Structure and
Support Vector Machines

Outline

- SVMs for structured outputs
  - Declarative view
  - Procedural view

Notation for Linear Models

- Training data: {(x1, y1), (x2, y2), …, (xN, yN)}
- Testing data: {(xN+1, yN+1), … (xN+N’, yN+N’)}
- Feature function: **g**
- Weights: **w**
- Decoding:

$$\text{decode}(\mathbf{w}, \boldsymbol{x}) = \arg\max_{\boldsymbol{y}} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y})$$

- Learning:

$$\text{learn}\left(\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N}\right) = \arg\max_{\mathbf{w}} \Phi\left(\mathbf{w}, \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N}\right)$$

- Evaluation:

$$\frac{1}{N'} \sum_{i=1}^{N'} \text{cost}\left(\text{decode}\left(\text{learn}\left(\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N}\right), \boldsymbol{x}_{N+i}\right), \boldsymbol{y}_{N+i}\right)$$

Empirical Risk Minimization

- A unifying framework for many learning algorithms.

$$\begin{aligned}
\text{learn}\left(\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N\right) &= \arg\max_{\mathbf{w}} \Phi\left(\mathbf{w}, \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N\right) \\
&= \arg\min_{\mathbf{w}} \underbrace{\frac{1}{N}\sum_{i=1}^{N} L(\mathbf{w}, \boldsymbol{x}_i, \boldsymbol{y}_i)}_{\approx\, \mathbb{E}[L(\mathbf{w}, \boldsymbol{X}, \boldsymbol{Y})]} + R(\mathbf{w})
\end{aligned}$$

- Many options for the loss function L and the regularization function R.

Loss Functions You Know

Name	Expression of $L(\mathbf{w}, x, y)$
Log loss (joint)	$-\log p(\boldsymbol{x}, \boldsymbol{y} \mid \mathbf{w})$
Log loss (conditional)	$-\log p(\boldsymbol{y} \mid \boldsymbol{x}, \mathbf{w})$
Zero-one loss	$\mathbf{1}\{\text{decode}(\mathbf{w}, \boldsymbol{x}) \neq \boldsymbol{y}\}$
Expected zero-one loss	$1 - p(\boldsymbol{y} \mid \boldsymbol{x}, \mathbf{w})$

Loss Functions You Know

Name	Expression of $L(\mathbf{w}, x, y)$
Log loss (joint)	$-\log p(\boldsymbol{x}, \boldsymbol{y} \mid \mathbf{w})$
Log loss (conditional)	$-\log p(\boldsymbol{y} \mid \boldsymbol{x}, \mathbf{w})$
Cost	$\mathrm{cost}(\mathrm{decode}(\mathbf{w}, \boldsymbol{x}), \boldsymbol{y})$
Expected cost, a.k.a. “risk”	$\mathbb{E}_{p(\boldsymbol{Y} \mid \boldsymbol{x}, \mathbf{w})}[\mathrm{cost}(\boldsymbol{Y}, \boldsymbol{y})]$

Structured Perceptron

- Described as an online algorithm.
- On each iteration, take one example, and update the weights according to:

$$\mathbf{w} \quad \leftarrow \quad \mathbf{w} + \alpha \left(\mathbf{g}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{g}(\boldsymbol{x}_t, \mathrm{decode}(\mathbf{w}, \boldsymbol{x}_t)) \right)$$

Loss Functions You Know

Name	Expression of $L(\mathbf{w}, x, y)$
Log loss (joint)	$-\log p(\boldsymbol{x}, \boldsymbol{y} \mid \mathbf{w})$
Log loss (conditional)	$-\log p(\boldsymbol{y} \mid \boldsymbol{x}, \mathbf{w})$
Cost	$\mathrm{cost}(\mathrm{decode}(\mathbf{w}, \boldsymbol{x}), \boldsymbol{y})$
Expected cost, a.k.a. “risk”	$\mathbb{E}_{p(\boldsymbol{Y} \mid \boldsymbol{x}, \mathbf{w})}[\mathrm{cost}(\boldsymbol{Y}, \boldsymbol{y})]$
Perceptron loss	$-\mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) + \max_{\boldsymbol{y}'} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}')$

The Ideal Loss Function

- Convex
- Continuous
- Cost-aware

Cost and Margin

- The “margin” is an important concept when we take the linear models point of view.
  - A “large margin” means that the correct output is well-separated from the incorrect outputs.
- Neither log loss nor “perceptron loss” takes into account the *cost* function, though.
  - In other words, some incorrect outputs are worse than others.

Multiclass SVM (Crammer and Singer, 2001)

$$\begin{aligned}
&\max_{\mathbf{w}} \gamma \\
&\quad \text{s.t.} \quad \|\mathbf{w}\| \leq 1 \\
&\qquad\qquad \forall i, \forall \boldsymbol{y},\ \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}_i, \boldsymbol{y}) \geq \begin{cases} \gamma & \text{if } \boldsymbol{y} \neq \boldsymbol{y_i} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

- The above can be understood as a 0-1 cost; let's generalize a bit:

$$\begin{aligned}
&\max_{\mathbf{w}} \gamma \\
&\quad \text{s.t.} \quad \|\mathbf{w}\| \leq 1 \\
&\qquad\qquad \forall i, \forall \boldsymbol{y},\ \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}_i, \boldsymbol{y}) \geq \gamma \text{cost}(\boldsymbol{y}, \boldsymbol{y}_i)
\end{aligned}$$

Max-Margin Markov Networks

- Starting point: multiclass SVM (Crammer and Singer, 2001)

$$
\begin{aligned}
&\max_{\mathbf{w}} \gamma \\
&\quad \text{s.t.} \quad \|\mathbf{w}\| \leq 1 \\
&\qquad\quad \forall i, \forall \boldsymbol{y},\ \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}_i, \boldsymbol{y}) \geq \gamma \mathrm{cost}(\boldsymbol{y}, \boldsymbol{y}_i)
\end{aligned}
$$

Max-Margin Markov Networks

- Standard transformation to get rid of explicit mention of γ, plus slack variables in case the constraints cannot be met:

$$\min_{\mathbf{w}} \frac{C}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{N} \xi_i$$
$$\text{s.t.} \quad \forall i, \forall \boldsymbol{y},\ \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}_i, \boldsymbol{y}) \geq \mathrm{cost}(\boldsymbol{y}, \boldsymbol{y}_i) - \xi_i$$

- Notice:

$$\begin{aligned} \forall i, \forall \boldsymbol{y},\ \xi_i &\geq -\mathbf{w}^\top \mathbf{g}(\boldsymbol{x}_i, \boldsymbol{y}_i) + \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}_i, \boldsymbol{y}) + \mathrm{cost}(\boldsymbol{y}, \boldsymbol{y}_i) \\ \forall i,\ \xi_i &\geq \max_{\boldsymbol{y}} -\mathbf{w}^\top \mathbf{g}(\boldsymbol{x}_i, \boldsymbol{y}_i) + \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}_i, \boldsymbol{y}) + \mathrm{cost}(\boldsymbol{y}, \boldsymbol{y}_i) \end{aligned}$$

Max-Margin Markov Networks

- Having solved for the slack variables, we can plug in; we now have an unconstrained problem:

$$\min_{\mathbf{w}} \frac{C}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{N} -\mathbf{w}^\top \mathbf{g}(\boldsymbol{x}_i, \boldsymbol{y}_i) + \max_{\boldsymbol{y}} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}_i, \boldsymbol{y}) + \mathrm{cost}(\boldsymbol{y}, \boldsymbol{y}_i)$$

- Ratliff, Bagnell, and Zinkevich (2007): subgradient descent (or stochastic version) – much, much simpler approach to optimizing this function.
  - And more perceptron-like!

$$-g_j(\boldsymbol{x}, \boldsymbol{y}) + g_j(\boldsymbol{x}, \mathrm{cost\_augmented\_decode}(\mathbf{w}, \boldsymbol{x}))$$

Structured Hinge Loss

- Small change to the perceptron loss:

$$L(\mathbf{w}, \boldsymbol{x}, \boldsymbol{y}) \quad = \quad -\mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) + \max_{\boldsymbol{y}'} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}') + \mathrm{cost}(\boldsymbol{y}', \boldsymbol{y})$$

- Resulting subgradient:

$$-g_j(\boldsymbol{x}, \boldsymbol{y}) + g_j(\boldsymbol{x}, \mathrm{cost\_augmented\_decode}(\mathbf{w}, \boldsymbol{x}))$$

  – Rather than merely decoding, find a candidate y' that is both high-scoring and *dangerous*.

Cost-Augmented Decoding

$$\begin{aligned}\mathrm{decode}(\mathbf{w}, \boldsymbol{x}) &= \arg\max_{\boldsymbol{y}'} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}') \\ \mathrm{cost\_augmented\_decode}(\mathbf{w}, \boldsymbol{x}, \boldsymbol{y}) &= \arg\max_{\boldsymbol{y}'} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}') + \mathrm{cost}(\boldsymbol{y}', \boldsymbol{y})\end{aligned}$$

- Efficient decoding is possible when the features factor locally:

$$\mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) = \sum_p \mathbf{f}(\boldsymbol{x}, \mathrm{part}_p(\boldsymbol{y}))$$

- Efficient cost-augmented decoding requires that the cost function break into parts the same way:

$$\mathrm{cost}(\boldsymbol{y}', \boldsymbol{y}) = \sum_p \mathrm{local\_cost}(\mathrm{part}_p(\boldsymbol{y}'), \boldsymbol{y})$$

An Exercise

- If the features are such that we can use the Viterbi algorithm for decoding, what are some cost functions we could inside an efficient cost-augmented decoding algorithm that’s a very small change to Viterbi?

Structured Hinge

- Three different lines of work all arrived at this idea, or something very close.
  - Max-margin Markov networks (Taskar, Guestrin, and Koller, 2003)
  - Structural support vector machines (Tsochantaridis, Joachims, Hoffman, and Altun, 2005)
  - Online passive-aggressive algorithms (Crammer, Keshet, Dekel, Shalev-Shwartz, and Singer, 2006)
- Important developments in optimization techniques since then!
  - I’ll highlight what I think it’s most useful to know.

Max-Margin Markov Networks

- Taskar et al. actually work through a *dual* version of the problem.
  - Primal and dual are both QPs; exponentially many constraints or variables, respectively.
- Key trick: *factored dual*.
  - Enables kernelized factors in the MN.
  - Actual algorithm is sequential minimal optimization (SMO) for SVMs, a coordinate ascent method (Platt, 1999).
- The paper includes a generalization bound that is argued to improve over the Collins perceptron.
- Experiments: handwriting recognition, text classification for hyperlinked documents.

I'm Taking Liberties

- The M3N view of the world really thinks about outputs as configurations in a Markov network.
- They assume y corresponds to a set of random variables, each of which gets a label in a finite set.
- Their cost function is Hamming cost: "how many r.v.s do I predict incorrectly?"
  - This is convenient and makes sense for their applications. But it's not as general as it could be.

Structural SVM

- Tsochantaridis et al. (2005) – extends their 2004 paper.
- Slightly different version of the loss function:

$$\min_{\mathbf{w}} \frac{C}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{N} \xi_i$$

$$\text{s.t.} \quad \forall i, \forall \boldsymbol{y},\ \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}_i, \boldsymbol{y}) \geq +1 - \frac{\xi_i}{\text{cost}(\boldsymbol{y}, \boldsymbol{y}_i)}$$

  – Alternative version of cost-augmented decoding ("slack rescaling" as opposed to Taskar et al.'s "margin rescaling")

Optimization Algorithms for SSVMs

- Taskar et al. (2003): SMO based on factored dual
- Bartlett et al. (2004) and Collins et al. (2008): exponentiated gradient
- Tsochantaridis et al. (2005): cutting plane (based on dual)

Cutting Plane

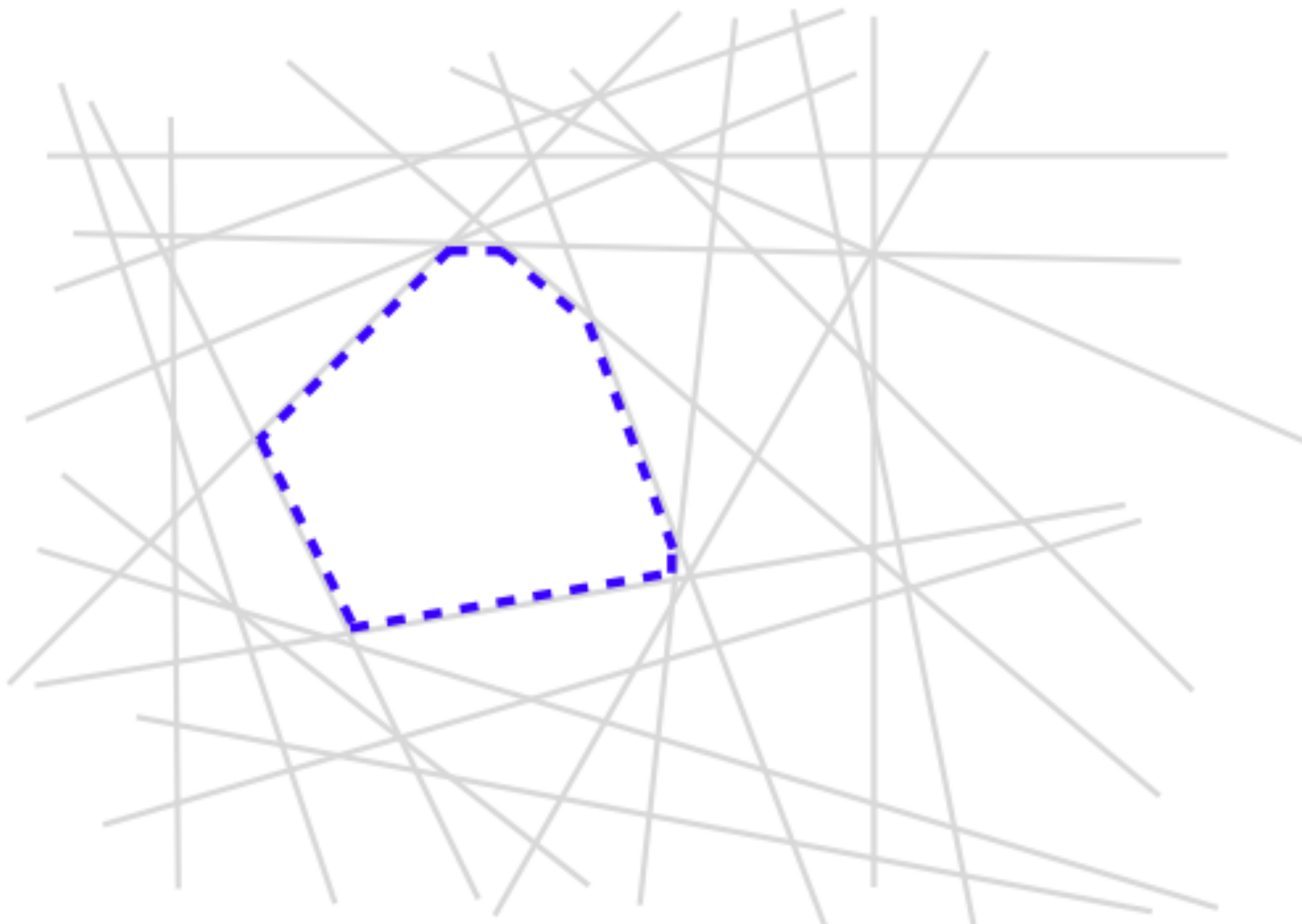
- There are exponentially many constraints!

$$\min_{\mathbf{w}} \frac{C}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{N} \xi_i$$

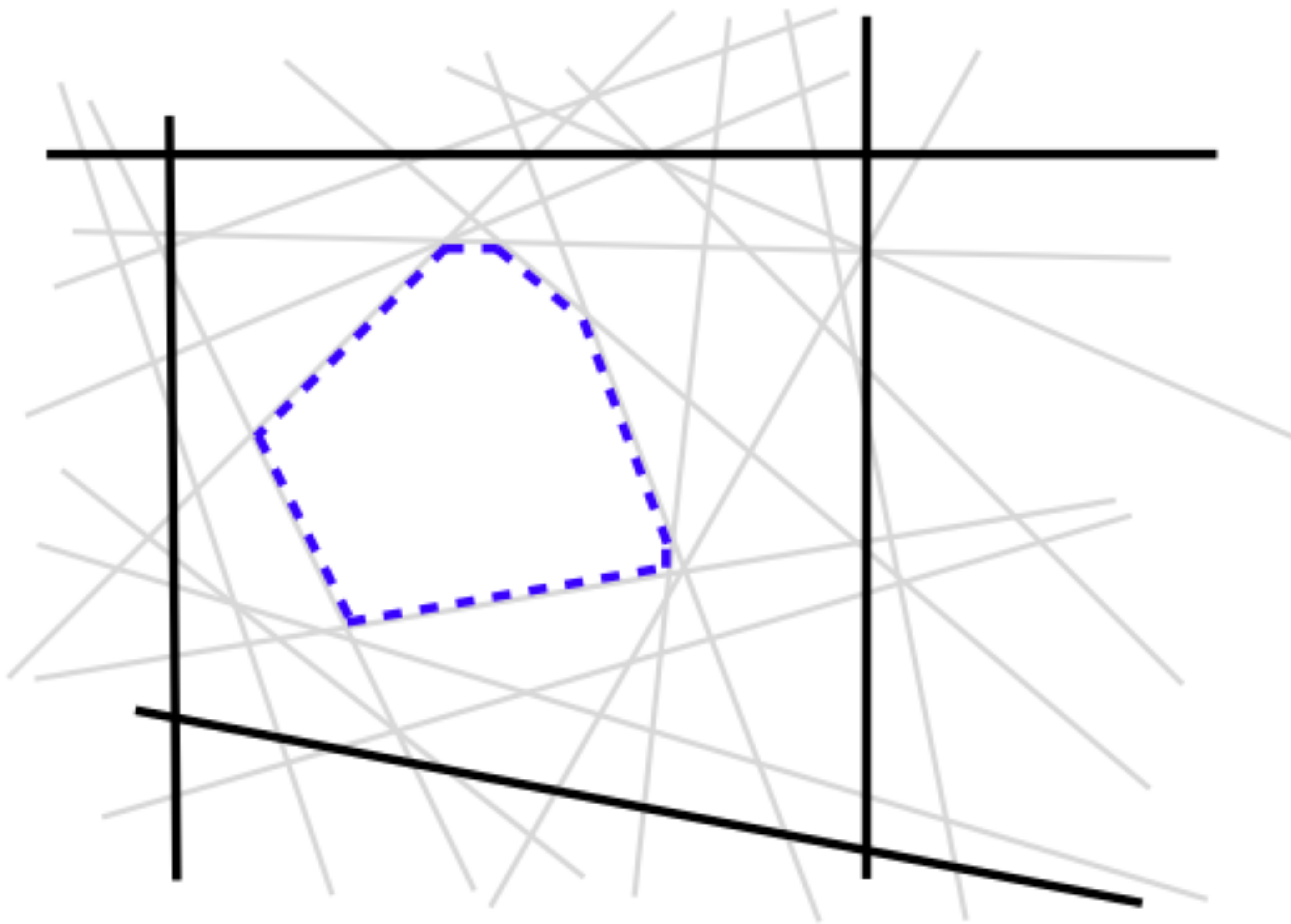
$$\text{s.t.} \quad \forall i, \forall \boldsymbol{y}, \; \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}_i, \boldsymbol{y}) \geq \text{cost}(\boldsymbol{y}, \boldsymbol{y}_i) - \xi_i$$

- Instead of enumerating them all, let's dynamically add constraints as needed
- Iterate: solve a relaxation with a subset of constraints, then add most violated constraint

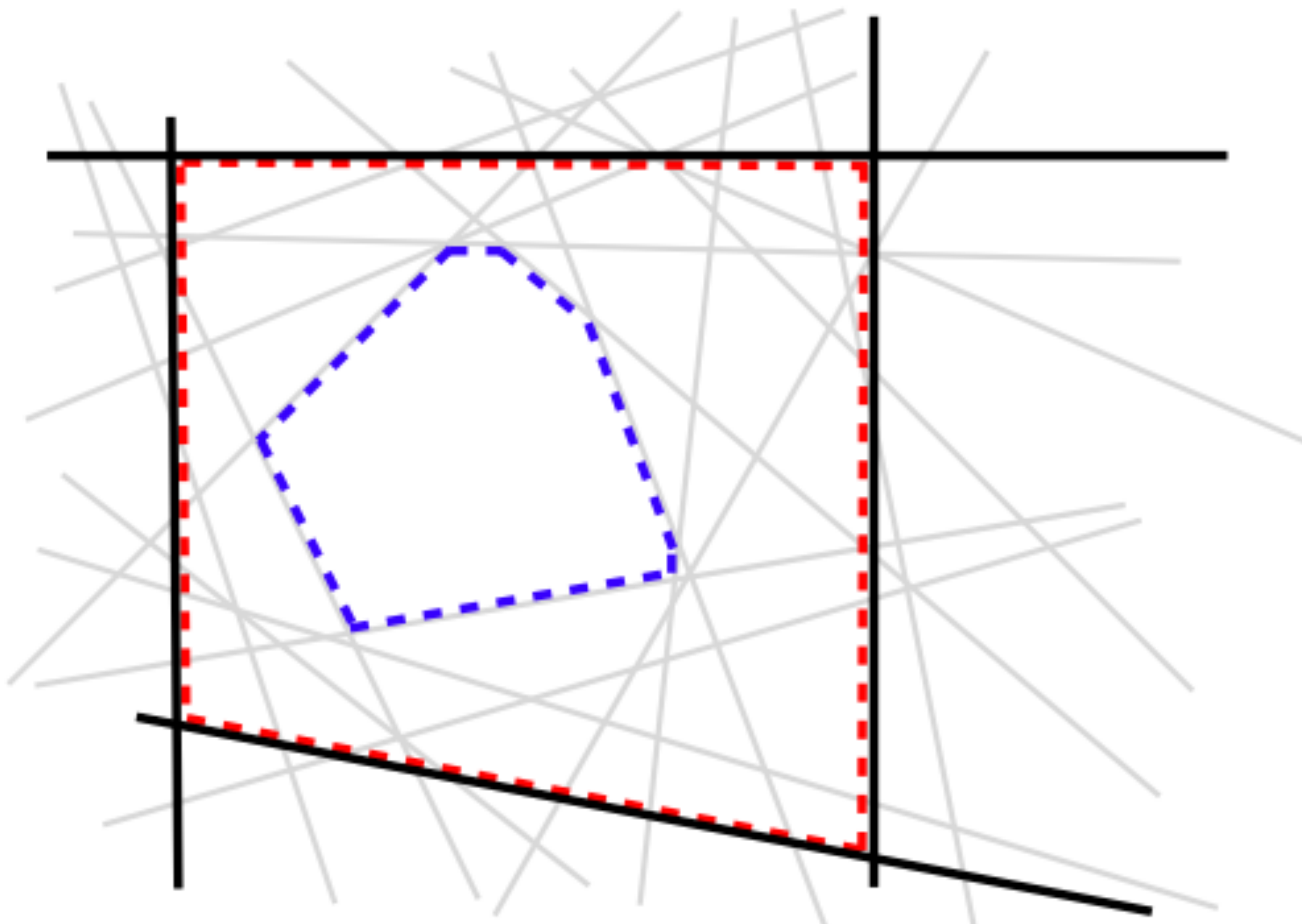
Cutting Plane



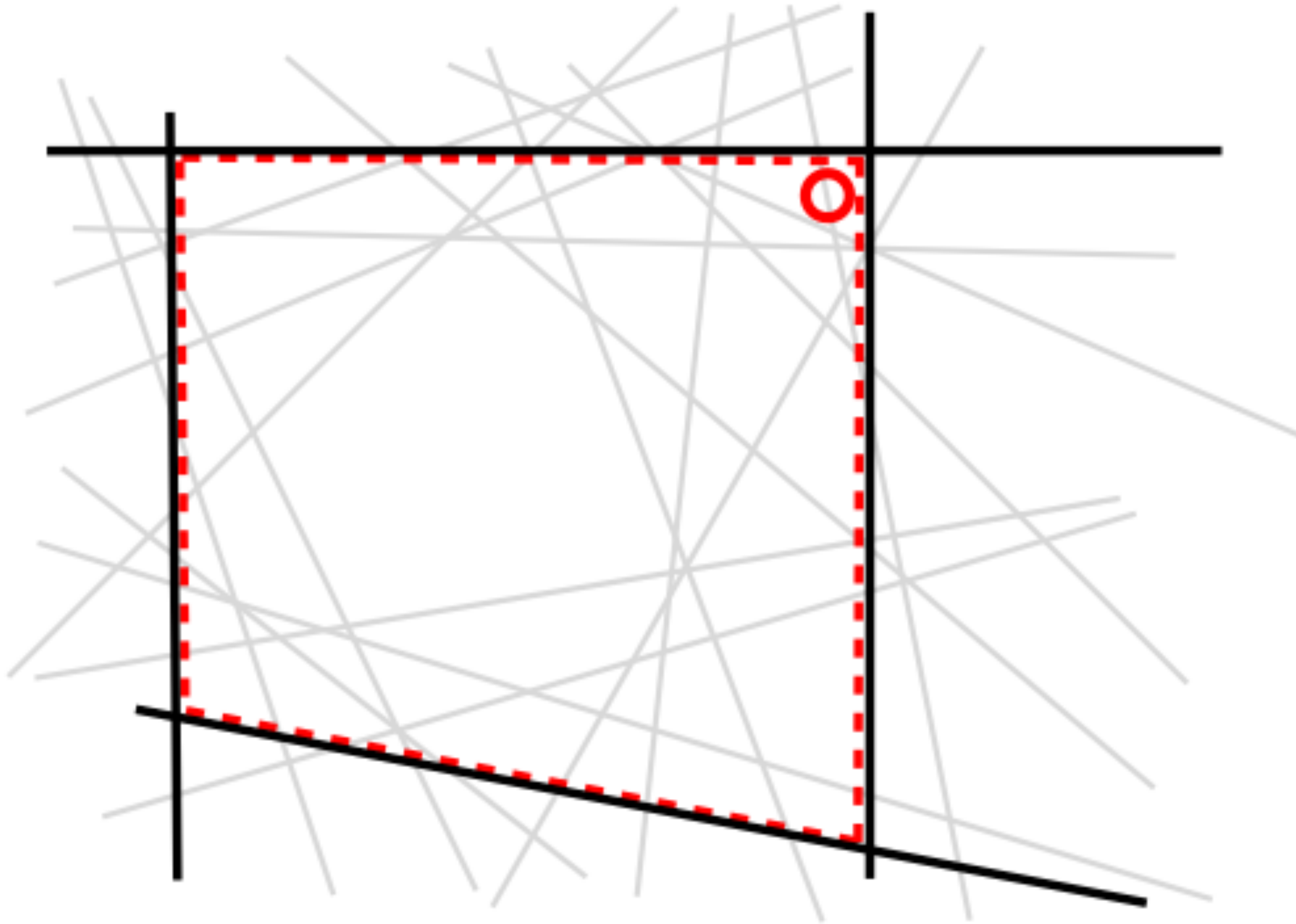
Cutting Plane



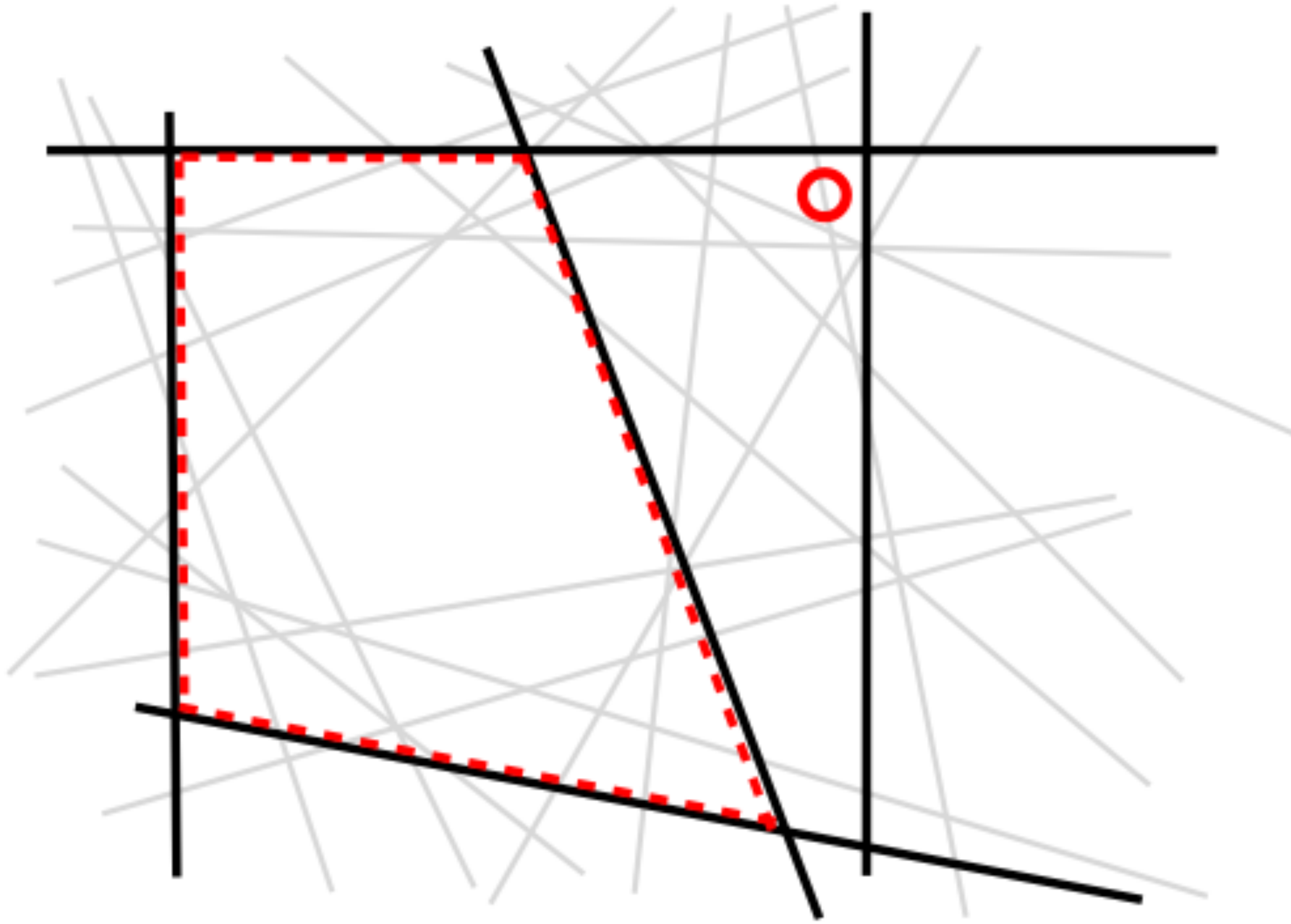
Cutting Plane



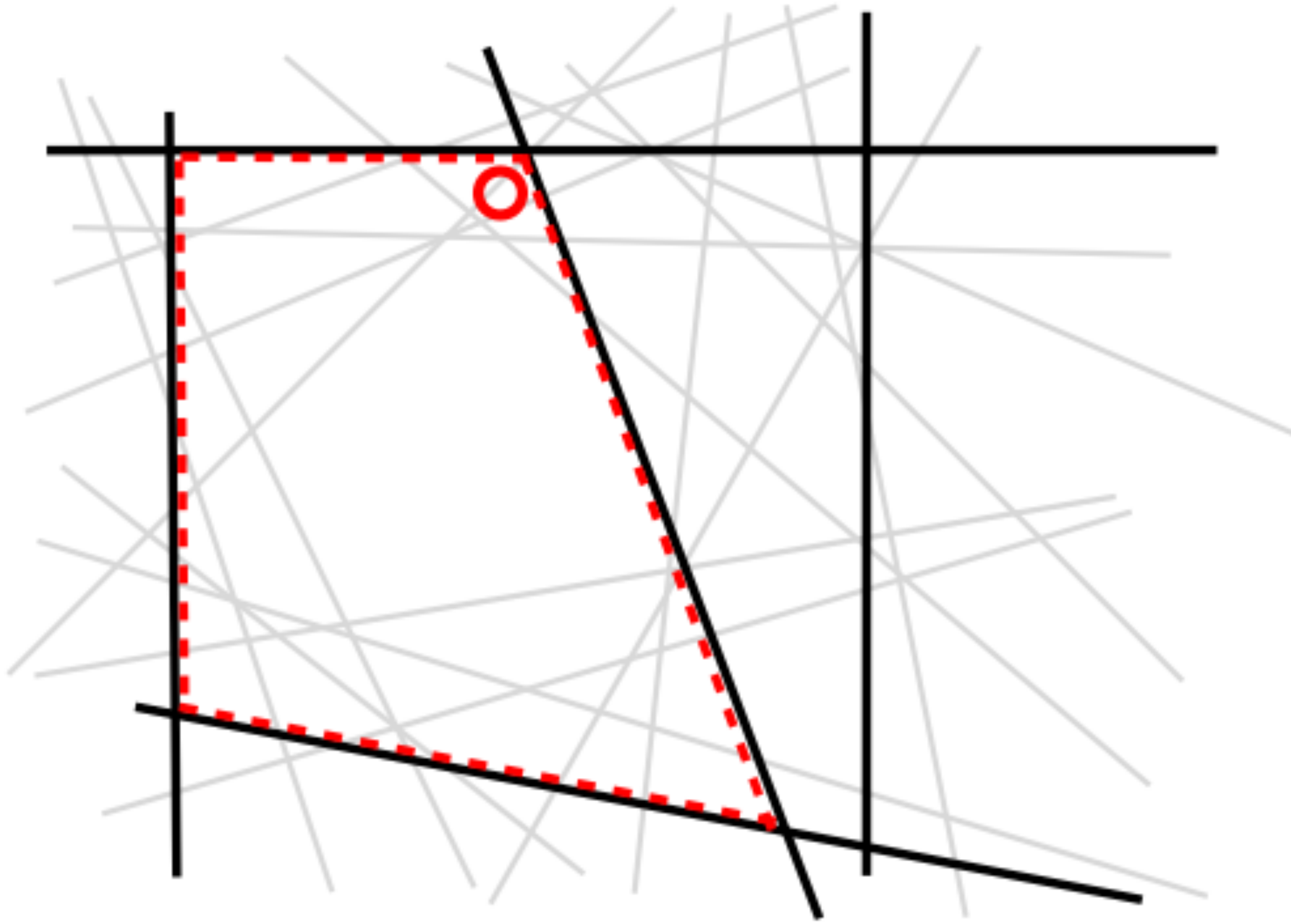
Cutting Plane



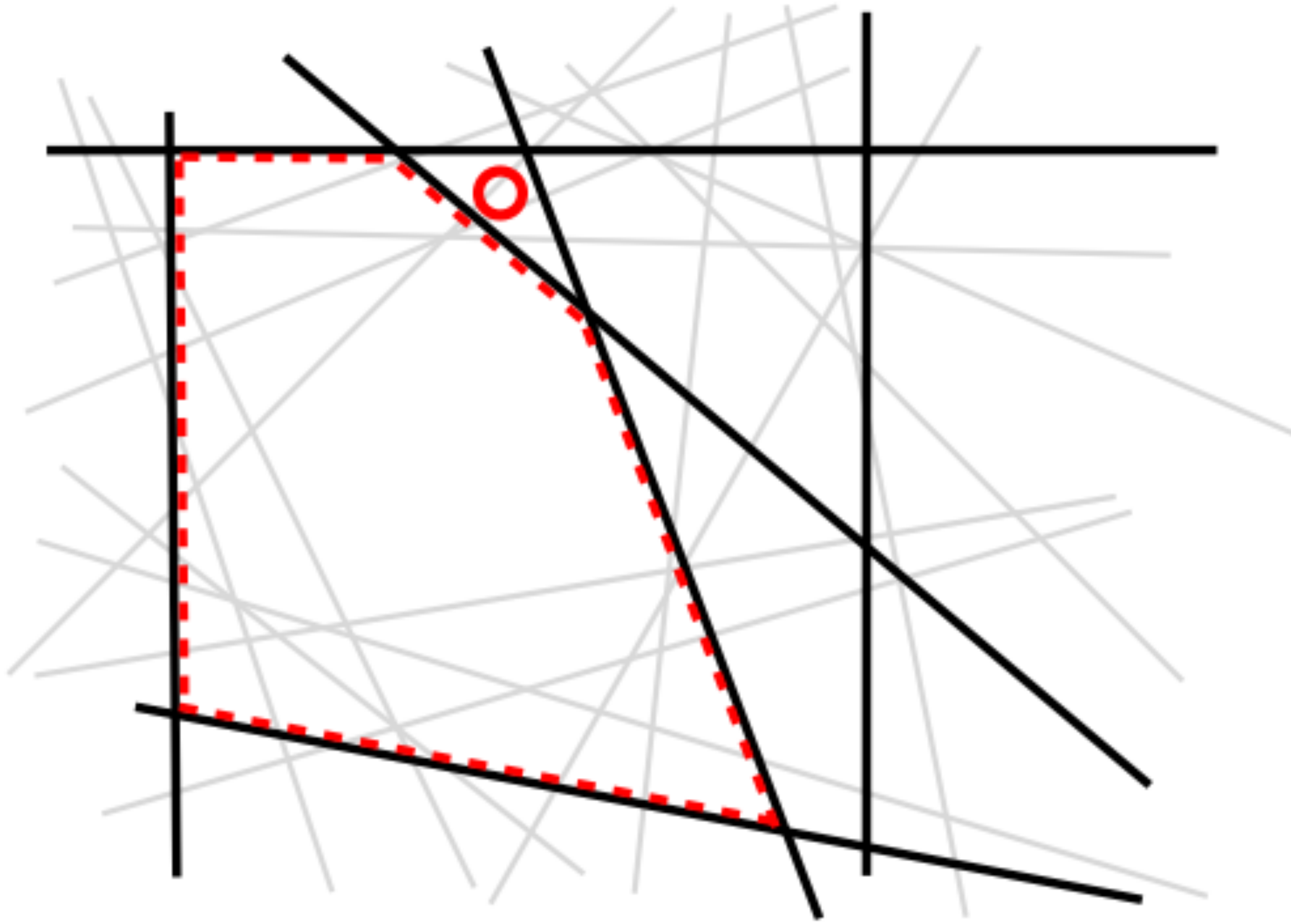
Cutting Plane



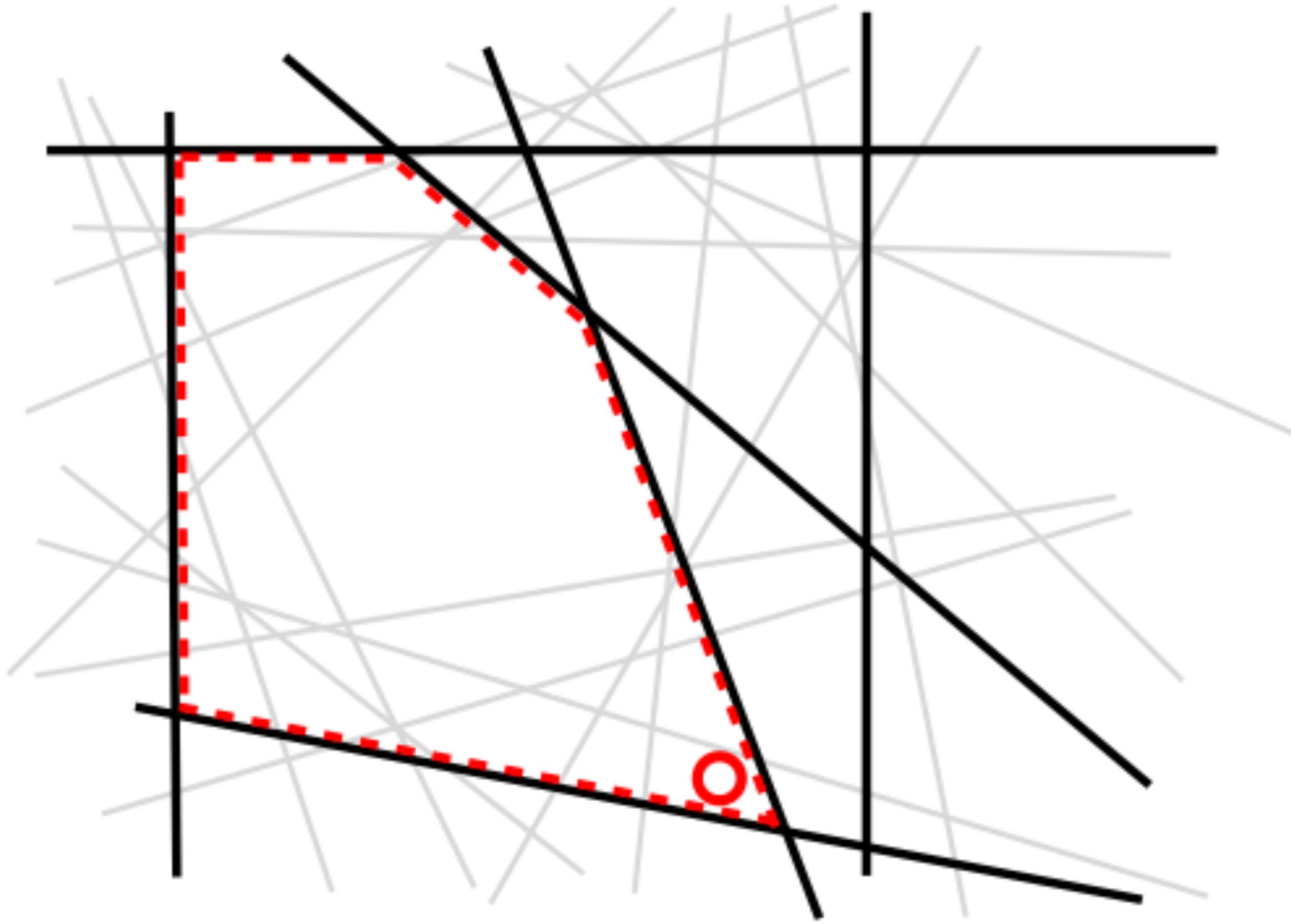
Cutting Plane



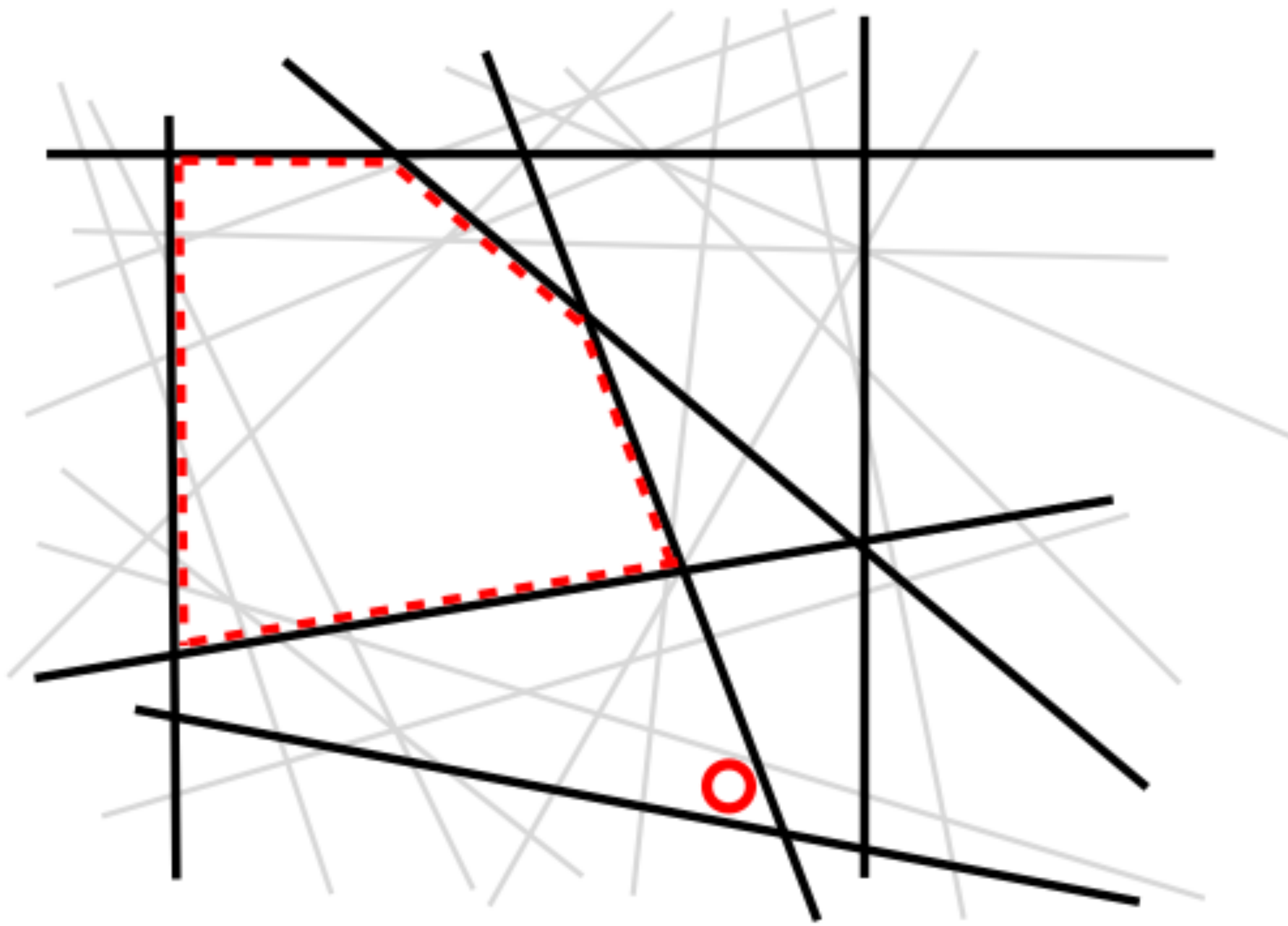
Cutting Plane



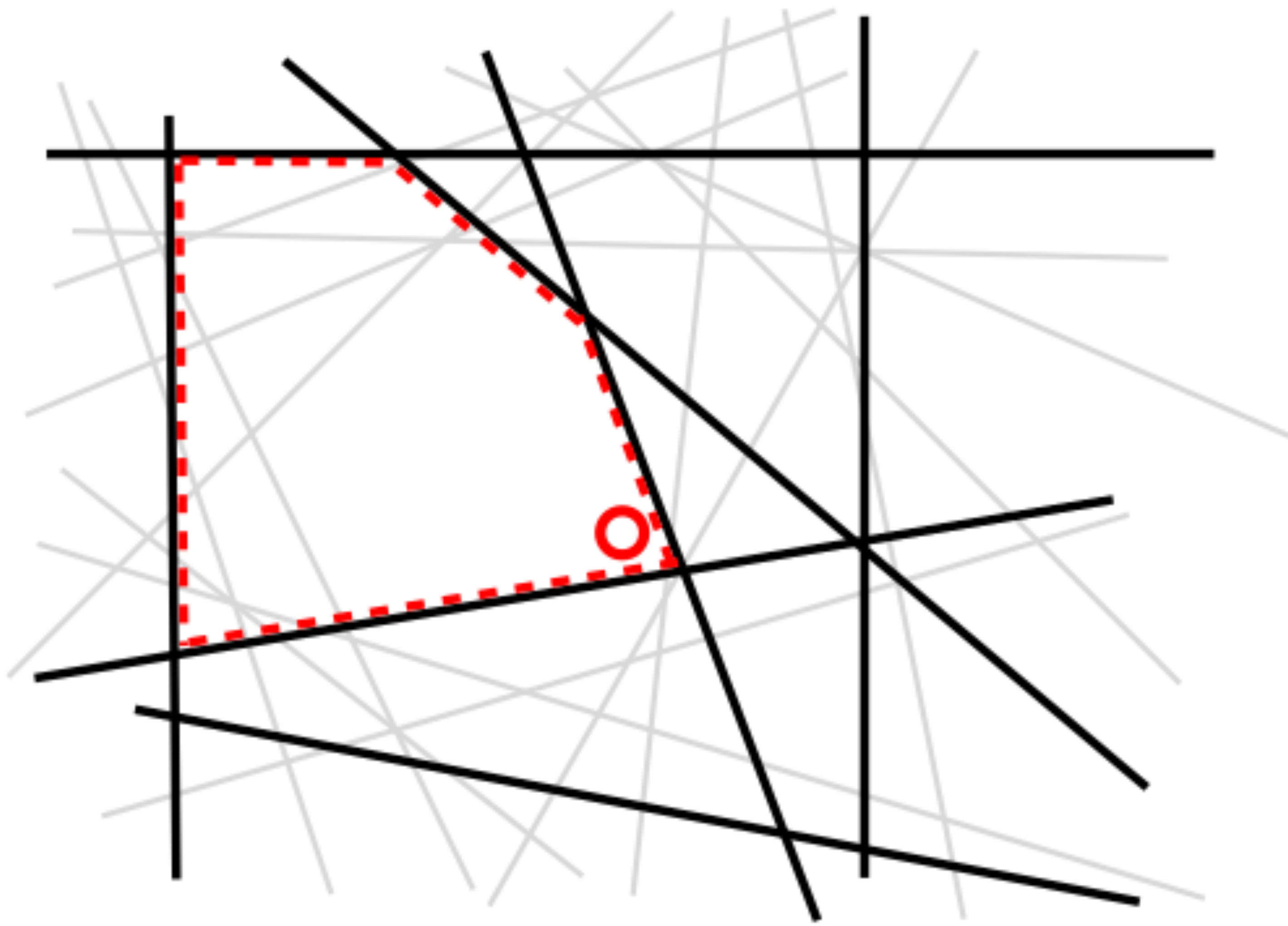
Cutting Plane



Cutting Plane



Cutting Plane



Optimization Algorithms for SSVMs

- Taskar et al. (2003): SMO based on factored dual
- Bartlett et al. (2004) and Collins et al. (2008): exponentiated gradient
- Tsochantaridis et al. (2005): cutting plane (based on dual)
- Taskar et al. (2005): dual extragradient

Easiest to use, in my opinion:

- Ratliff et al. (2006): (stochastic) subgradient descent
- Crammer et al. (2006): online "passive-aggressive" algorithms

Stochastic Subgradient Descent

- Unconstrained primal objective:

$$L(\mathbf{w}, \boldsymbol{x}, \boldsymbol{y}) \quad = \quad -\mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) + \max_{\boldsymbol{y}'} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}') + \mathrm{cost}(\boldsymbol{y}', \boldsymbol{y})$$

- Resulting subgradient:

$$-g_j(\boldsymbol{x}, \boldsymbol{y}) + g_j(\boldsymbol{x}, \mathrm{cost\_augmented\_decode}(\mathbf{w}, \boldsymbol{x}))$$

– Only requires loss-augmented decode! No need for marginals / summing (cf. CRF)

"Passive Aggressive" Learners

- Starting point is the perceptron, and the focus is on the step size.
- In NLP, people often use a specific instance called "1-best MIRA" (margin infused relaxation algorithm).
  - Sometimes with regular decoding, sometimes cost-augmented decoding.
- I do not understand the name (kind of I do)

Passive-Aggressive Update in a Nutshell ("1-best MIRA")

- Given x (and y), perform decoding (or cost-augmented decoding) to obtain y'.
- To get the updated weights, solve:

$$\min_{\mathbf{w}'} \|\mathbf{w}' - \mathbf{w}\|_2^2$$
$$\text{s.t.} \quad \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) - \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}') \geq \text{cost}(\boldsymbol{y}', \boldsymbol{y})$$

- Closed form solution!
  - Essentially, a subgradient update with a closed-form step size.

Perceptron and PA

- The PA papers (e.g., Crammer et al., 2006) take a procedural view of online learning and prove convergence and regret-style bounds.
- An alternative view, described by Martins et al. (2010), derives the same updates via dual coordinate ascent.
  - Confusing name: it doesn't work in the dual!
  - More general: applies to many other loss functions, so you can get a closed-form step size for perceptron and CRFs.
  - Assumes L2 regularization; role of regularization constant C is very clear in the form of the update.

Dual Coordinate Ascent Update

$$\mathbf{w} \quad \leftarrow \quad \mathbf{w} - \underbrace{\min\left\{\frac{1}{C}, \frac{L(\mathbf{w}, \boldsymbol{x}, \boldsymbol{y})}{\|\nabla_{\mathbf{w}} L(\mathbf{w}, \boldsymbol{x}, \boldsymbol{y})\|_2^2}\right\}}_{\text{step size}} \underbrace{\nabla_{\mathbf{w}} L(\mathbf{w}, \boldsymbol{x}, \boldsymbol{y})}_{\text{subgradient}}$$

- Assumes L2 regularization.
- 1-best MIRA is a special case with structured hinge loss.
- Can get regularization into perceptron this way (use perceptron loss).
- Can get closed-form step size for CRF stochastic GD.

Hinge Loss and Log Loss

- Hinge loss (M3N):

$$-\mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) + \max_{\boldsymbol{y}'} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}') + \mathrm{cost}(\boldsymbol{y}', \boldsymbol{y})$$

- Log loss (CRF):

$$-\mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) + \log \sum_{\boldsymbol{y}'} \exp \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}')$$

Aside: Probabilities *and* Cost?

- Hinge loss (M3N):

$$-\mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) + \max_{\boldsymbol{y}'} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}') + \mathrm{cost}(\boldsymbol{y}', \boldsymbol{y})$$

- Log loss (CRF):

$$-\mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) + \log \sum_{\boldsymbol{y}'} \exp \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}')$$

- “Softmax margin” (Gimpel and Smith, 2010):

$$-\mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) + \log \sum_{\boldsymbol{y}'} \exp \left(\mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}') + \mathrm{cost}(\boldsymbol{y}', \boldsymbol{y})\right)$$

Loss Functions You Know

Name	Expression of $L(\mathbf{w}, \boldsymbol{x}, \boldsymbol{y})$
Log loss (joint)	$-\log p(\boldsymbol{x}, \boldsymbol{y} \mid \mathbf{w})$
Log loss (conditional)	$-\log p(\boldsymbol{y} \mid \boldsymbol{x}, \mathbf{w})$
Cost	$\mathrm{cost}(\mathrm{decode}(\mathbf{w}, \boldsymbol{x}), \boldsymbol{y})$
Expected cost, a.k.a. “risk”	$\mathbb{E}_{p(\boldsymbol{Y} \mid \boldsymbol{x}, \mathbf{w})}[\mathrm{cost}(\boldsymbol{Y}, \boldsymbol{y})]$
Perceptron loss	$\max_{\boldsymbol{y}'} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}') - \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y})$
Hinge (margin rescaling version)	$\max_{\boldsymbol{y}'} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}') + \mathrm{cost}(\boldsymbol{y}', \boldsymbol{y}) - \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y})$

On Regularization

- In principle, this choice is orthogonal to the loss function.
- L2 is the most common starting place.
- L1 and other sparsity-inducing regularizers have some nice properties, but they can make optimization more complicated

Does this matter?

Practical Advice

- Features still more important than the loss function.
  - But general, easy-to-implement algorithms are quite useful!
- Perceptron is easiest to implement.
- CRFs and SSVMs usually do better.
- If the cost function factors locally, I recommend using a hinge loss and stochastic subgradient descent.
- Tune the regularization constant.
  - Never on the test data.