Outline

1. Support vector classification
2. Optimisation in large scale supervised learning
   - Subgradient descent
   - Stochastic Gradient Descent (SGD) and PEGASOS
   - Stochastic Dual Coordinate Ascent (SDCA)
   - Practical details

Support Vector Machines

- There are countless ML methods:
  - Nearest neighbors, perceptron, bagging, boosting, AdaBoost, logistic regression, Support Vector Machines (SVMs), random forests, metric learning, ...
  - Markov random fields, Bayesian networks, Gaussian Processes, ...
  - E.g. [Schölkopf and Smola, 2002b, Hastie et al., 2001]

We will focus on **SVMs and their generalisations**.

1. Good accuracy (when applicable).
2. Clean formulation.
3. Large scale.

Structured output SVMs

Extending SVMs to handle arbitrary output spaces, particularly ones with non-trivial structure (e.g. space of poses, textual translations, sentences in a grammar, etc.).
Scoring function and classification

- The **input** $x \in \mathbb{R}^d$ is a vector to be classified.
- The **parameter** $w \in \mathbb{R}^d$ is a vector.
- The **score** is $\langle x, w \rangle$.
- The **output** $\hat{y}(x; w)$ is either $+1$ (relevant) or $-1$ (not relevant).

The “machine” part of an SVM is a simple **classification rule** that test the sign of the score:

$$\hat{y}(x; w) = \text{sign}(\langle x, w \rangle)$$

E.g. [Schölkopf and Smola, 2002a].

Feature maps

- In the SVM $\langle x, w \rangle$ the **input** $x$ is a **vectorial representation** of a datum.
- Alternatively, one can introduce a **feature map**:

  $$\Phi : \mathcal{X} \rightarrow \mathbb{R}^d, \ x \mapsto \Phi(x).$$

The classification rule becomes

$$\hat{y}(x; w) = \text{sign}(\langle \Phi(x), w \rangle).$$

With a feature map, the nature of the input $x \in \mathcal{X}$ is irrelevant (image, video, audio, ...).
The other defining aspect of an SVM is the **objective function** used to learn it.

Given **example pairs** \((x_1, y_1), \ldots, (x_n, y_n)\), the objective function is

\[
E(w) = \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{j=1}^{n} \max\{0, 1 - y_j \langle x_j, w \rangle\}.
\]

Learning the SVM amounts to minimising \(E(w)\) to obtain the optimal parameter \(w^*\).

---

**Hinge loss**

\[
E(w) = \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{j=1}^{n} \max\{0, 1 - y_j \langle x_j, w \rangle\}.
\]

**Intuition**

When the hinge loss is small, then the scoring function fits the example data well, with a “safety margin”.

---

**Learning formulation**

The other defining aspect of an SVM is the **objective function** used to learn it.

Given **example pairs** \((x_1, y_1), \ldots, (x_n, y_n)\), the objective function is

\[
E(w) = \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{j=1}^{n} \max\{0, 1 - y_j \langle x_j, w \rangle\}.
\]

Learning the SVM amounts to minimising \(E(w)\) to obtain the optimal parameter \(w^*\).

---

**An aside: support vectors**

One can show that the minimiser has a sparse decomposition

\[
w^* = \beta_1 x_1 + \cdots + \beta_n x_n
\]

where only a few of the \(\beta_i \neq 0\). The corresponding \(x_i\) are the **support vectors**.
Hinge loss

\[ E(w) = \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \langle x_i, w \rangle \} \]

**Intuition**

When the hinge loss is small, then the scoring function fits the example data well, with a “safety margin”.

**Hinge loss**

\[ L_i(w) = \max\{0, 1 - y_i \langle x_i, w \rangle \} \]

**Margin condition**

\[ L_i(w) = 0 \Rightarrow y_i \langle x_i, w \rangle \geq 1 \]
\[ \Rightarrow \text{sign}(x_i, w) = y_i. \]

**Convexity**

The hinge loss is a convex function!

**The regulariser**

\[ E(w) = \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \langle x_i, w \rangle \} \]

**Intuition**

If the regulariser \( \|w\|^2 \) is small, then the scoring function \( \langle w, x \rangle \) varies slowly.
The regulariser

\[ E(w) = \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \langle x_i, w \rangle\} \]

Intuition
If the regulariser \( \|w\|^2 \) is small, then the scoring function \( \langle w, x \rangle \) varies slowly.

To see this:
1. The regulariser is the norm of the derivative of the scoring function:
   \[ \|\nabla_x \langle x, w \rangle\|^2 = \|w\|^2. \]
2. Using the Cauchy-Schwarz inequality:
   \[ (\langle x, w \rangle - \langle x', w \rangle)^2 \leq \|x - x'\|^2 \|w\|^2. \]

The feature map

\( \langle w, \Phi(x) \rangle \) encodes a notion of similarity:

\[ (\langle \Phi(x), w \rangle - \langle \Phi(x'), w \rangle)^2 \leq \|\Phi(x) - \Phi(x')\|^2 \times \|w\|^2. \]

Intuition
Inputs with similar features receive similar scores.

Note: in all cases, points whose difference \( \Phi(x) - \Phi(x') \) is orthogonal to \( w \) receive the same score. This is a \( d - 1 \) dimensional subspace of irrelevant variations!

SVM summary

The goal is to find a scoring function \( \langle w, \Phi(x) \rangle \) that:

 Fits the data by a marging

The scoring function \( \langle w, \Phi(x) \rangle \) should fit the data by a margin:

\[
\begin{align*}
\text{if } y_i > 0 & \text{ then } \langle \Phi(x_i), w \rangle \geq 1 \\
\text{if } y_i < 0 & \text{ then } \langle \Phi(x_i), w \rangle \leq -1
\end{align*}
\]
The goal is to find a scoring function $\langle w, \Phi(x) \rangle$ that:

**Fits the data by a margin**

The scoring function $\langle w, \Phi(x) \rangle$ should fit the data by a margin:

- if $y_i > 0$ then $\langle \Phi(x_i), w \rangle \geq 1$
- if $y_i < 0$ then $\langle \Phi(x_i), w \rangle \leq -1$

**Is regular**

A small variation of the feature $\Phi(x)$ should not change the score $\langle w, \Phi(x) \rangle$ too much. The regulariser $\|w\|^2$ is a bound on this variation.

---

**Outline**

- Support vector classification
- Optimisation in large scale supervised learning
  - Subgradient descent
  - Stochastic Gradient Descent (SGD) and PEGASOS
  - Stochastic Dual Coordinate Ascent (SDCA)
  - Practical details

---

**Optimisation problems in supervised learning**

We take the *Support Vector Machines* (SVM) **primal formulation** as a typical optimisation task:

$$\min_{w \in \mathbb{R}^n} E(w), \quad E(w) = \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \langle x_i, w \rangle\}$$

Good news:
- This function is **convex**
- Hence it has a single local optimum, which is also global
Optimisation problems in supervised learning

We take the Support Vector Machines (SVM) primal formulation as a typical optimisation task:

\[
\min_{\mathbf{w} \in \mathbb{R}^d} E(\mathbf{w}), \quad E(\mathbf{w}) = \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \langle \mathbf{x}_i, \mathbf{w} \rangle\}
\]

Bad news (1): this is a large problem:

- the data dimensionality \(d\) is high
  - e.g. a bag-of-visual-words descriptor may have \(d = 10^3 - 10^4\), a Fisher Vector \(d = 10^4 - 10^5\)
- the number of training examples \(n\) is high
  - e.g. in PASCAL VOC \(n = 10^4\), in Caltech-256 \(n = 10^5\), in ImageNet \(n = 10^6\); in object detection \(n\) is as large as you like
- there exist problems that are even more complex; for instance: structured-output SVMs

Optimisation problems in supervised learning

We take the Support Vector Machines (SVM) primal formulation as a typical optimisation task:

\[
\min_{\mathbf{w} \in \mathbb{R}^d} E(\mathbf{w}), \quad E(\mathbf{w}) = \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \langle \mathbf{x}_i, \mathbf{w} \rangle\}
\]

Bad news (1): this is a large problem:

- the data dimensionality \(d\) is high
  - e.g. a bag-of-visual-words descriptor may have \(d = 10^3 - 10^4\), a Fisher Vector \(d = 10^4 - 10^5\)
- the number of training examples \(n\) is high
  - e.g. in PASCAL VOC \(n = 10^4\), in Caltech-256 \(n = 10^5\), in ImageNet \(n = 10^6\); in object detection \(n\) is as large as you like

Second order optimisation methods may be tricky to use:

- This is due to the hinge loss terms
- Second order optimisation methods may be tricky to use

Optimisation problems in supervised learning

We take the Support Vector Machines (SVM) primal formulation as a typical optimisation task:

\[
\min_{\mathbf{w} \in \mathbb{R}^d} E(\mathbf{w}), \quad E(\mathbf{w}) = \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \langle \mathbf{x}_i, \mathbf{w} \rangle\}
\]

Bad news (2): this function is non-smooth:

- This is due to the hinge loss terms
- Second order optimisation methods may be tricky to use
Quadratic programming approach

Instead of the problem \( \min_w E(w) \)

\[
E(w) = \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \langle x_i, w \rangle\}
\]

one can consider the equivalent one \( \min_{w, \xi} E(w, \xi) \)

\[
E(w, \xi) = \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} \xi_i, \quad \xi_i \geq 0, \quad \xi_i \geq 1 - y_i \langle x_i, w \rangle.
\]

This is a \textbf{linearly constrained quadratic program}.

Dual optimisation

\[\text{▶ } \text{This problem is often converted to its dual:} \]

\[
D(\alpha) = -\frac{1}{2\lambda n^2} \alpha^\top X^\top X \alpha + \frac{1}{n} \sum_{i=1}^{n} y_i \alpha_i, \quad 0 \leq y_i \alpha_i \leq 1,
\]

\[K = X^\top X, \quad X = [x_1 \ldots x_n].\]

\[\text{▶ Strong duality. In general, } D(\alpha) \leq E(w). \text{ Furthermore, the optimal solutions } w^* \text{ and } \alpha^* \text{ are "the same":}\]

\[E(w^*) = D(\alpha^*), \quad w^* = \frac{1}{\lambda} X \alpha^*.\]
Dual optimisation

- This problem is often converted to its **dual**:
  \[ D(\alpha) = -\frac{1}{2\lambda n^2} \alpha^\top X^\top X \alpha + \frac{1}{n} \sum_{i=1}^{n} y_i \alpha_i, \quad 0 \leq y_i \alpha_i \leq 1, \]
  \[ K = X^\top X, \quad X = [x_1 \ldots x_n]. \]
- **Strong duality.** In general, \( D(\alpha) \leq E(w) \). Furthermore, the optimal solutions \( w^* \) and \( \alpha^* \) are "the same":
  \[ E(w^*) = D(\alpha^*), \quad w^* = \frac{1}{\lambda} X \alpha^*. \]

Problem size

- Dimensionality = num. constraints = \( n \) (num. training examples).
- The dual is smaller than the primal if \( d \gg n \). Especially useful for very high dimensional features, even infinite dimensional ones! (does it make sense?)

Primal optimisation

We will focus first on algorithms that optimise the primal directly as currently they are faster. The prototypical objective function is then:
\[ E(w) = \frac{\lambda}{2} ||w||^2 + \frac{1}{n} \sum_{i=1}^{n} L_i(\langle x_i, w \rangle) \]
where the loss could be, for example,
\[ L_i(z) = \begin{cases} 
  \max\{0, 1 - y_i z\}, & \text{hinge}, \\
  \max\{0, 1 - y_i z\}^2, & \text{squared hinge}, \\
  (y_i - z)^2, & \text{square}, \\
  \log(1 + e^{-y_i z}), & \text{logistic}, \\
  \ldots
\end{cases} \]

Current optimisation techniques

Something else is needed for large datasets (\( n = 10^3, 10^6, 10^9 \)).

Where to look for efficiency

1. **Effective dimensionality**: may be small due to the features / data.
2. **Effective smoothness**: may be high due to the regulariser.
3. **Effective information content**: may be bounded and the data redundant.
4. **Accuracy**: numerically accurate solutions are unnecessary.

A few popular approaches:

- Stochastic Gradient Descent (SGD) [Shalev-Shwartz et al., 2007]
- Cutting plane, BMRM [Joachims, 1999, Smola et al., 2008]

Gradient descent

- Start with \( w_1 = 1; \)
- For \( t = 1, 2, \ldots \)
  1. Compute the gradient \( g_t = \nabla E(w_t) \)
  2. Update the learning rate \( \eta_t \)
  3. Update \( w_{t+1} = w_t - \eta_t g_t \)
- Stop when \( w_t \) or \( E(w_t) \) do not change anymore
Subgradient

A subgradient \( g \) of a convex function \( L(w) \) at \( w_0 \) is the parameter of a plane that passes through \((w_0, L(w_0))\) and underestimates the function:

\[
L(w) \geq \langle w - w_0, g \rangle + L(w_0)
\]
The gradient gives the best local linear approximation to a smooth function.

If the function is a convex as well as smooth function, it gives a lower bound.

The latter definition extends to convex non-smooth functions.

A subgradient \( g \) of a convex function \( L(w) \) at \( w_0 \) is the parameter of a plane that passes through \( (w_0, L(w_0)) \) and underestimates the function:

\[
L(w) \geq \langle w - w_0, g \rangle + L(w_0)
\]

at points where \( L(w) \) is non-differentiable there exist infinite sub-gradients; the set of all subgradients is denoted \( \partial L(w) \ni g \).

at points where \( L(w) \) is differentiable, the subgradient is unique and coincides with the gradient.

Subgradient descent:

- Start with \( w_1 = 1 \);
- For \( t = 1, 2, \ldots \)
  1. Compute a subgradient \( g_t \in \partial E(w_t) \)
  2. Update the learning rate \( \eta_t \)
  3. Update \( w_{t+1} = w_t - \eta_t g_t \)
- Stop when \( w_t \) or \( E(w_t) \) do not change anymore

A simple variant of gradient descent that works for non-smooth functions.

Many questions:
- Does it find the optimum?
- How do you choose the best learning rate \( \eta_t \)?
- How fast is it?
- Why?

Consider the example non-differentiable function \( g|w - w^*| \) and a fixed step \( \eta > 0 \).
Consider the example non-differentiable function $g|w - w^*|$ and a fixed step $\eta > 0$

- The gradient is $\pm g$
- Starting from $w_t = R$, it takes $T = R/\eta g$ steps to pass $w^*$
- After that, $w_t$ jumps around $w^*$ in an interval $\pm \eta g$, and $E(w_t) \leq \epsilon = \eta g^2$

In this case:

\[
T = \frac{R}{\eta g}, \quad |w_T - w^*| \leq \eta g, \quad \epsilon = \eta g^2
\]
Subgradient descent: intuition (2)

Assume now $|g| < G$ everywhere

Once converged, similar behaviour as before:

$|w_T - w^*| \leq \eta G$, $\epsilon = \eta G^2$.

Could the gradient $g$ become too small and the algorithm stop?

Key lower bound on speed:

$$(w_t - w^*)g \geq E(w_t) - E(w^*)$$

If not converged $E(w_t) - E(w^*) > \epsilon$ and the step is at least $|\eta g| \geq \eta \epsilon / |w_t - w^*|$. If $|w_t - w^*| = R$, then the minimum step is $|\eta g| \geq \eta \epsilon / R$. Hence

$$T \leq \frac{R}{\eta \epsilon / R} = \frac{R^2}{\eta} = \frac{R^2}{\eta^2 G^2}.$$ 

Subgradient descent: summary

The simple subgradient descent with fixed learning rate $\eta$:

- Converges in
  $$T(\epsilon) = \frac{R^2}{\eta \epsilon} = \frac{R^2 G^2}{\epsilon^2} = O\left(\frac{1}{\epsilon^2}\right), \quad \text{or} \quad \epsilon(T) = O\left(\frac{1}{\sqrt{T}}\right).$$
- The accuracy $\epsilon$ is controlled by the learning rate as
  $$\epsilon = \eta G^2.$$ 
- Instead, it is more common to use a diminishing rate, such as
  $$\eta_t = \frac{1}{\sqrt{t}}.$$ 

The convergence rates are the same, and optimal for this algorithm.

Subgradient descent: the ugly

- Really slow convergence:
  $$\epsilon(T) = O\left(\frac{1}{\sqrt{T}}\right).$$ 
- Each time a subgradient must be evaluated, all the training data must be visited.
Stochastic (sub)gradient descent (SGD)

Rewrite the objective as an average over points:
\[ E(w) = \frac{1}{n} \sum_{i=1}^{n} E_i(w), \quad E_i(w) = \frac{\lambda}{2} \|w\|^2 + \max\{0, 1 - y_i \langle x_i, w \rangle\} \]

- Start with \( w_1 = 1 \);
- For \( t = 1, 2, \ldots \)
  1. Pick an index \( i \) uniformly at random
  2. Compute a subgradient \( g_t \in \partial E_i(w_t) \)
  3. Update the learning rate \( \eta_t \)
  4. Update \( w_{t+1} = w_t - \eta_t g_t \)
- Stop when \( w_t \) or \( E(w_t) \) do not change anymore

The same as before, but now the subgradient is computed from a single training point each time. Iterations are therefore blazing fast (\( n \times \)).

SGD convergence

For non-smooth functions it is the same as the subgradient method with a decreasing learning rate:
\[ \epsilon(T) = O\left(\frac{1}{\sqrt{T}}\right). \]

Now, however, doing \( T \) iterations is \( n \) times faster!

PEGASOS: Faster SGD for SVMs

The PEGASOS algorithm Shalev-Shwartz et al. [2007] is a version of SGD specific to objective of the type
\[ E(w) = \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \langle x_i, w \rangle\} \]

- The key idea is to exploit the curvature of the quadratic regularizer \( \|w\|^2 \).
- Same as SGD, but use the special learning rate
  \[ \eta_t = \frac{1}{\lambda t}. \]

Intuition: if \( n = 0 \), this learning rate finds the optimum in one step.
- Convergence rate:
  \[ \epsilon(T) = O\left(\frac{\log T}{\lambda T}\right). \]
- And iterations are still blazing fast.
Learning a linear SVM using PEGASOS SGD.

Objective $E(w_t)$; suboptimality $E(w_t) - E(w^*)$ (log-plot)

- marks the point sampled at each iteration
Learning a linear SVM using PEGASOS SGD.

Objective $E(w_t)$; suboptimality $E(w_t) - E(w^*)$ (log-plot)

◦ marks the point sampled at each iteration
Learning a linear SVM using PEGASOS SGD.

Objective $E(w_t)$; suboptimality $E(w_t) - E(w^*)$ (log-plot)

○ marks the point sampled at each iteration
Learning a linear SVM using PEGASOS SGD.

Objective $E(w_t)$; suboptimality $E(w_t) - E(w^*)$ (log-plot)

○ marks the point sampled at each iteration

After several epochs (swipes across the dataset) the algorithm converges
Alternatively, one can optimise the dual formulation by maximizing:

\[ D(\alpha) = -\frac{1}{2\lambda n^2} \alpha^T X^T X \alpha + \frac{1}{n} \sum_{i=1}^{n} -L_i^*(-\alpha_i) \]

The dual is a lower bound on the primal as for all \( w \) and \( \alpha \):

\[ D(\alpha) \leq E(w) \]

The duality gap is an upper bound the sub-optimality of a point \( w \):

\[ E(w) - E(w^*) \leq E(w) - D(\alpha) \]

This can be used to diagnose convergence.

---

Stochastic Dual Coordinate Ascent (SDCA)

- Start with \( \alpha = 0, w = 0 \)
- Until \( E(w) - D(\alpha) > \epsilon \) repeat:
  1. Pick an index \( i \) (sequentially or at random)
  2. Update the dual variable \( \alpha_i \) as
     \[ \alpha_i \leftarrow \arg\max_{u} D(\alpha_1, \ldots, \alpha_{i-1}, u, \alpha_{i+1}, \ldots, \alpha_n) \]
  3. Update the primal parameter vector \( w \):
     \[ w \leftarrow \frac{1}{\lambda n} X \alpha \]
Stochastic Dual Coordinate Ascent (SDCA)

- Start with $\alpha = 0, \mathbf{w} = 0$
- Until $E(\mathbf{w}) - D(\alpha) > \epsilon$ repeat:
  1. Pick an index $i$ (sequentially or at random)
  2. Update the dual variable $\alpha_i$ as
     $$\alpha_i \leftarrow \arg\max_{\alpha_i} D(\alpha_1, \ldots, \alpha_{i-1}, u, \alpha_{i+1}, \ldots, \alpha_n)$$
  3. Update the primal parameter $\mathbf{w}$:
     $$\mathbf{w} \leftarrow \frac{1}{\lambda n} \mathbf{X}\alpha$$

The algorithm has a similar structure to SGD, but updates one dual variable (coordinate) at a time.

The dual variable is maximised exactly, given that the others are fixed.

Then, the primal parameter $\mathbf{w}$ is updated too and $E(\mathbf{w})$ is computed.

Hsieh et al. [2008] and Shalev-Shwartz and Zhang [2012]

SDCA: coordinate maximization

To update $\alpha_i$, look at how the dual objective changes:

$$D(\alpha_i \leftarrow \alpha_i + \delta | \alpha) = -\frac{1}{2\lambda n^2} \langle \mathbf{x}_i, \mathbf{x}_i \rangle \delta^2 - \frac{1}{\lambda n^2} \mathbf{x}_i^\top \mathbf{x}_i \delta - \frac{1}{n} L_i^* (\alpha_i - \delta) + \text{const.}$$

Recall that $\mathbf{w} = \mathbf{X}\alpha / \lambda n$. Hence, the update is:

$$\alpha_i \leftarrow \alpha_i + \delta^*, \quad \delta^* = \arg\max_{\delta} -\frac{1}{2\lambda n} \langle \mathbf{x}_i, \mathbf{x}_i \rangle \delta^2 - \langle \mathbf{x}_i, \mathbf{w} \rangle \delta - L_i^* (\alpha_i - \delta)$$

This is a simple 1D optimisation, and solvable in closed form form most losses.

SDCA: coordinate maximization

To update $\alpha_i$, look at how the dual objective changes:

$$D(\alpha_i \leftarrow \alpha_i + \delta | \alpha) = -\frac{1}{2\lambda n^2} \langle \mathbf{x}_i, \mathbf{x}_i \rangle \delta^2 - \frac{1}{\lambda n^2} \mathbf{x}_i^\top \mathbf{x}_i \delta - \frac{1}{n} L_i^* (\alpha_i - \delta) + \text{const.}$$

Recall that $\mathbf{w} = \mathbf{X}\alpha / \lambda n$. Hence, the update is:

$$\alpha_i \leftarrow \alpha_i + \delta^*, \quad \delta^* = \arg\max_{\delta} -\frac{1}{2\lambda n} \langle \mathbf{x}_i, \mathbf{x}_i \rangle \delta^2 - \langle \mathbf{x}_i, \mathbf{w} \rangle \delta - L_i^* (\alpha_i - \delta)$$

This is a simple 1D optimisation, and solvable in closed form form most losses.
SDCA: coordinate maximization

- To update $\alpha_i$, look at how the dual objective changes:
  
  \[
  D(\alpha_i \leftarrow \alpha_i + \delta | \alpha) = -\frac{1}{2\lambda n^2} (\mathbf{x}_i, \mathbf{x}_i) \delta^2 - \frac{1}{\lambda n^2} \alpha^T X \delta - \frac{1}{n} L_i^* (-\alpha_i - \delta) \]

- Recall that $\mathbf{w} = X \alpha / \lambda n$. Hence, the update is:
  
  \[
  \alpha_i \leftarrow \alpha_i + \delta^*, \quad \delta^* = \arg \max_\delta -\frac{1}{2\lambda n} (\mathbf{x}_i, \mathbf{x}_i) \delta^2 - (\mathbf{x}_i, \mathbf{w}) \delta - L_i^* (-\alpha_i - \delta)
  \]

- This is a simple 1D optimisation, and solvable in closed form.

- Once $\alpha_i$ is updated, $\mathbf{w}$ can be updated cheaply too:
  
  \[
  \mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}_i \delta^* / \lambda n
  \]

Learning a linear SVM using SDCA.

Objective $E(\mathbf{w}_i)$ and lower bound $D(\alpha)$; suboptimality $E(\mathbf{w}_i) - E(\mathbf{w}^*) \mathbf{a}$ and duality gap $E(\mathbf{w}) - D(\alpha)$ (log-plot)

- marks the point sampled at each iteration
Learning a linear SVM using SDCA.

Objective $E(w_t)$ and lower bound $D(\alpha)$;
suboptimality $E(w_t) - E(w^*)a$ and duality gap $E(w) - D(\alpha)$ (log-plot)

○ marks the point sampled at each iteration
Learning a linear SVM using SDCA.

Objective $E(w_t)$ and lower bound $D(\alpha)$;
suboptimality $E(w_t) - E(w^*)a$ and duality gap $E(w) - D(\alpha)$ (log-plot)

◦ marks the point sampled at each iteration
Learning a linear SVM using SDCA.

Objective $E(w_t)$ and lower bound $D(\alpha)$; suboptimality $E(w_t) - E(w^*)\alpha$ and duality gap $E(w) - D(\alpha)$ (log-plot)

• marks the point sampled at each iteration
Learning a linear SVM using SDCA.

After several epochs (swipes across the dataset) the algorithm converges

The bias term

- So far, SVMs have been assumed to be unbiased. However, the bias term is of practical importance:
  \[ \langle w, x \rangle + b \]

- The bias term can be incorporated into the feature vector:
  \[ \langle \begin{bmatrix} w \\ w_b \end{bmatrix}, \begin{bmatrix} x \\ B \end{bmatrix} \rangle, \quad B > 0, \quad w_b = \frac{b}{B} \]

- \( B \) is the bias multiplier. It affects the regulariser:
  \[ \| \begin{bmatrix} w \\ w_b \end{bmatrix} \|^2 = \| w \|^2 + \left( \frac{b}{B} \right)^2 \]

- If \( B \gg \| w^* \|/\| b^* \| \), the perturbation of the SVM is small
- However, \( B \gg \| x_i \| \) increases \( G \), making convergence slow
- Reasonable setting: \( B \approx 5 \| x_i \| \) (esp. after centering and normalising the data).
Bibliography I


