \Leftrightarrow \quad y_i \, \overline{w}^\star \cdot \overline{x}_i = 1 \quad \Leftrightarrow \quad x_i \text{ is on the margin}, \\ \alpha_i^\star = C & \Leftrightarrow \quad y_i \, \overline{w}^\star \cdot \overline{x}_i < 1 \quad \Leftrightarrow \quad x_i \text{ is inside the margin}. \end{cases}$$

Most samples will be away. The others (by design few) are the **support vectors**.

- But wait!

- But wait!

$$w^\star \cdot x + b^\star = \sum_{\alpha_i \neq 0} \alpha_i^\star y_i x_i \cdot x + b^\star$$

is still worse than a single product $w^\star \cdot x + b^\star$!

- But wait!

$$w^\star \cdot x + b^\star = \sum_{\alpha_i \neq 0} \alpha_i^\star \, y_i \, x_i \cdot x + b^\star$$

  is still worse than a single product $w^\star \cdot x + b^\star$!

- Substitute $\phi(\overline{x}_i)\,\phi(\overline{x}_j)$ for $\overline{x}_i \cdot \overline{x}_j$, for a [large class] of ϕs. The output will be:

- But wait!

$$w^\star \cdot x + b^\star = \sum_{\alpha_i \neq 0} \alpha_i^\star \, y_i \, x_i \cdot x + b^\star$$

is still worse than a single product $w^\star \cdot x + b^\star$!

- Substitute $\phi(\overline{x}_i)\,\phi(\overline{x}_j)$ for $\overline{x}_i \cdot \overline{x}_j$, for a [large class] of ϕs. The output will be:

$$w^\star \cdot \phi(x) + b^\star = \sum_{\alpha_i \neq 0} \alpha_i^\star \, y_i \underbrace{\phi(x_i) \cdot \phi(x)}_{=k(x_i, x)} + b^\star$$

- But wait!

$$w^\star \cdot x + b^\star = \sum_{\alpha_i \neq 0} \alpha_i^\star \, y_i \, x_i \cdot x + b^\star$$

is still worse than a single product $w^\star \cdot x + b^\star$!

- Substitute $\phi(\overline{x}_i)\,\phi(\overline{x}_j)$ for $\overline{x}_i \cdot \overline{x}_j$, for a [large class] of ϕs. The output will be:

$$w^\star \cdot \phi(x) + b^\star = \sum_{\alpha_i \neq 0} \alpha_i^\star \, y_i \underbrace{\phi(x_i) \cdot \phi(x)}_{=k(x_i, x)} + b^\star$$

- We optimise:

- But wait!

$$w^\star \cdot x + b^\star = \sum_{\alpha_i \neq 0} \alpha_i^\star \, y_i \, x_i \cdot x + b^\star$$

  is still worse than a single product $w^\star \cdot x + b^\star$!

- Substitute $\phi(\overline{x}_i)\,\phi(\overline{x}_j)$ for $\overline{x}_i \cdot \overline{x}_j$, for a [large class] of ϕs. The output will be:

$$w^\star \cdot \phi(x) + b^\star = \sum_{\alpha_i \neq 0} \alpha_i^\star \, y_i \underbrace{\phi(x_i) \cdot \phi(x)}_{=k(x_i, x)} + b^\star$$

- We optimise:

$$\min_{\alpha, r; \alpha_i \geqslant 0} \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \, \alpha_j \, y_i \underbrace{\phi(\overline{x}_i) \cdot \phi(\overline{x}_j)}_{=\overline{k}(\overline{x}_i, \overline{x}_j)} - \sum_{i=1}^{N} \alpha_i.$$

- But wait!

$$w^\star \cdot x + b^\star = \sum_{\alpha_i \neq 0} \alpha_i^\star \, y_i \, x_i \cdot x + b^\star$$

  is still worse than a single product $w^\star \cdot x + b^\star$!

- Substitute $\phi(\overline{x}_i)\,\phi(\overline{x}_j)$ for $\overline{x}_i \cdot \overline{x}_j$, for a [large class] of ϕs. The output will be:

$$w^\star \cdot \phi(x) + b^\star = \sum_{\alpha_i \neq 0} \alpha_i^\star \, y_i \underbrace{\phi(x_i) \cdot \phi(x)}_{=k(x_i,x)} + b^\star$$

- We optimise:

$$\min_{\alpha, r; \alpha_i \geqslant 0} \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \, \alpha_j \, y_i \underbrace{\phi(\overline{x}_i) \cdot \phi(\overline{x}_j)}_{=\overline{k}(\overline{x}_i, \overline{x}_j)} - \sum_{i=1}^{N} \alpha_i.$$

- This **kernel trick** “embeds” the problem in a high(er) dimensional feature space (but as a lower dimensional set, no magic).

- $\phi\colon \mathcal{X} \longrightarrow \mathcal{F}$ will typically be **costly** to compute and store.

- $\phi\colon \mathcal{X} \to \mathcal{F}$ will typically be **costly** to compute and store.

  E.g. $\phi(x_1, x_2) = \left(x_1^2, \sqrt{2}\, x_1\, x_2, x_2^2\right)$. Then

  $$w \cdot \phi(x) + b = w_1\, x_1^2 + w_2\, \sqrt{2}\, x_1\, x_2 + w_3\, x_2^2 + b.$$

  The **decision boundary** will be a conic.

- $\phi\colon \mathcal{X} \to \mathcal{F}$ will typically be **costly** to compute and store.

  E.g. $\phi(x_1, x_2) = \left(x_1^2, \sqrt{2}\, x_1\, x_2, x_2^2\right)$. Then

  $$w \cdot \phi(x) + b = w_1\, x_1^2 + w_2\, \sqrt{2}\, x_1\, x_2 + w_3\, x_2^2 + b.$$

  The **decision boundary** will be a conic.

- **If** $w = \sum_{i\in I} \alpha_i\, \phi(x_i)$ then

  $$w \cdot \phi(x) + b = \sum_{i \in I} \alpha_i \underbrace{\phi(x_i)\ \phi(x)}_{= k(x_i, x)} + b.$$

  In the example:

  $$k(x, y) = \left(x_1^2, \sqrt{2}\, x_1\, x_2, x_2^2\right) \cdot \left(y_1^2, \sqrt{2}\, y_1\, y_2, y_2^2\right) = (x \cdot y)^2.$$

- $\phi\colon \mathcal{X} \to \mathcal{F}$ will typically be **costly** to compute and store.

  E.g. $\phi(x_1, x_2) = \left(x_1^2, \sqrt{2}\, x_1\, x_2, x_2^2\right)$. Then

$$w \cdot \phi(x) + b = w_1\, x_1^2 + w_2 \sqrt{2}\, x_1\, x_2 + w_3\, x_2^2 + b.$$

  The **decision boundary** will be a conic.

- **If** $w = \sum_{i \in I} \alpha_i\, \phi(x_i)$ then

$$w \cdot \phi(x) + b = \sum_{i \in I} \alpha_i \underbrace{\phi(x_i)\ \phi(x)}_{=k(x_i, x)} + b.$$

  In the example:

$$k(x, y) = \left(x_1^2, \sqrt{2}\, x_1\, x_2, x_2^2\right) \cdot \left(y_1^2, \sqrt{2}\, y_1\, y_2, y_2^2\right) = (x \cdot y)^2.$$

- Two typical kernels:

$$k(x, y) = (x \cdot y + 1)^n, \quad k(x, y) = \mathrm{e}^{-c\,|x-y|^2}.$$

- Optimising the quadratic form $\sum_{i,j=1}^{N} \alpha_i \alpha_j y_i k(\overline{x}_i, \overline{x}_j)$ involves an $\mathcal{O}(N^2)$ matrix!

The dual problem

- Optimising the quadratic form $\sum_{i,j=1}^{N} \alpha_i \alpha_j y_i k(\overline{x}_i, \overline{x}_j)$ involves an $\mathcal{O}(N^2)$ matrix!
- **Optimisation** using **S**equential **M**inimal **O**ptimisation

- Optimising the quadratic form $\sum_{i,j=1}^{N} \alpha_i \alpha_j y_i k(\overline{x}_i, \overline{x}_j)$ involves an $\mathcal{O}(N^2)$ matrix!
- **Optimisation** using **S**equential **M**inimal **O**ptimisation
    - Optimises two variables at each step.

- Optimising the quadratic form $\sum_{i,j=1}^{N} \alpha_i \alpha_j y_i k(\overline{x}_i, \overline{x}_j)$ involves an $\mathcal{O}(N^2)$ matrix!
- **Optimisation** using **S**equential **M**inimal **O**ptimisation
  - Optimises two variables at each step.
  - Optimisation can be performed analytically.

- Optimising the quadratic form $\sum_{i,j=1}^{N} \alpha_i \alpha_j y_i k(\overline{x}_i, \overline{x}_j)$ involves an $\mathcal{O}(N^2)$ matrix!
- **Optimisation** using **S**equential **M**inimal **O**ptimisation
  - Optimises two variables at each step.
  - Optimisation can be performed analytically.
  - No matrix multiplications $\Rightarrow$ fewer precision issues.

- Optimising the quadratic form $\sum_{i,j=1}^{N} \alpha_i \alpha_j y_i k(\overline{x}_i, \overline{x}_j)$ involves an $\mathcal{O}(N^2)$ matrix!
- **Optimisation** using **S**equential **M**inimal **O**ptimisation
  - Optimises two variables at each step.
  - Optimisation can be performed analytically.
  - No matrix multiplications $\Rightarrow$ fewer precision issues.
  - No storage of $\mathcal{O}(N^2)$ matrix.

- Optimising the quadratic form $\sum_{i,j=1}^{N} \alpha_i \alpha_j y_i k(\overline{x}_i, \overline{x}_j)$ involves an $\mathcal{O}(N^2)$ matrix!
- **Optimisation** using **S**equential **M**inimal **O**ptimisation
  - Optimises two variables at each step.
  - Optimisation can be performed analytically.
  - No matrix multiplications $\Rightarrow$ fewer precision issues.
  - No storage of $\mathcal{O}(N^2)$ matrix.
- Optimisation using **S**tochastic **C**oordinate **D**escent

- Optimising the quadratic form $\sum_{i,j=1}^{N} \alpha_i \alpha_j y_i k(\overline{x}_i, \overline{x}_j)$ involves an $\mathcal{O}(N^2)$ matrix!
- **Optimisation** using **S**equential **M**inimal **O**ptimisation
  - Optimises two variables at each step.
  - Optimisation can be performed analytically.
  - No matrix multiplications $\Rightarrow$ fewer precision issues.
  - No storage of $\mathcal{O}(N^2)$ matrix.
- Optimisation using **S**tochastic **C**oordinate **D**escent
  - Optimises one variable at each step.

- Optimising the quadratic form $\sum_{i,j=1}^{N} \alpha_i \alpha_j y_i k(\overline{x}_i, \overline{x}_j)$ involves an $\mathcal{O}(N^2)$ matrix!
- **Optimisation** using **S**equential **M**inimal **O**ptimisation
  - Optimises two variables at each step.
  - Optimisation can be performed analytically.
  - No matrix multiplications $\Rightarrow$ fewer precision issues.
  - No storage of $\mathcal{O}(N^2)$ matrix.
- Optimisation using **S**tochastic **C**oordinate **D**escent
  - Optimises one variable at each step.
  - Clear stopping criterion.

Sequential Minimal Optimization

- **Idea**: pick the minimal amount of variables to optimise.

- **Idea**: pick the minimal amount of variables to optimise.

  Constraint: $\sum_{i=1}^{N} \alpha_i y_i = 0 \Rightarrow$ one variable not enough. Pick two $\Rightarrow$ 2D problem.

Sequential Minimal Optimization

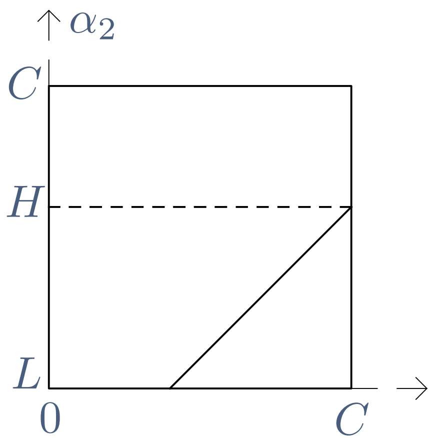
- **Idea**: pick the minimal amount of variables to optimise.

  Constraint: $\sum_{i=1}^{N} \alpha_i \, y_i = 0 \Rightarrow$ one variable not enough. Pick two $\Rightarrow$ 2D problem.
- W.l.o.g. fix α_1, α_2, then $\alpha_2 = a - s\,\alpha_1$, $a = a(\alpha_3, ...,)$

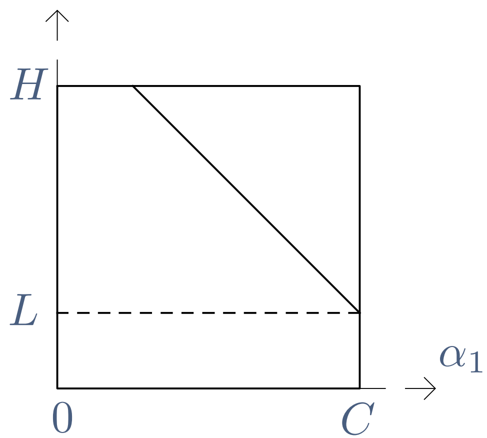
- **Idea**: pick the minimal amount of variables to optimise.

  Constraint: $\sum_{i=1}^{N} \alpha_i y_i = 0 \Rightarrow$ one variable not enough. Pick two $\Rightarrow$ 2D problem.

- W.l.o.g. fix α_1, α_2, then $\alpha_2 = a - s\,\alpha_1$, $a = a(\alpha_3, ...,)$



$y_1 \neq y_2 \Rightarrow \alpha_2 = a + \alpha_1$



$y_1 = y_2 \Rightarrow \alpha_2 = a - \alpha_1$

- Write $g(\alpha)$ as a function of α_1, differentiate, equate to 0, plug $\tilde{\alpha}_1, \tilde{\alpha}_2$ from previous step:

$$\alpha_1^\star = \tilde{\alpha}_1 - \frac{y_1 \left[(\tilde{w} \cdot x_1 - y_1) - (\tilde{w} \cdot x_2 - y_2)\right]}{\eta}.$$

- Write $g(\alpha)$ as a function of α_1, differentiate, equate to 0, plug $\tilde{\alpha}_1, \tilde{\alpha}_2$ from previous step:

$$\alpha_1^\star = \tilde{\alpha}_1 - \frac{y_1\left[(\tilde{w} \cdot x_1 - y_1) - (\tilde{w} \cdot x_2 - y_2)\right]}{\eta}.$$

- Clip it to the bounding box $[0, C]^2$: the solution is $\alpha_1^* = \max\{0, \min\{C, \alpha_1^\star\}\}$.

- Write $g(\alpha)$ as a function of α_1, differentiate, equate to 0, plug $\tilde{\alpha}_1, \tilde{\alpha}_2$ from previous step:

$$\alpha_1^\star = \tilde{\alpha}_1 - \frac{y_1 \left[(\tilde{w} \cdot x_1 - y_1) - (\tilde{w} \cdot x_2 - y_2) \right]}{\eta}.$$

- Clip it to the bounding box $[0, C]^2$: the solution is $\alpha_1^* = \max\{0, \min\{C, \alpha_1^\star\}\}$.
- Compute α_2^* from this value using that $\alpha_2^* = a - s\,\alpha_1^* = \tilde{\alpha}_2 + s\,\tilde{\alpha}_1 - s\,\alpha_1^*$:

$$\alpha_2^* = \tilde{\alpha}_2 + s\,(\tilde{\alpha}_1 - \alpha_1^*).$$

- Write $g(\alpha)$ as a function of α_1, differentiate, equate to 0, plug $\tilde{\alpha}_1, \tilde{\alpha}_2$ from previous step:

$$\alpha_1^\star = \tilde{\alpha}_1 - \frac{y_1 \left[(\tilde{w} \cdot x_1 - y_1) - (\tilde{w} \cdot x_2 - y_2) \right]}{\eta}.$$

- Clip it to the bounding box $[0, C]^2$: the solution is $\alpha_1^* = \max\{0, \min\{C, \alpha_1^\star\}\}$.
- Compute α_2^* from this value using that $\alpha_2^* = a - s\,\alpha_1^* = \tilde{\alpha}_2 + s\,\tilde{\alpha}_1 - s\,\alpha_1^*$:

$$\alpha_2^* = \tilde{\alpha}_2 + s\,(\tilde{\alpha}_1 - \alpha_1^*).$$

- **Problem**: how to choose which α_i (i.e. which indexes) to optimise?

- **Heuristics** for choosing the next best α_i, α_j to optimise:

- **Heuristics** for choosing the next best α_i, α_j to optimise:
  - Outer loop: go through all α_i violating KKT.
  - Outer loop: then, go through all *non-clipped* α_i violating KKT
    - → Until all satisfy KKT within ε (most CPU time in non-clipped samples).
  - Inner loop: choose α_j to maximise the step taken ($k(\cdot, \cdot)$ costly, so approximate).
  - Corner cases
    - → Duplicate input vectors ⇒ k semidefinite ⇒ more heuristics.
    - → More...

- **Heuristics** for choosing the next best α_i, α_j to optimise:
  - Outer loop: go through all α_i violating KKT.
  - Outer loop: then, go through all *non-clipped* α_i violating KKT
    - → Until all satisfy KKT within ε (most CPU time in non-clipped samples).
  - Inner loop: choose α_j to maximise the step taken ($k(\cdot, \cdot)$ costly, so approximate).
  - Corner cases
    - → Duplicate input vectors ⇒ k semidefinite ⇒ more heuristics.
    - → More...
- Recompute the threshold...

- **Heuristics** for choosing the next best α_i, α_j to optimise:
  - Outer loop: go through all α_i violating KKT.
  - Outer loop: then, go through all *non-clipped* α_i violating KKT
    - $\rightarrow$ Until all satisfy KKT within ε (most CPU time in non-clipped samples).
  - Inner loop: choose α_j to maximise the step taken ($k(\cdot, \cdot)$ costly, so approximate).
  - Corner cases
    - $\rightarrow$ Duplicate input vectors $\Rightarrow$ k semidefinite $\Rightarrow$ more heuristics.
    - $\rightarrow$ More...
- Recompute the threshold...
- Profit!

- Easiest approach: **One versus the rest**.

- Easiest approach: **One versus the rest**.

  Train K **binary** classifiers. Let them vote.

- Easiest approach: **One versus the rest**.

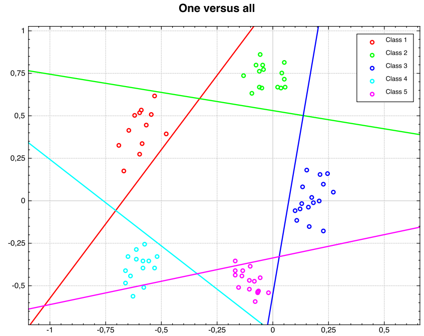
  Train K **binary** classifiers. Let them vote.

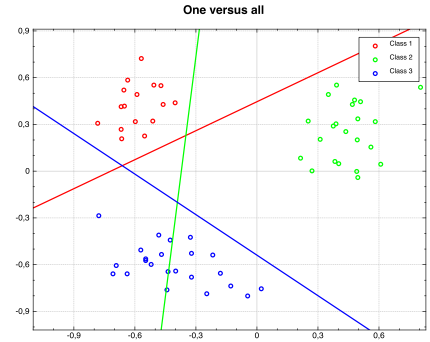
  But caution! Ambiguities and unbalanced training samples.

- Easiest approach: **One versus the rest**.

  Train K **binary** classifiers. Let them vote.

  But caution! Ambiguities and unbalanced training samples.

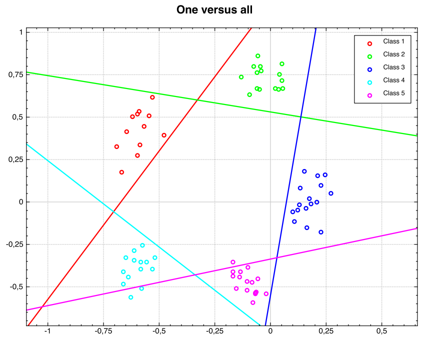


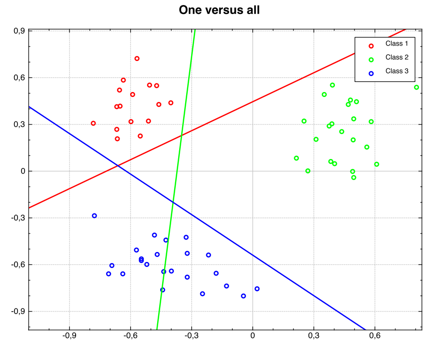


- Easiest approach: **One versus the rest**.

  Train K **binary** classifiers. Let them vote.

  But caution! Ambiguities and unbalanced training samples.



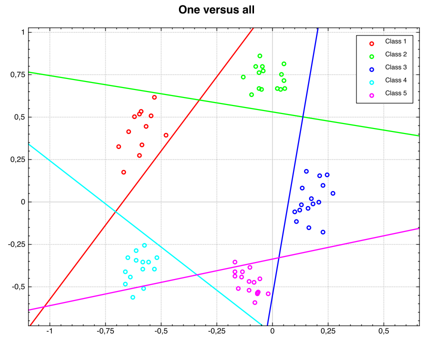


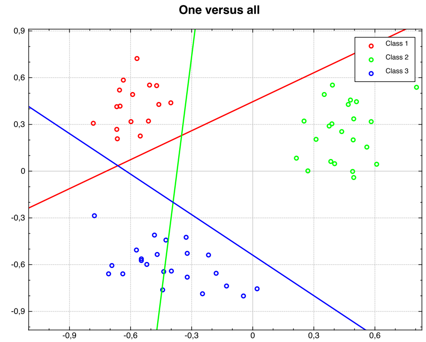
- Similar approach: **One versus one**.

- Easiest approach: **One versus the rest**.

  Train K **binary** classifiers. Let them vote.

  But caution! Ambiguities and unbalanced training samples.





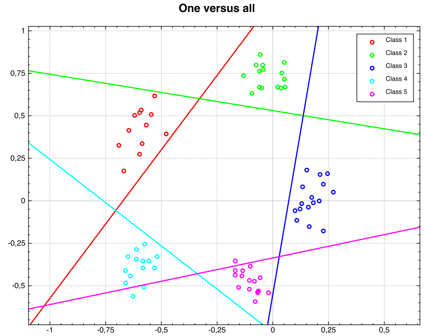
- Similar approach: **One versus one**.

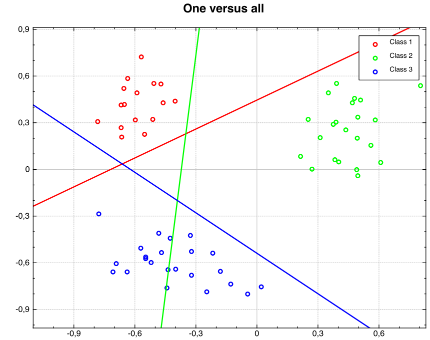
  Train $\binom{K}{2}$ classifiers. Let them vote.

- Easiest approach: **One versus the rest**.

  Train K **binary** classifiers. Let them vote.

  But caution! Ambiguities and unbalanced training samples.





- Similar approach: **One versus one**.

  Train $\binom{K}{2}$ classifiers. Let them vote.

  Again, caution.

Afterword: Multiple classes

- Why not train for all classes simultaneously? **Multiclass** classifier.

- Why not train for all classes simultaneously? **Multiclass** classifier.

  Find $W = (w_1, ..., w_K) \in \mathbb{R}^{K\times D}, b \in \mathbb{R}^K, \xi \in \mathbb{R}_+^{N\times K}$ minimising

$$C(W, b, \xi) := \frac{1}{2} W : W + C \sum_{i=1}^{N} \sum_{k \neq y_i} \xi_{ik},$$

  subject to (y_i is the correct class for sample x_i)

$$\overline{w}_{y_i} \cdot \overline{x}_i - \overline{w}_k \cdot \overline{x}_i \geqslant 2 - \xi_{ik}, \text{ and } \xi_{ik} \geqslant 0.$$

- Why not train for all classes simultaneously? **Multiclass** classifier.

  Find $W = (w_1, \dots, w_K) \in \mathbb{R}^{K \times D}, b \in \mathbb{R}^K, \xi \in \mathbb{R}_+^{N \times K}$ minimising

$$C(W, b, \xi) := \frac{1}{2} W : W + C \sum_{i=1}^{N} \sum_{k \neq y_i} \xi_{ik},$$

  subject to (y_i is the correct class for sample x_i)

$$\overline{w}_{y_i} \cdot \overline{x}_i - \overline{w}_k \cdot \overline{x}_i \geqslant 2 - \xi_{ik}, \text{ and } \xi_{ik} \geqslant 0.$$

  Equivalently, compute

$$\operatorname*{argmin}_{W, b} \frac{1}{2} W : W + C \sum_{i=1}^{N} \sum_{k \neq y_i} \max\{0, \overline{w}_{y_i} \cdot \overline{x}_i - \overline{w}_k \cdot \overline{x}_i + 2\}.$$

- SVMs for regression problems.

- SVMs for regression problems.
- Paralellization techniques.

- SVMs for regression problems.
- Paralellization techniques.
- Bayesian SVMs: the **Relevance Vector Machine**.

What next?

- SVMs for regression problems.
- Paralellization techniques.
- Bayesian SVMs: the **Relevance Vector Machine**.
- Go to the beach.

Available on request: the internet remembers everything! You'll need:

- A C++11 compiler. Any recent version of GCC or CLANG should do.
- CMake version $\geqslant 3.0.2$.
- The Qt4 libraries if you want to try the examples with a graphical interface.
- The Armadillo linear algebra library, version $\geqslant 5.200$. OpenBLAS is recommended.
- Optionally some datasets: I've used CIFAR-10 and MNIST.

References

- ⋆ Christopher M. Bishop, *Pattern Recognition and Machine Learning*, (Springer, 2006), ch. 4 and 7.
- ⋆ Andrew Ng, "Support Vector Machines", CS 229 Lecture Notes.
- ⋆ Léon Bottou and Lin Chih-Jen, "Support Vector Machine solvers", in *Large-Scale Kernel Machines*, (MIT Press, 2007), 1–27.
- ⋆ Stephen Boyd and Lieven Vanderberghe, *Convex Optimization* (Cambridge University Press, 2004), ch. 5.
- Jason Weston and Chris Watkins, "Support Vector Machines for multi-class pattern recognition", *ESANN* 99 (1999), 219–24.
- John Platt, "Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines", Technical report (Microsoft Research, April 1998).
- Léon Bottou, "Stochastic Learning", in *Advanced Lectures on Machine Learning*, ed. Olivier Bousquet and Ulrike von Luxburg, LNAI 3176 (Springer, 2004), 146–68.
- Shai Shalev-Shwartz and Nathan Srebo, "Theory and practice of Support Vector Machines optimization", in *Automatic speech and speaker recognition: large margin and kernel methods*, ed. Joseph Keshet and Samy Bengio, 2009.
- Scott Adams, "The best of Dilbert", vols. 1 and 2 (Boxtree Ltd., 2002).

End

Happy TeXmacs-ing!