Support Vector Machines Meet Goldfarb Cubes

Bernd Gärtner, IPAM Conference "Quo Vadis Hirsch Conjecture?"

January 19, 2011

Disproving the Hirsch Conjecture of Machine Learning

Bernd Gärtner, IPAM Conference "Quo Vadis Hirsch Conjecture?"

January 19, 2011

Outline

- Introduction to Support Vector Machines, and their interpretation as parametric *quadratic* programs
- Statement of the main result: disproof of a conjecture by Hastie et al. concerning the solution path complexity of support vector machines
- Parametric *linear* programs, the shadow vertex pivot rule, and the Goldfarb cubes
- Putting it together: constructing exponentially long solution paths for support vector machines, using Goldfarb cubes (sketch)

Support Vector Machines (SVM)

- Suppose an optical character recognition needs to distinguish between the letters ‘A’ and ‘B’

Support Vector Machines

- Suppose an optical character recognition needs to distinguish between the letters ‘A’ and ‘B’

- Training phase: the system gets to see letters ‘A’ and ‘B’ **plus** the information whether it’s an ‘A’ or a ‘B’

Support Vector Machines

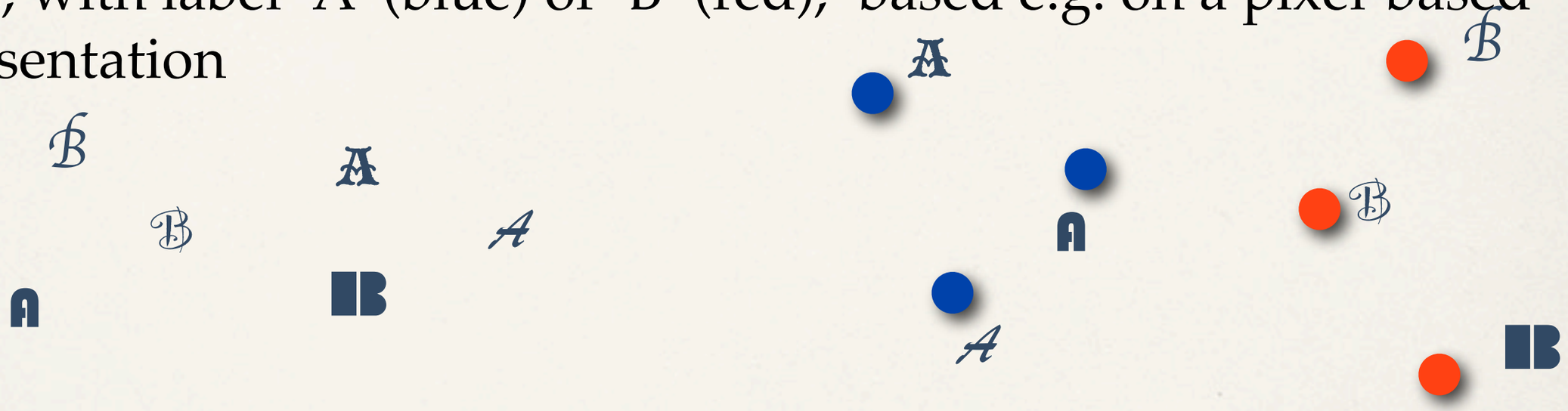
- Suppose an optical character recognition needs to distinguish between the letters ‘A’ and ‘B’

- Training phase: the system gets to see letters ‘A’ and ‘B’ **plus** the information whether it’s an ‘A’ or a ‘B’

- After the training phase, the system is supposed to decide on its own which letter it sees.

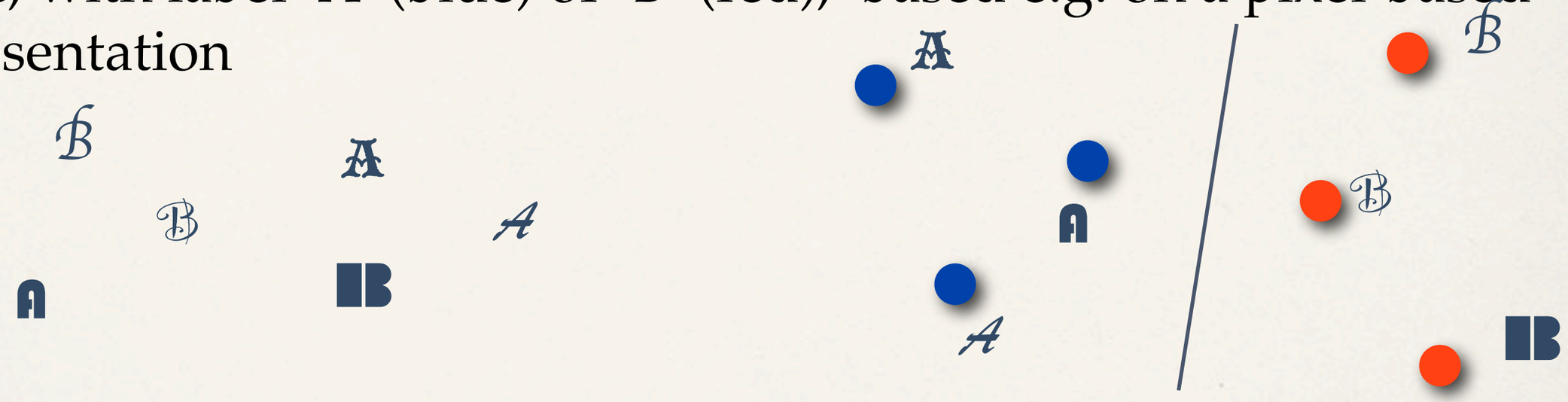
Support Vector Machines

- Solution: map training letters to points in some high-dimensional space, with label 'A' (blue) or 'B' (red), based e.g. on a pixel-based representation



Support Vector Machines

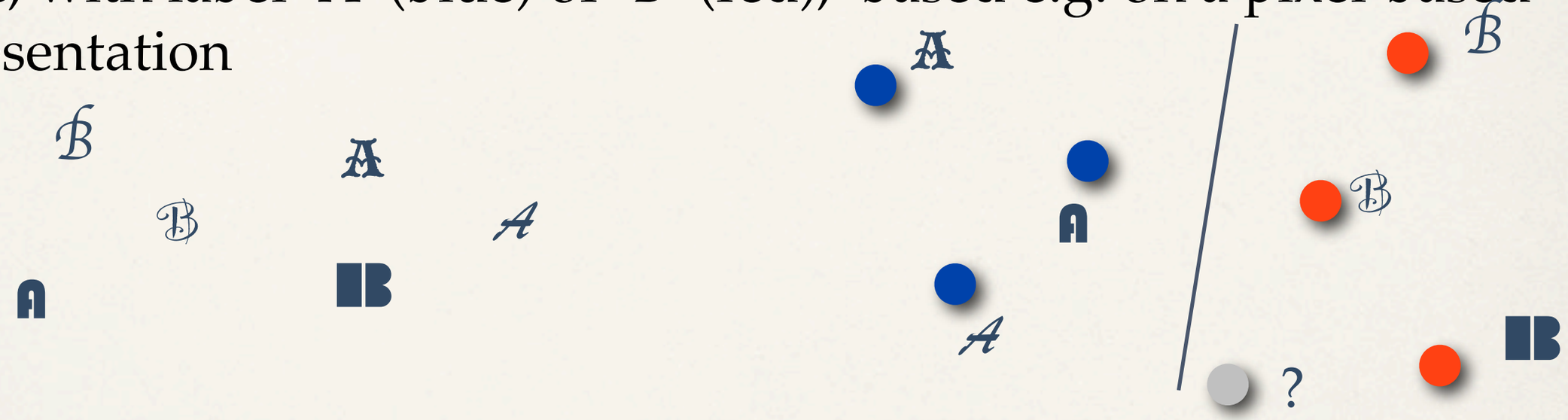
- Solution: map training letters to points in some high-dimensional space, with label 'A' (blue) or 'B' (red), based e.g. on a pixel-based representation



- Separate the 'A's from the 'B's by a simple shape, for example the *maximum-margin hyperplane* (separating hyperplane of maximum distance to any point).

Support Vector Machines

- Solution: map training letters to points in some high-dimensional space, with label 'A' (blue) or 'B' (red), based e.g. on a pixel-based representation



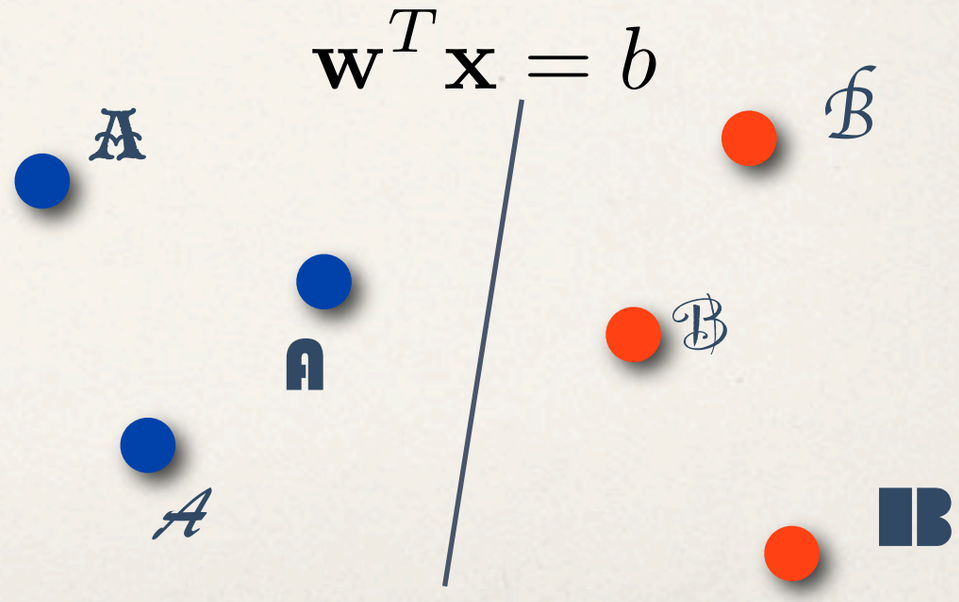
- Separate the 'A's from the 'B's by a simple shape, for example the *maximum-margin hyperplane* (separating hyperplane of maximum distance to any point).

- Classify an unknown letter according to this hyperplane (= 'B')

Support Vector Machines
The primal program

- Computation of Maximum-Margin Hyperplane is a **convex quadratic program:**

$$\begin{array}{lrcl} \min_{\mathbf{w},b} & \frac{1}{2}\mathbf{w}^T\mathbf{w} & & \\ \text{subject to} & \mathbf{w}^T\mathbf{p}_i - b & \geq & 1, \quad \mathbf{p}_i \in A \\ & \mathbf{w}^T\mathbf{p}_i - b & \leq & 1, \quad \mathbf{p}_i \in B \end{array}$$



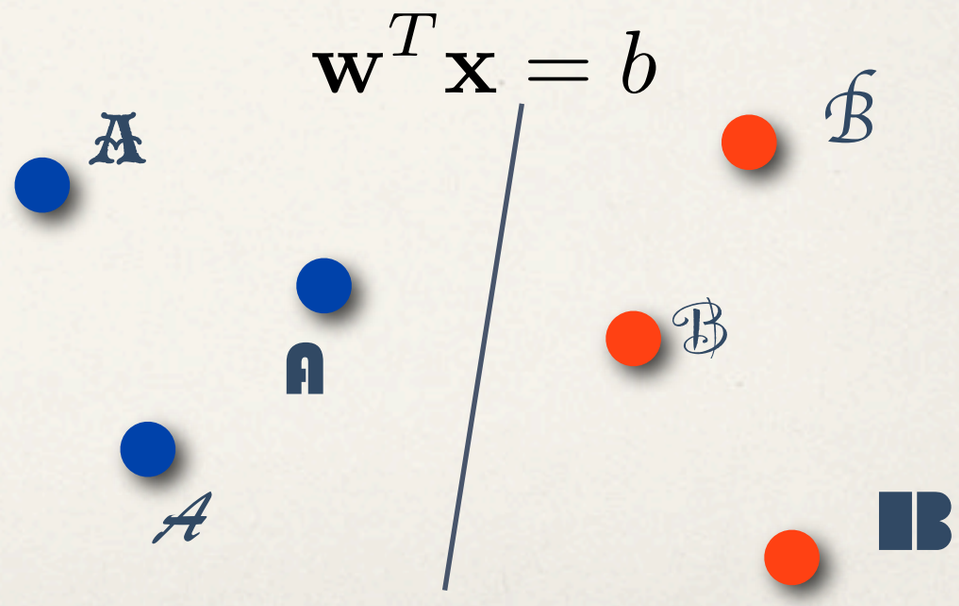
$$\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n$$

Support Vector Machines
The primal program, class labels

- Computation of Maximum-Margin Hyperplane is a **convex quadratic program:**

$$\begin{array}{lll} \min_{\mathbf{w},b} & \frac{1}{2}\mathbf{w}^T\mathbf{w} & \\ \text{subject to} & y_i(\mathbf{w}^T\mathbf{p}_i - b) & \geq \quad 1, \quad i = 1, 2, \ldots, n \end{array}$$

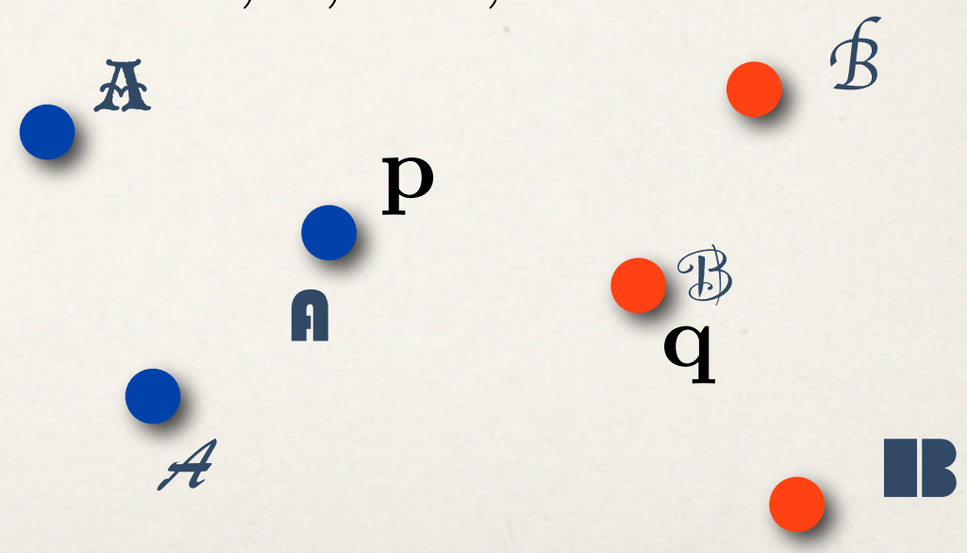
$$y_i = \begin{cases} +1, & \mathbf{p}_i \in A \\ -1, & \mathbf{p}_i \in B \end{cases}$$



$$\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n$$

Support Vector Machines
Dual program

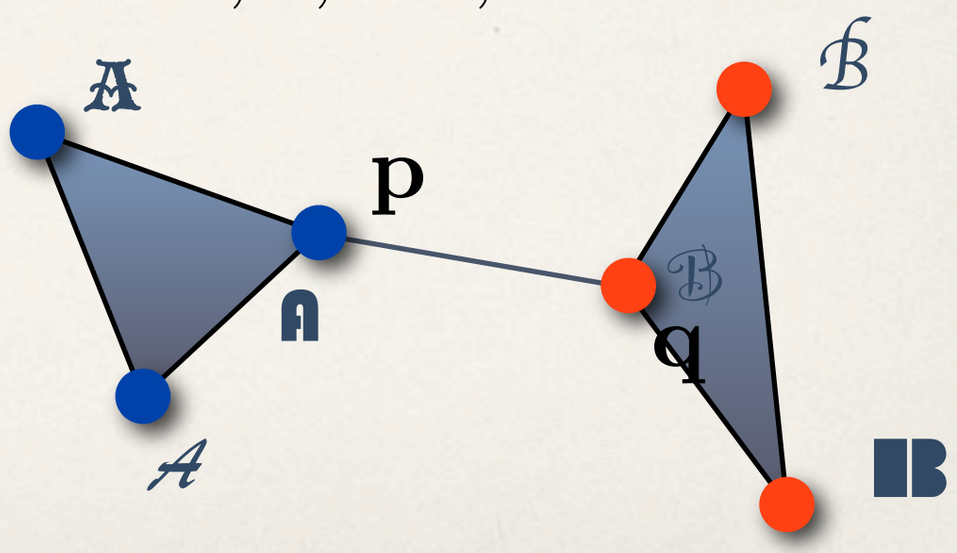
$$
\begin{array}{lll}
\min_{\lambda_i} & \|\mathbf{p}-\mathbf{q}\|^2 & \\
\text{subject to} & \sum_{\mathbf{p}_i \in A} \lambda_i \mathbf{p}_i = \mathbf{p} & \\
& \sum_{\mathbf{p}_i \in B} \lambda_i \mathbf{p}_i = \mathbf{q} & \\
& \sum_{\mathbf{p}_i \in A} \lambda_i = 1 & \\
& \sum_{\mathbf{p}_i \in B} \lambda_i = 1 & \\
& \lambda_i \geq 0 & i = 1, 2, \ldots, n
\end{array}
$$



Support Vector Machines
Dual program

$$
\begin{array}{lll}
\min_{\lambda_i} & \|\mathbf{p}-\mathbf{q}\|^2 & \\
\text{subject to} & \sum_{\mathbf{p}_i \in A} \lambda_i \mathbf{p}_i = \mathbf{p} & \\
& \sum_{\mathbf{p}_i \in B} \lambda_i \mathbf{p}_i = \mathbf{q} & \\
& \sum_{\mathbf{p}_i \in A} \lambda_i = 1 & \\
& \sum_{\mathbf{p}_i \in B} \lambda_i = 1 & \\
& \lambda_i \geq 0 & i = 1, 2, \ldots, n
\end{array}
$$

Geometrically, this is the problem of finding the distance between the two polytopes $conv(A)$ and $conv(B)$

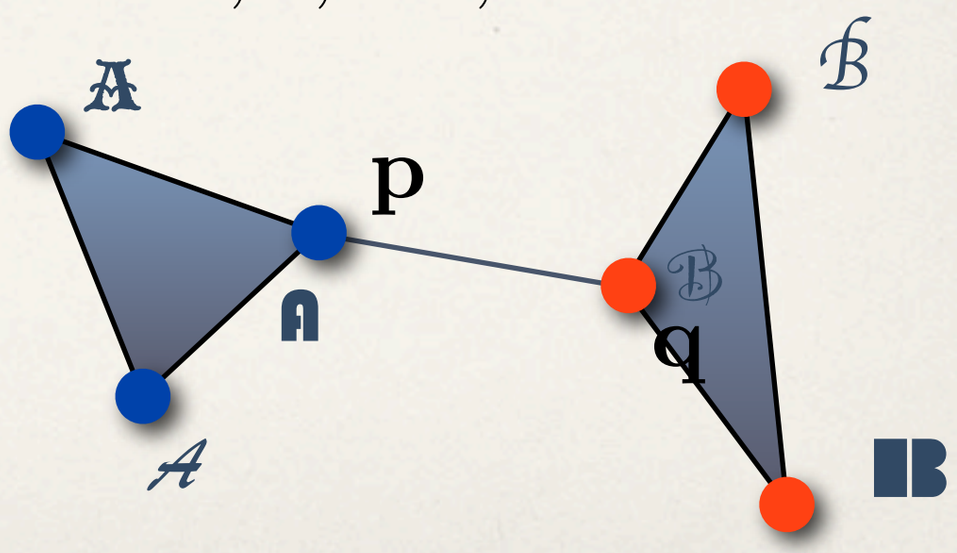


Support Vector Machines
Dual program

$$
\begin{array}{lll}
\min_{\lambda_i} & \|\mathbf{p}-\mathbf{q}\|^2 & \\
\text{subject to} & \sum_{\mathbf{p}_i \in A} \lambda_i \mathbf{p}_i = \mathbf{p} & \\
& \sum_{\mathbf{p}_i \in B} \lambda_i \mathbf{p}_i = \mathbf{q} & \\
& \sum_{\mathbf{p}_i \in A} \lambda_i = 1 & \\
& \sum_{\mathbf{p}_i \in B} \lambda_i = 1 & \\
& \lambda_i \geq 0 & i = 1, 2, \ldots, n
\end{array}
$$

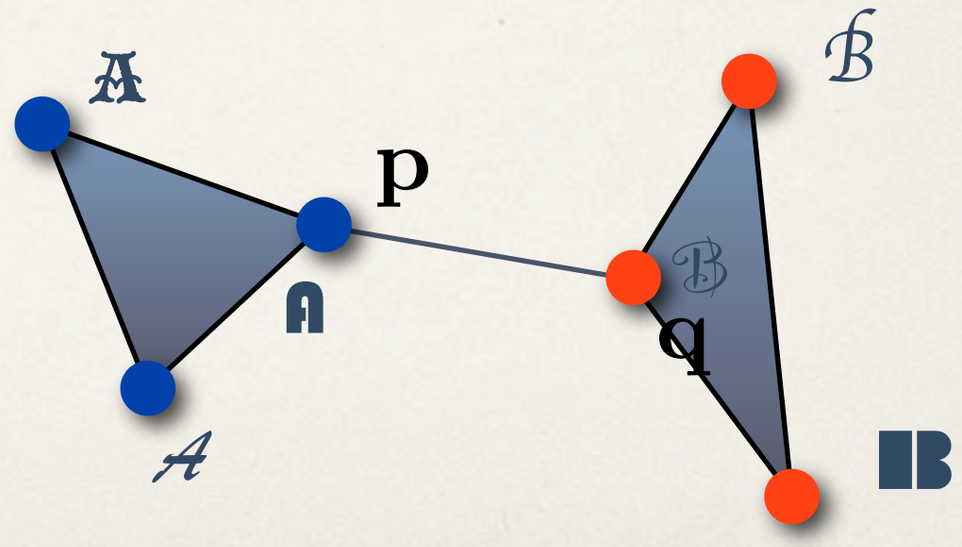
The points with positive λ_i are the *support vectors*

Geometrically, this is the problem of finding the distance between the two polytopes $conv(A)$ and $conv(B)$



Support Vector Machines
Dual program, class labels

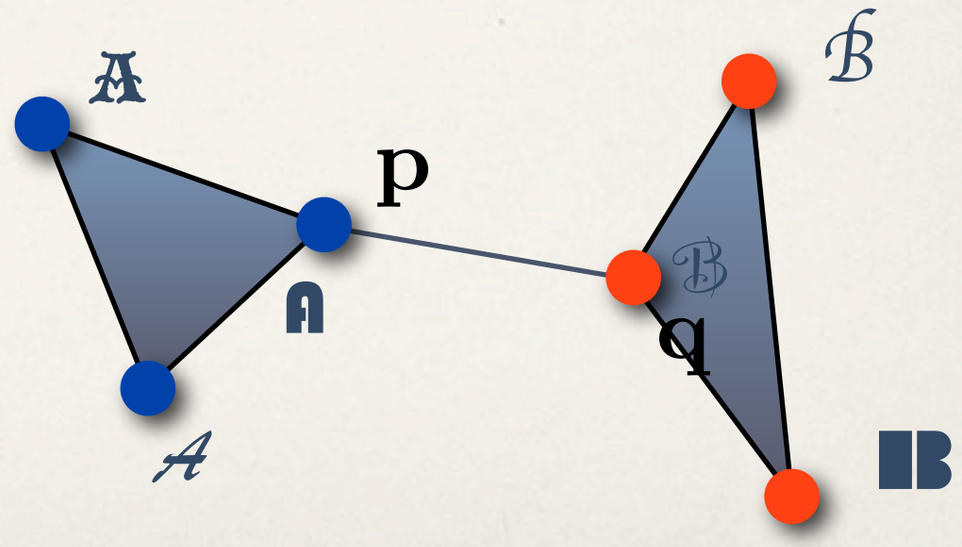
$$
\begin{array}{lll}
\min_{\lambda_i} & \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{p}_i^T \mathbf{p}_j & \\
\text{subject to} & \sum_{\mathbf{p}_i \in A} \lambda_i = 1 & \\
& \sum_{\mathbf{p}_i \in B} \lambda_i = 1 & \\
& \lambda_i \geq 0 & i = 1, 2, \ldots, n
\end{array}
$$



Support Vector Machines
Dual program, class labels

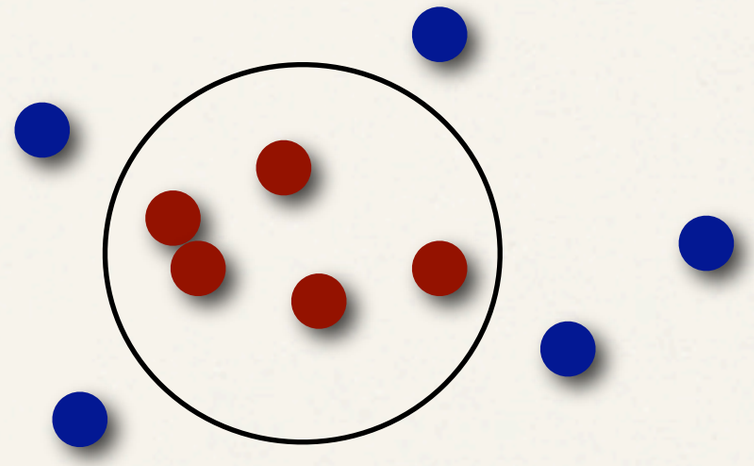
$$
\begin{array}{lll}
\min_{\lambda_i} & \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{p}_i^T \mathbf{p}_j & \\
\text{subject to} & \sum_{\mathbf{p}_i \in A} \lambda_i = 1 & \\
& \sum_{\mathbf{p}_i \in B} \lambda_i = 1 & \\
& \lambda_i \geq 0 & i = 1, 2, \ldots, n
\end{array}
$$

The dual program does not need to know the points, just their pairwise scalar products!



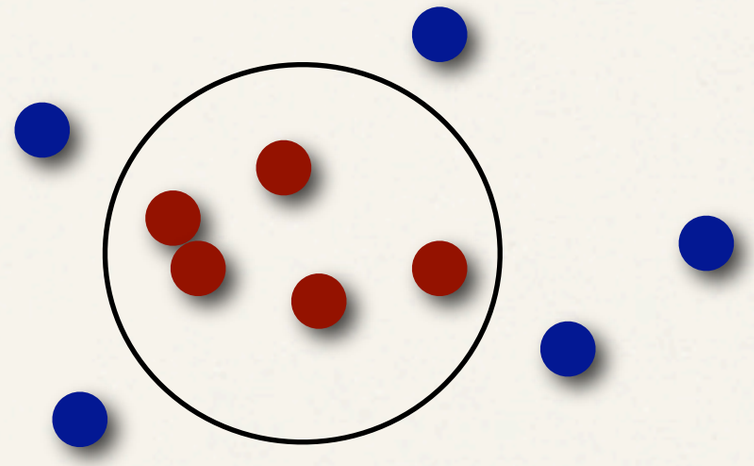
Support Vector Machines
Nonlinear separators

- Other separating shapes (e.g. spheres) may be the "right" shapes for the given problem



Support Vector Machines
Nonlinear separators

- Other separating shapes (e.g. spheres) may be the "right" shapes for the given problem



- Apply *lifting* to reduce to separating hyperplane in higher dimension!

Support Vector Machines
Circular Separators

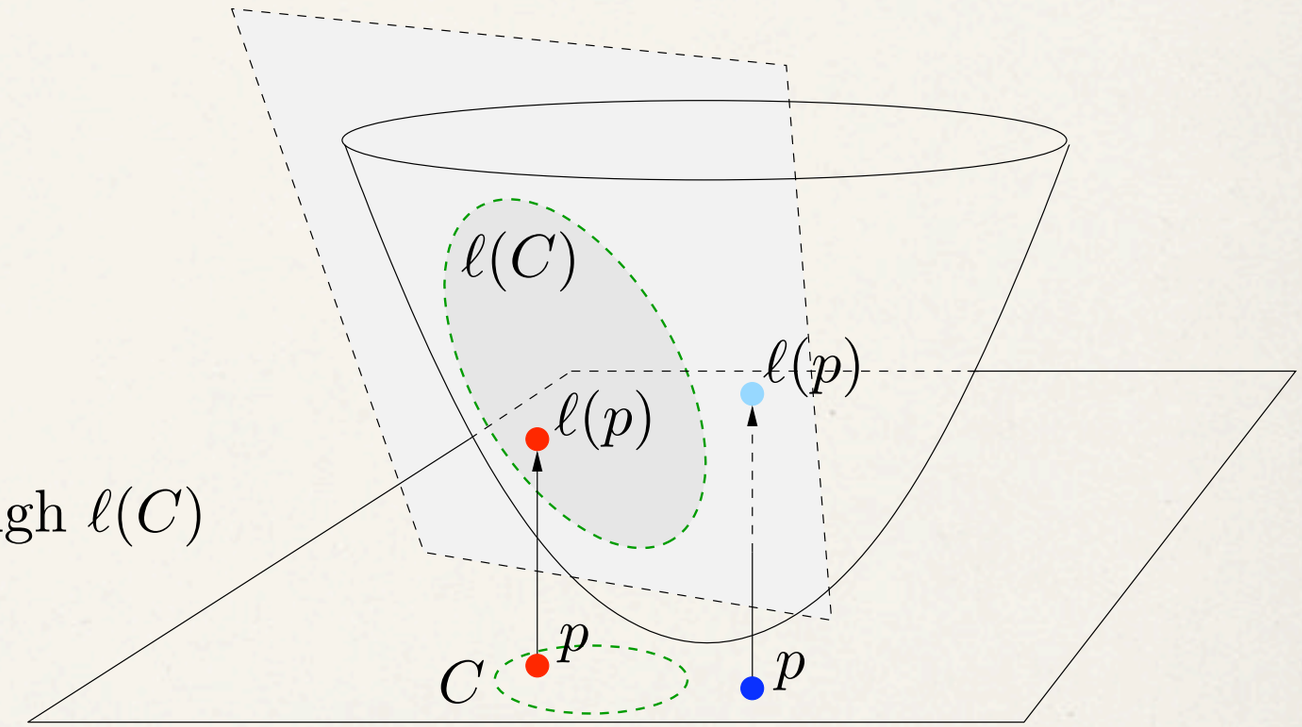
- Apply *lifting map*

$$\ell : (x, y) \mapsto (x, y, x^2 + y^2)$$

$$p \left\{ \begin{array}{l} \text{inside} \\ \text{on} \\ \text{outside} \end{array} \right\} C$$

$$\Updownarrow$$

$$\ell(p) \left\{ \begin{array}{l} \text{below} \\ \text{on} \\ \text{above} \end{array} \right\} \text{the plane through } \ell(C)$$



Support Vector Machines
Circular Separators

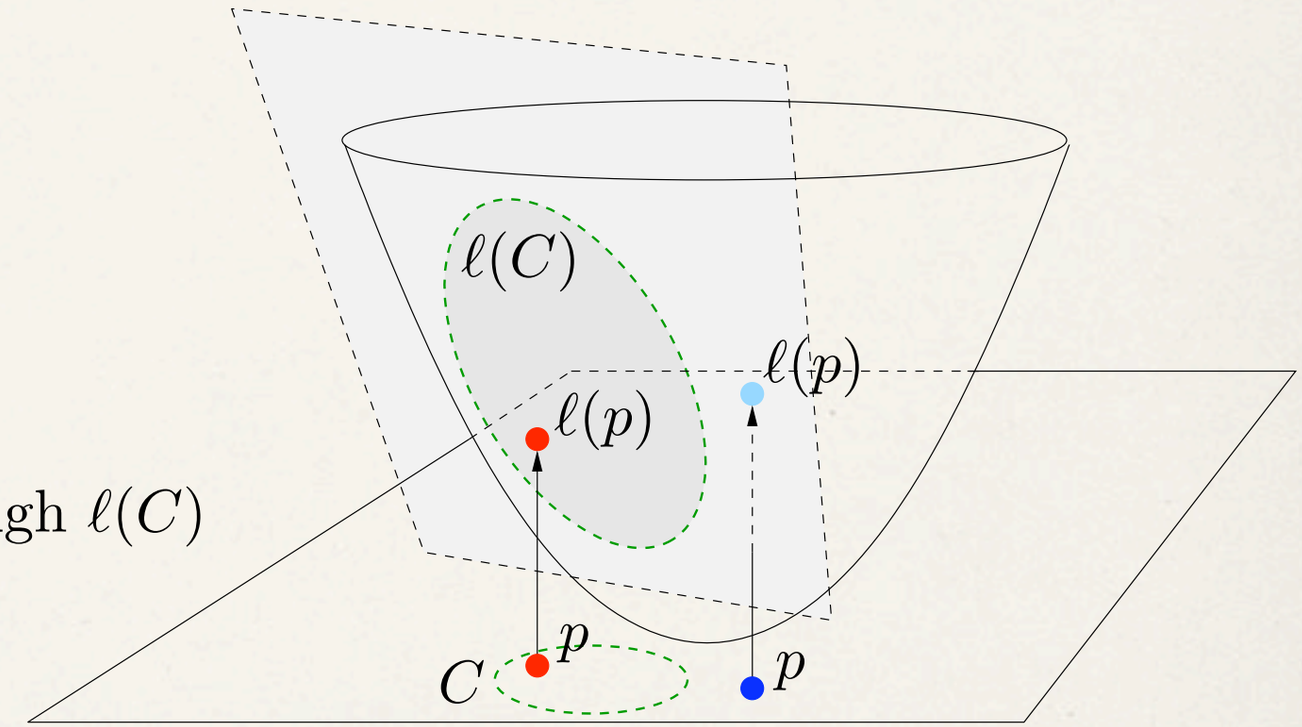
- Apply *lifting map*

$$\ell : (x, y) \mapsto (x, y, x^2 + y^2)$$

$$p \left\{ \begin{array}{l} \text{inside} \\ \text{on} \\ \text{outside} \end{array} \right\} C$$

$$\Updownarrow$$

$$\ell(p) \left\{ \begin{array}{l} \text{below} \\ \text{on} \\ \text{above} \end{array} \right\} \text{the plane through } \ell(C)$$



- Use preimage of maximum-margin hyperplane in lifted space

Support Vector Machines
Circular Separators

- Apply *lifting map*

$$\ell : (x, y) \mapsto (x, y, x^2 + y^2)$$

- Dual quadratic program becomes

$$\begin{array}{lll} \min_{\lambda_i} & \sum_{i,j} \lambda_i \lambda_j y_i y_j k(\mathbf{p}_i, \mathbf{p}_j) & \\ \text{subject to} & \sum_{\mathbf{p}_i \in A} \lambda_i = 1 & \\ & \sum_{\mathbf{p}_i \in B} \lambda_i = 1 & \\ & \lambda_i \geq 0 & i = 1, 2, \ldots, n \end{array}$$

$$k(\mathbf{p}, \mathbf{q}) = \mathbf{p}_x \mathbf{q}_x + \mathbf{p}_y \mathbf{q}_y + (\mathbf{p}_x^2 + \mathbf{p}_y^2)(\mathbf{q}_x^2 + \mathbf{q}_y)^2$$

Support Vector Machines
Circular Separators

- Apply *lifting map*

$$\ell : (x, y) \mapsto (x, y, x^2 + y^2)$$

- Dual quadratic program becomes

$$\begin{array}{lll} \min_{\lambda_i} & \sum_{i,j} \lambda_i \lambda_j y_i y_j k(\mathbf{p}_i, \mathbf{p}_j) & \\ \text{subject to} & \sum_{\mathbf{p}_i \in A} \lambda_i = 1 & \\ & \sum_{\mathbf{p}_i \in B} \lambda_i = 1 & \\ & \lambda_i \geq 0 & i = 1, 2, \ldots, n \end{array}$$

$$k(\mathbf{p}, \mathbf{q}) = \mathbf{p}_x \mathbf{q}_x + \mathbf{p}_y \mathbf{q}_y + (\mathbf{p}_x^2 + \mathbf{p}_y^2)(\mathbf{q}_x^2 + \mathbf{q}_y)^2$$ ← *Kernel function*

Support Vector Machines
Kernel Trick

- Let $k : \mathbf{R}^d \times \mathbf{R}^d \to \mathbf{R}$ be symmetric and positive semidefinite, meaning that

$$\sum_{i,j} \lambda_i \lambda_j k(\mathbf{p}_i, \mathbf{p}_j) \geq 0, \quad \forall \mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n, \lambda_1, \lambda_2, \ldots, \lambda_n$$

Support Vector Machines
Kernel Trick

- Let $k : \mathbf{R}^d \times \mathbf{R}^d \to \mathbf{R}$ be symmetric and positive semidefinite, meaning that

$$\sum_{i,j} \lambda_i \lambda_j k(\mathbf{p}_i, \mathbf{p}_j) \geq 0, \quad \forall \mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n, \lambda_1, \lambda_2, \ldots, \lambda_n$$

- Then the kernel k is a scalar product in some (possibly infinite dimensional) space H *(Mercer's Theorem)*

Support Vector Machines
Kernel Trick

- Let $k : \mathbf{R}^d \times \mathbf{R}^d \to \mathbf{R}$ be symmetric and positive semidefinite, meaning that

$$\sum_{i,j} \lambda_i \lambda_j k(\mathbf{p}_i, \mathbf{p}_j) \geq 0, \quad \forall \mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n, \lambda_1, \lambda_2, \ldots, \lambda_n$$

- Then the kernel k is a scalar product in some (possibly infinite dimensional) space H *(Mercer's Theorem)*

The quadratic program implicitly computes the maximum-margin hyperplane in H

$$\begin{array}{lll} \min_{\lambda_i} & \sum_{i,j} \lambda_i \lambda_j y_i y_j k(\mathbf{p}_i, \mathbf{p}_j) & \\ \text{subject to} & \sum_{\mathbf{p}_i \in A} \lambda_i = 1 & \\ & \sum_{\mathbf{p}_i \in B} \lambda_i = 1 & \\ & \lambda_i \geq 0 & i = 1, 2, \ldots, n \end{array}$$

Support Vector Machines Kernel Trick

- Then the kernel *k* is a scalar product in some (possibly infinite dimensional) space *H (Mercer's Theorem)*

The quadratic program implicitly computes the maximum-margin hyperplane in *H*

$$
\begin{array}{lll}
\min_{\lambda_i} & \sum_{i,j} \lambda_i \lambda_j y_i y_j k(\mathbf{p}_i, \mathbf{p}_j) & \\
\text{subject to} & \sum_{\mathbf{p}_i \in A} \lambda_i = 1 & \\
& \sum_{\mathbf{p}_i \in B} \lambda_i = 1 & \\
& \lambda_i \geq 0 & i = 1, 2, \ldots, n
\end{array}
$$

- Separating shape in the original space may be "weird" and hard to compute, but we can still classify unknown points based on it!

Support Vector Machines Kernel Trick

- Then the kernel *k* is a scalar product in some (possibly infinite dimensional) space *H (Mercer's Theorem)*

The quadratic program implicitly computes the maximum-margin hyperplane in *H*

$$\begin{array}{lll} \min_{\lambda_i} & \sum_{i,j} \lambda_i \lambda_j y_i y_j k(\mathbf{p}_i, \mathbf{p}_j) & \\ \text{subject to} & \sum_{\mathbf{p}_i \in A} \lambda_i & = \quad 1 \\ & \sum_{\mathbf{p}_i \in B} \lambda_i & = \quad 1 \\ & \lambda_i & \geq \quad 0 \quad i = 1, 2, \ldots, n \end{array}$$

- Separating shape in the original space may be "weird" and hard to compute, but we can still classify unknown points based on it!

- Compute $\mathbf{w}^T \mathbf{x} - b$, where $\mathbf{w}^T \mathbf{x} = \sum_i \lambda_i y_i k(\mathbf{p}_i, \mathbf{x})$

support vector

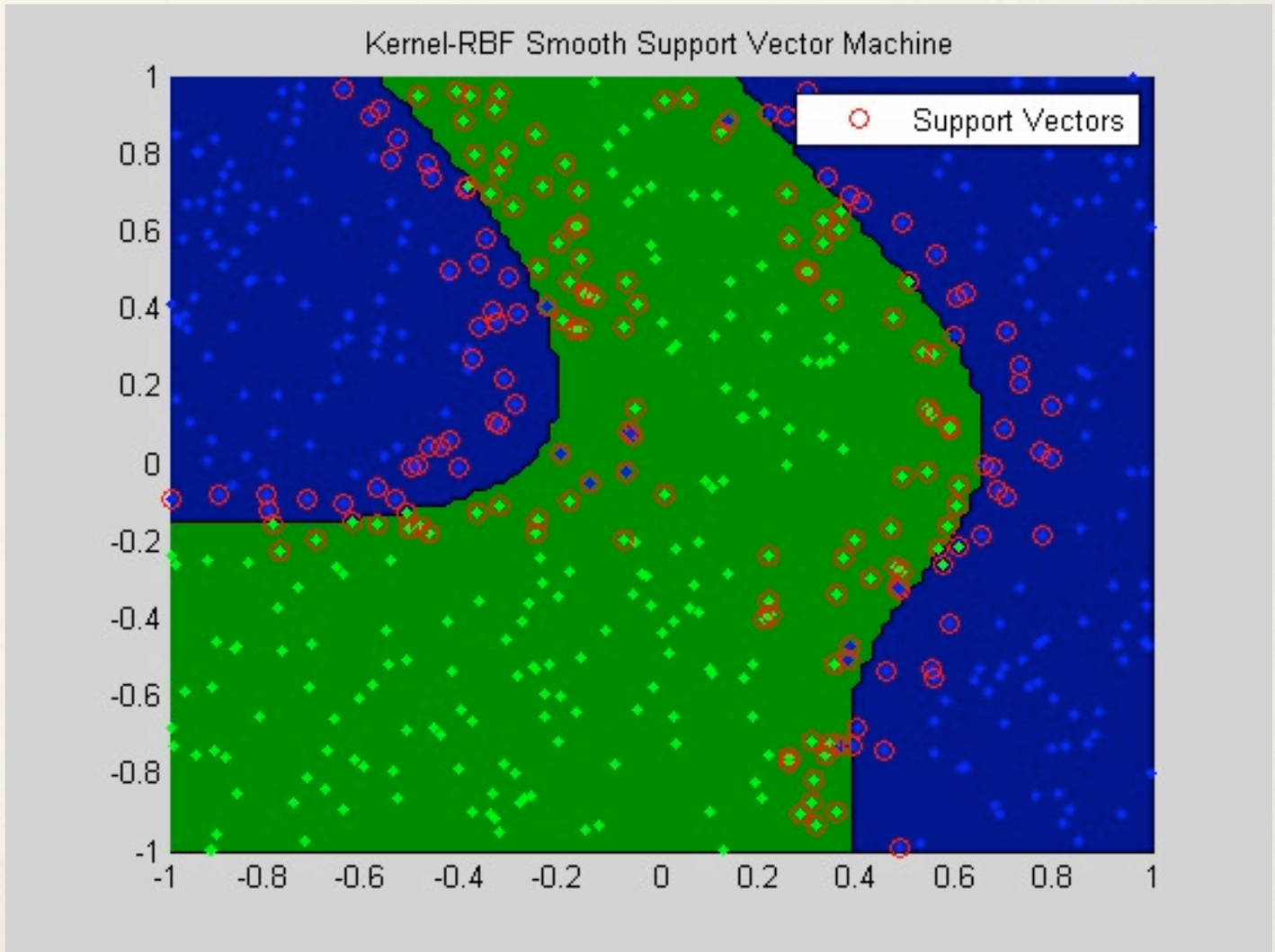
$$b = \sum_i^i \lambda_i y_i k(\mathbf{p}_i, \mathbf{p}_j) - y_j$$

Support Vector Machines
Kernel Trick

- Visualization of nonlinear separator in original space

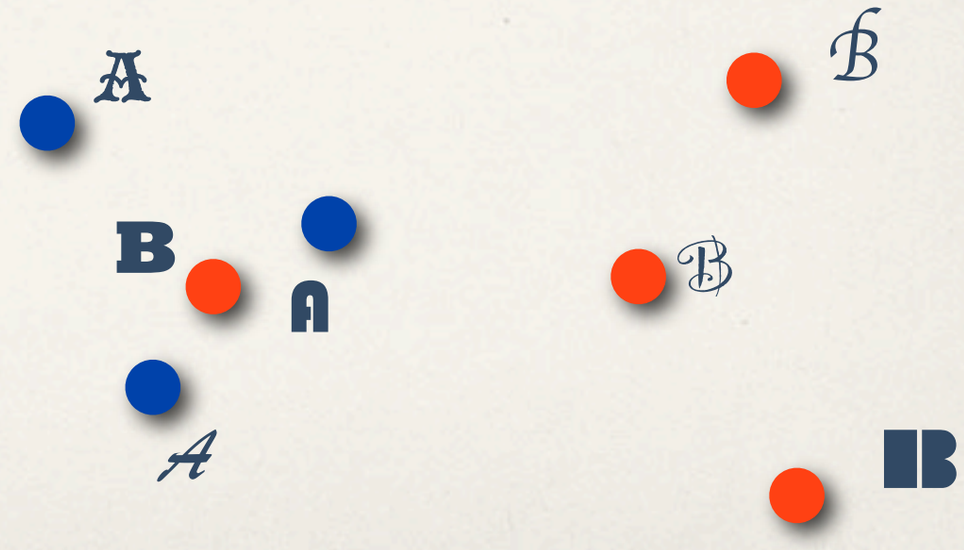
$$k(\mathbf{p}, \mathbf{q}) = e^{-\mu \|\mathbf{p}-\mathbf{q}\|}$$

"Radial Basis Function" (RBF)



Support Vector Machines
Soft Margin

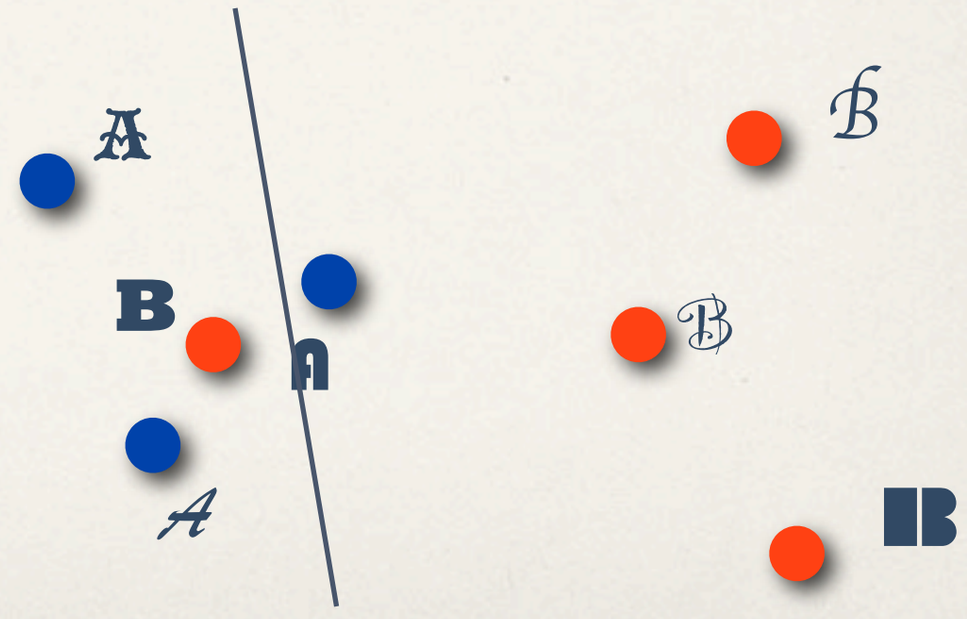
- In many cases, a proper separation cannot be expected, due to noise and outliers



Support Vector Machines
Soft Margin

- In many cases, a proper separation cannot be expected, due to noise and outliers

- Allow misclassifications of training points, but penalize them!



Support Vector Machines
Soft Margin

- In many cases, a proper separation cannot be expected, due to noise and outliers

- Allow misclassifications of training points, but penalize them!

- ν-SVM, where ν is a fixed parameter

$$
\begin{array}{lllll}
\min_{\mathbf{w},b,\rho,\xi_i} & \frac{1}{2}\mathbf{w}^T\mathbf{w} - \nu\rho & + & \frac{1}{n}\sum_{i=1}^{n}\xi_i & \\
\text{subject to} & y_i(\mathbf{w}^T\mathbf{p}_i - b) & \geq & \rho - \xi_i, & i = 1, 2, \ldots, n \\
& \xi_i & \geq & 0, & i = 1, 2, \ldots, n \\
& \rho & \geq & 0 &
\end{array}
$$

previously,
we had 1 here

Support Vector Machines
Soft Margin

- In many cases, a proper separation cannot be expected, due to noise and outliers

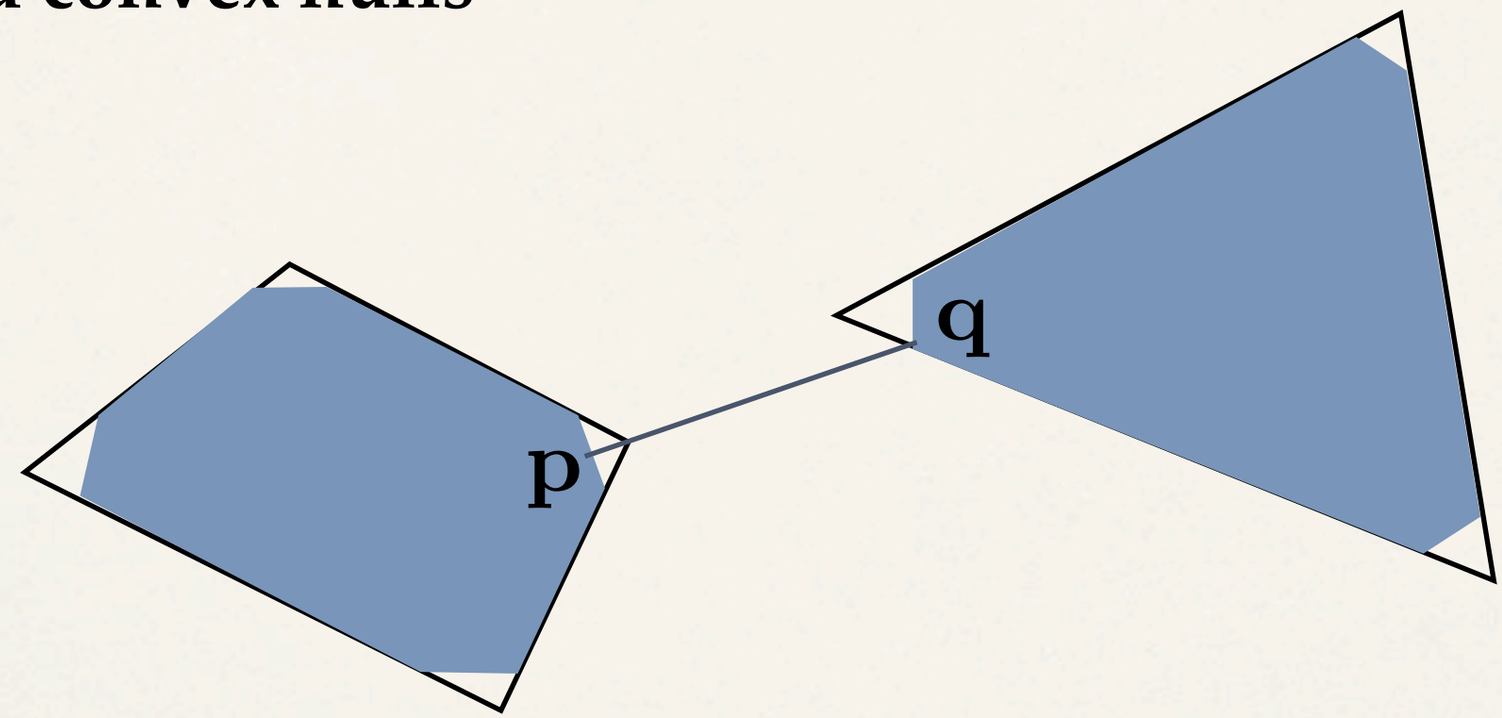
- Allow misclassifications of training points, but penalize them!

- dual ν-SVM (with $\mu := \frac{2}{n\nu}$):

$$
\begin{array}{lll}
\min_{\lambda_i} & \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{p}_i^T \mathbf{p}_j & \\
\text{subject to} & \sum_{\mathbf{p}_i \in A} \lambda_i = 1 & \\
& \sum_{\mathbf{p}_i \in B} \lambda_i = 1 & \\
& \lambda_i \geq 0 & i = 1, 2, \ldots, n \\
\text{new} \longrightarrow & \lambda_i \leq \mu & i = 1, 2, \ldots, n
\end{array}
$$

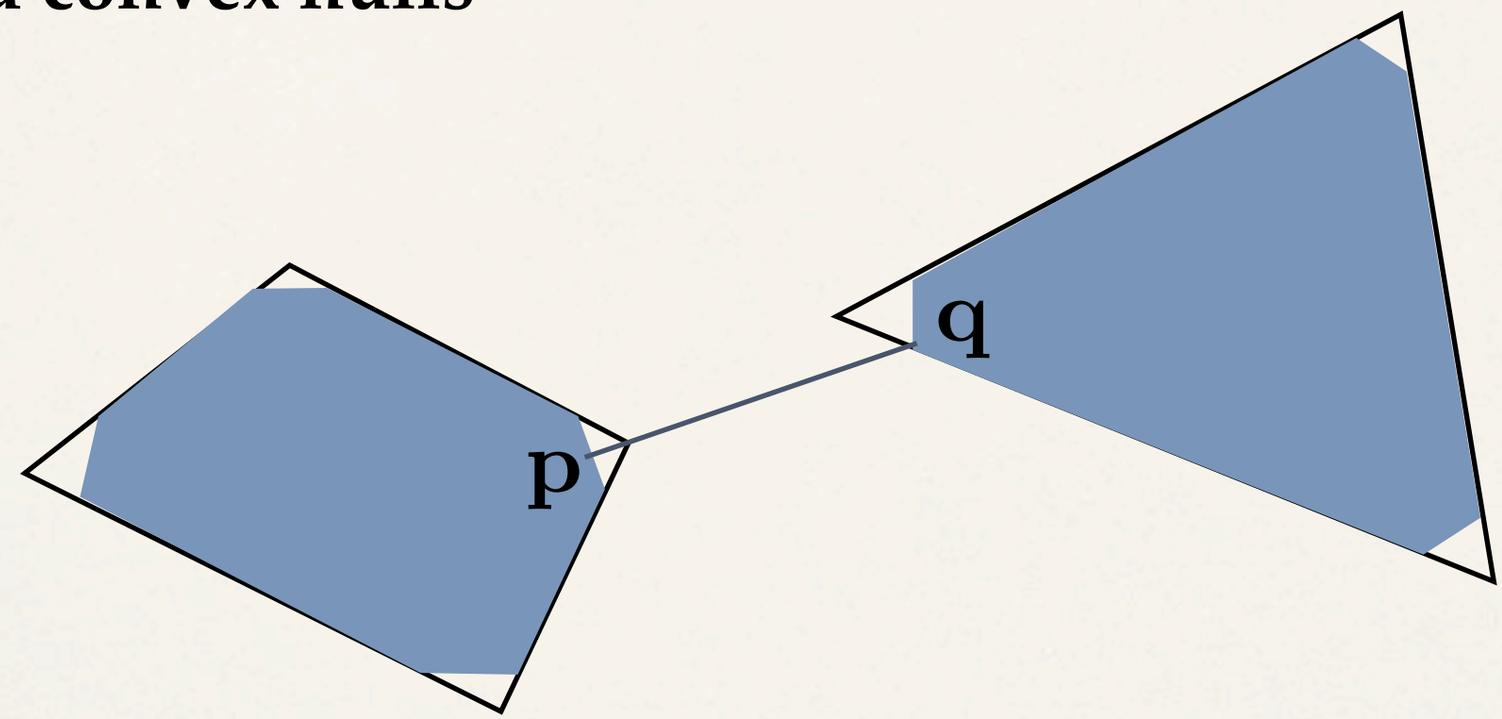
Support Vector Machines
Soft Margin: The Geometric View

- The dual ν-SVM is the problem of finding the distance between two **reduced convex hulls**



Support Vector Machines
Soft Margin: The Geometric View

- The dual ν-SVM is the problem of finding the distance between two **reduced convex hulls**

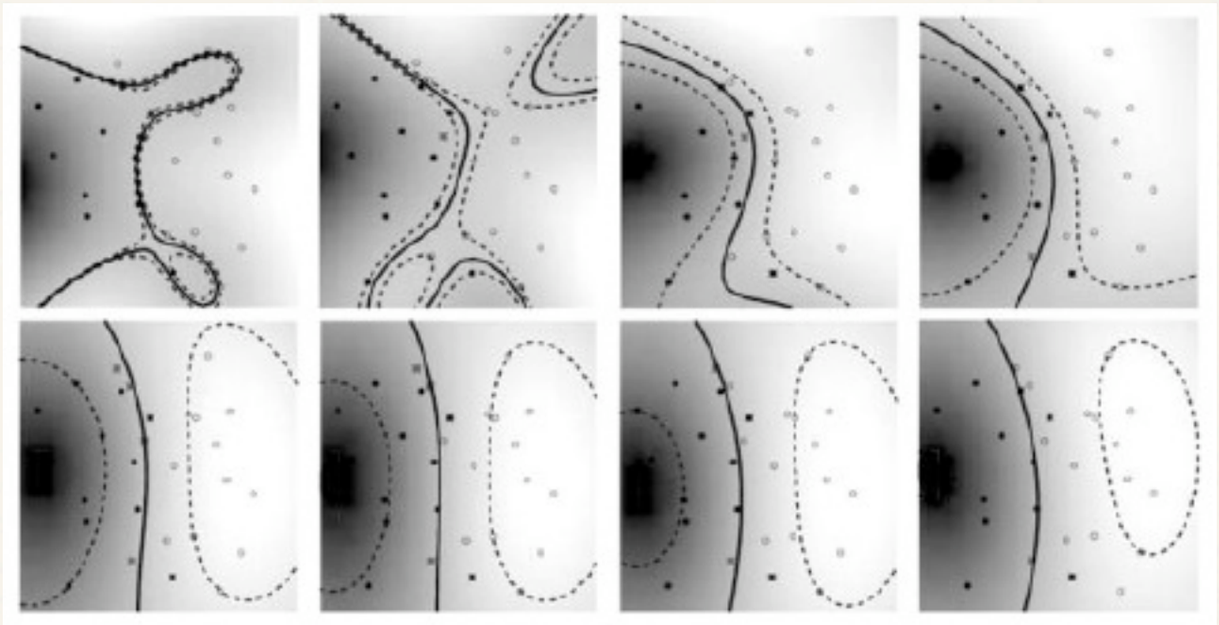


- $\mu = 1$: no reduction

$\mu = 1/2$: edges start to disappear

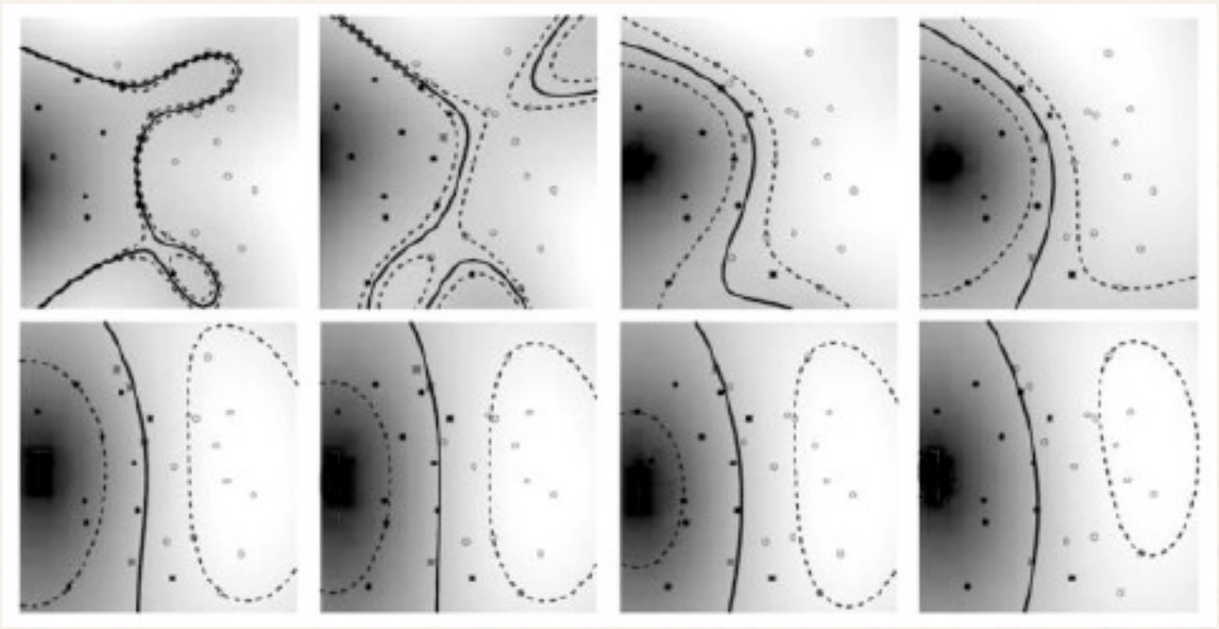
Support Vector Machines
Soft Margin: Parameter Choice

- How does ν (or μ in the dual) influence the separator?

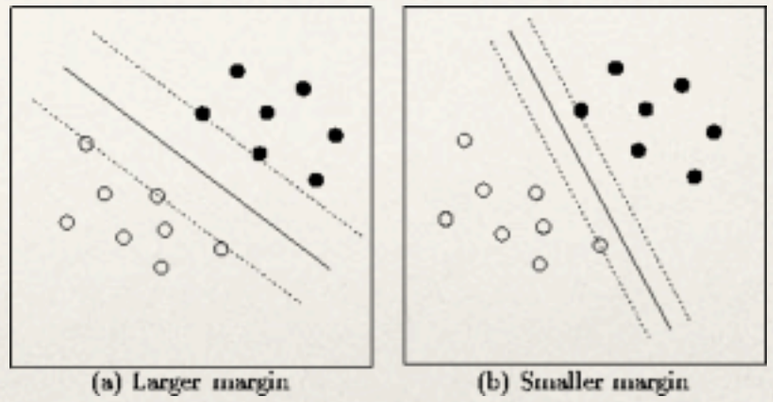


Support Vector Machines
Soft Margin: Parameter Choice

- How does ν (or μ in the dual) influence the separator?



- Small ν: small margin (**overfitting)**
- Large ν: large margin (**underfitting**)



Support Vector Machines
Soft Margin: Parameter Choice

- How to choose the parameter in a given application?

Support Vector Machines
Soft Margin: Parameter Choice

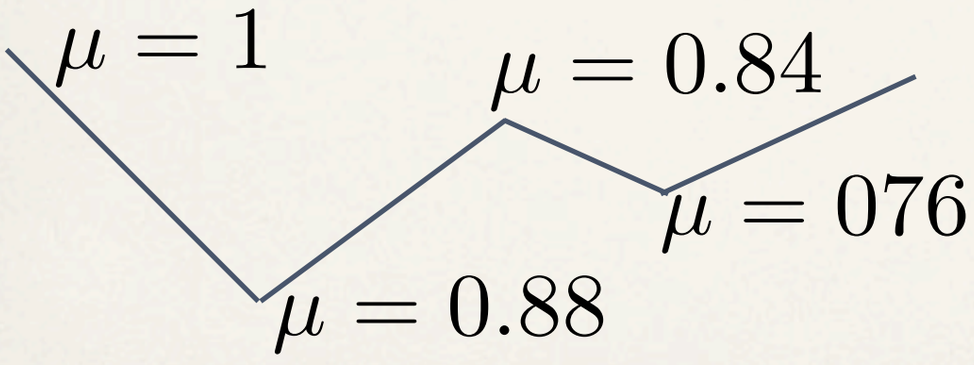
- How to choose the parameter in a given application?
- Standard method: **grid search** (discretize the relevant parameter range and try all values in the discretized range; one QP per value)

Support Vector Machines
Soft Margin: Parameter Choice

- How to choose the parameter in a given application?

- Standard method: **grid search** (discretize the relevant parameter range and try all values in the discretized range; one QP per value)

- Solution path method: compute the solution of the dual SVM as a piecewise linear function in μ, then grid search (one lookup per value)



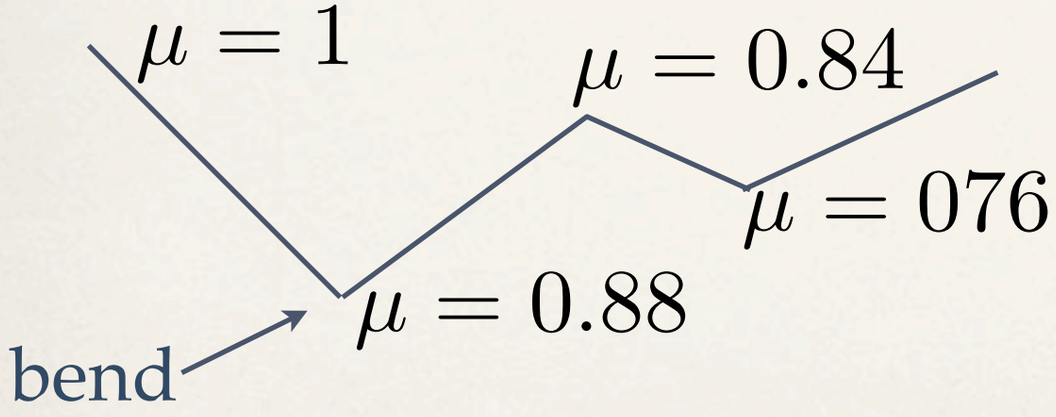
$$
\begin{array}{lllll}
\min_{\lambda_i} & \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{p}_i^T \mathbf{p}_j & & & \\
\text{subject to} & \sum_{\mathbf{p}_i \in A} \lambda_i & = & 1 & \\
& \sum_{\mathbf{p}_i \in B} \lambda_i & = & 1 & \\
& \lambda_i & \geq & 0 & i = 1, 2, \ldots, n \\
& \lambda_i & \leq & \mu & i = 1, 2, \ldots, n
\end{array}
$$

Support Vector Machines
Soft Margin: Parameter Choice

- How to choose the parameter in a given application?

- Standard method: **grid search** (discretize the relevant parameter range and try all values in the discretized range; one QP per value)

- Solution path method: compute the solution of the dual SVM as a piecewise linear function in μ, then grid search (one lookup per value)



$$
\begin{array}{lllll}
\min_{\lambda_i} & \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{p}_i^T \mathbf{p}_j & & & \\
\text{subject to} & \sum_{\mathbf{p}_i \in A} \lambda_i & = & 1 & \\
& \sum_{\mathbf{p}_i \in B} \lambda_i & = & 1 & \\
& \lambda_i & \geq & 0 & i = 1, 2, \ldots, n \\
& \lambda_i & \leq & \mu & i = 1, 2, \ldots, n
\end{array}
$$

- Performance depends on the **number of bends** in the solution path

Support Vector Machines
The Hastie et al. Conjecture

- Hastie et al. 2004: *The entire regularization path for the support vector machine*

Support Vector Machines
The Hastie et al. Conjecture

- Hastie et al. 2004: *The entire regularization path for the support vector machine*

- **Conjecture:** The number of bends is O (min ($|A|$, $|B|$) = O(n)

Support Vector Machines
The Hastie et al. Conjecture

- Hastie et al. 2004: *The entire regularization path for the support vector machine*

- **Conjecture:** The number of bends is O (min ($|A|$, $|B|$) = O(n)

- Conjecture was repeatedly stated in other articles, and it was experimentally "confirmed" in a large number of applications

Support Vector Machines The Conjecture is False

- Hastie et al. 2004: *The entire regularization path for the support vector machine*

- **Conjecture:** The number of bends is O (min ($|A|$, $|B|$) = O(n)

- Conjecture was repeatedly stated in other articles, and it was experimentally "confirmed" in a large number of applications

- **Theorem (G., Jaggi, Maria 2010) :** There are point sets A and B in d-dimensional space with $|A| = 2d$, $|B| = 2$, for which the solution path of the dual ν-SVM has an *exponential* number of at least $2^d / 4$ many bends.

Parametric Linear Programs

- **Recall**: primal ν-SVM is a parametric quadratic program (parameter in the objective function):

$$
\begin{array}{lllll}
\min_{\mathbf{w},b,\rho,\xi_i} & \frac{1}{2}\mathbf{w}^T\mathbf{w} - \nu\rho & + & \frac{1}{n}\sum_{i=1}^{n}\xi_i & \\
\text{subject to} & y_i(\mathbf{w}^T\mathbf{p}_i - b) & \geq & \rho - \xi_i, & i = 1, 2, \ldots, n \\
& \xi_i & \geq & 0, & i = 1, 2, \ldots, n \\
& \rho & \geq & 0 &
\end{array}
$$

Parametric Linear Programs

- **Recall**: primal ν-SVM is a parametric quadratic program (parameter in the objective function):

$$\begin{array}{lllll} \min_{\mathbf{w},b,\rho,\xi_i} & \frac{1}{2}\mathbf{w}^T\mathbf{w} - \nu\rho & + & \frac{1}{n}\sum_{i=1}^{n}\xi_i & \\ \text{subject to} & y_i(\mathbf{w}^T\mathbf{p}_i - b) & \geq & \rho - \xi_i, & i = 1, 2, \ldots, n \\ & \xi_i & \geq & 0, & i = 1, 2, \ldots, n \\ & \rho & \geq & 0 & \end{array}$$

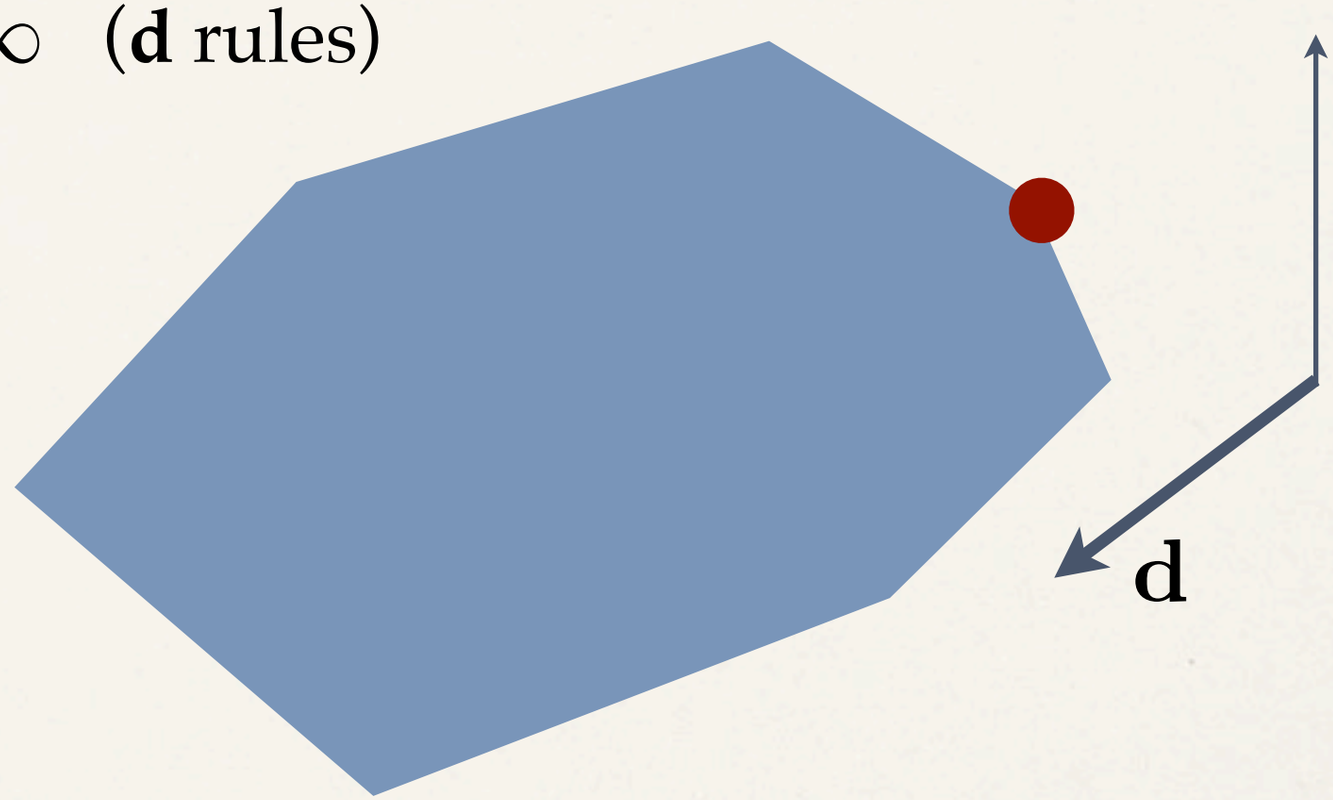
- **Parametric linear program:**

$$\begin{array}{lll} \min & (\mathbf{c}^T + \nu\mathbf{d})^T\mathbf{x} & \\ \text{subject to} & A\mathbf{x} = b & 0 \leq \nu \leq \infty \\ & \mathbf{x} \geq 0 & \end{array}$$

Parametric Linear Programs
The Solution Path

$$\min\ (\mathbf{c}^T + \nu\mathbf{d})^T\mathbf{x}$$

- Start with $\nu = \infty$ (**d** rules)

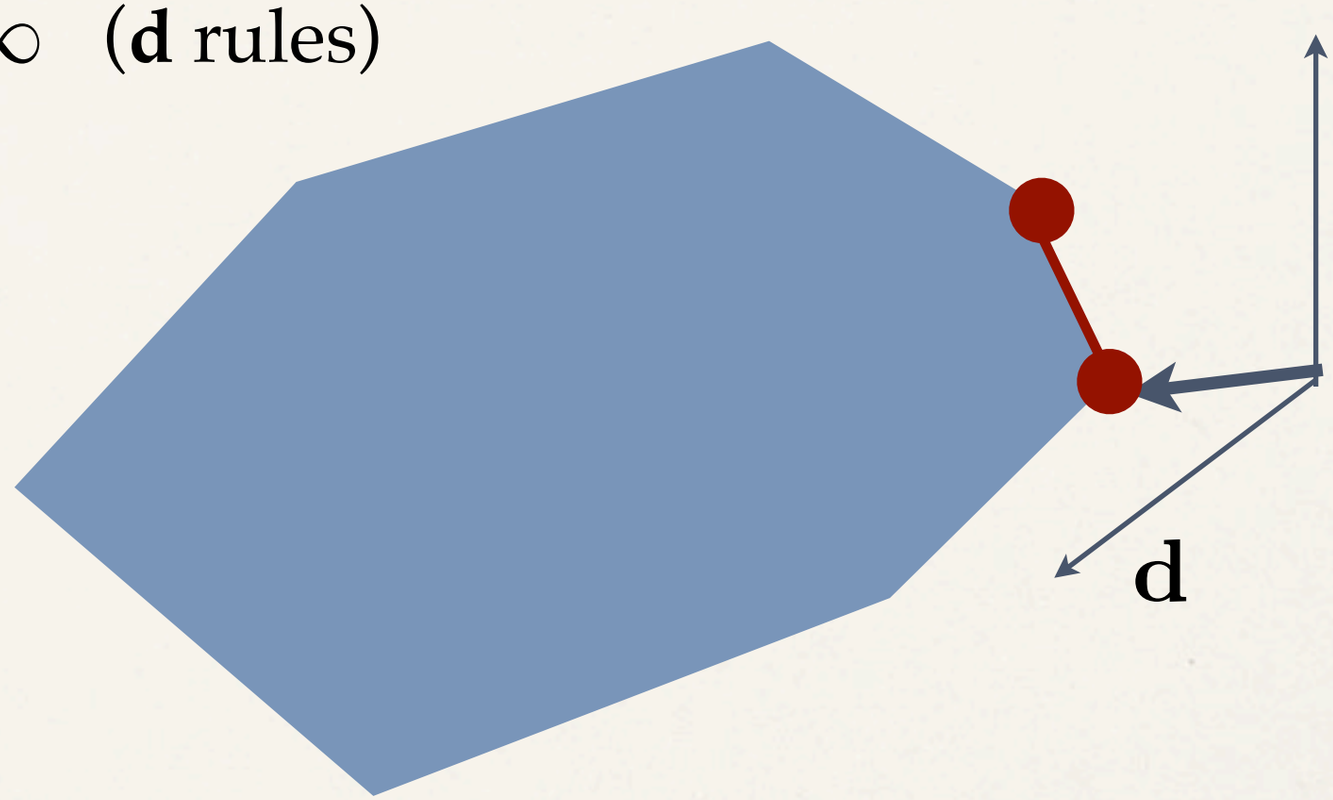


● = opt

Parametric Linear Programs
The Solution Path

$$\min\ (\mathbf{c}^T + \nu\mathbf{d})^T\mathbf{x}$$

- Start with $\nu = \infty$ (**d** rules)
- Decrease ν...

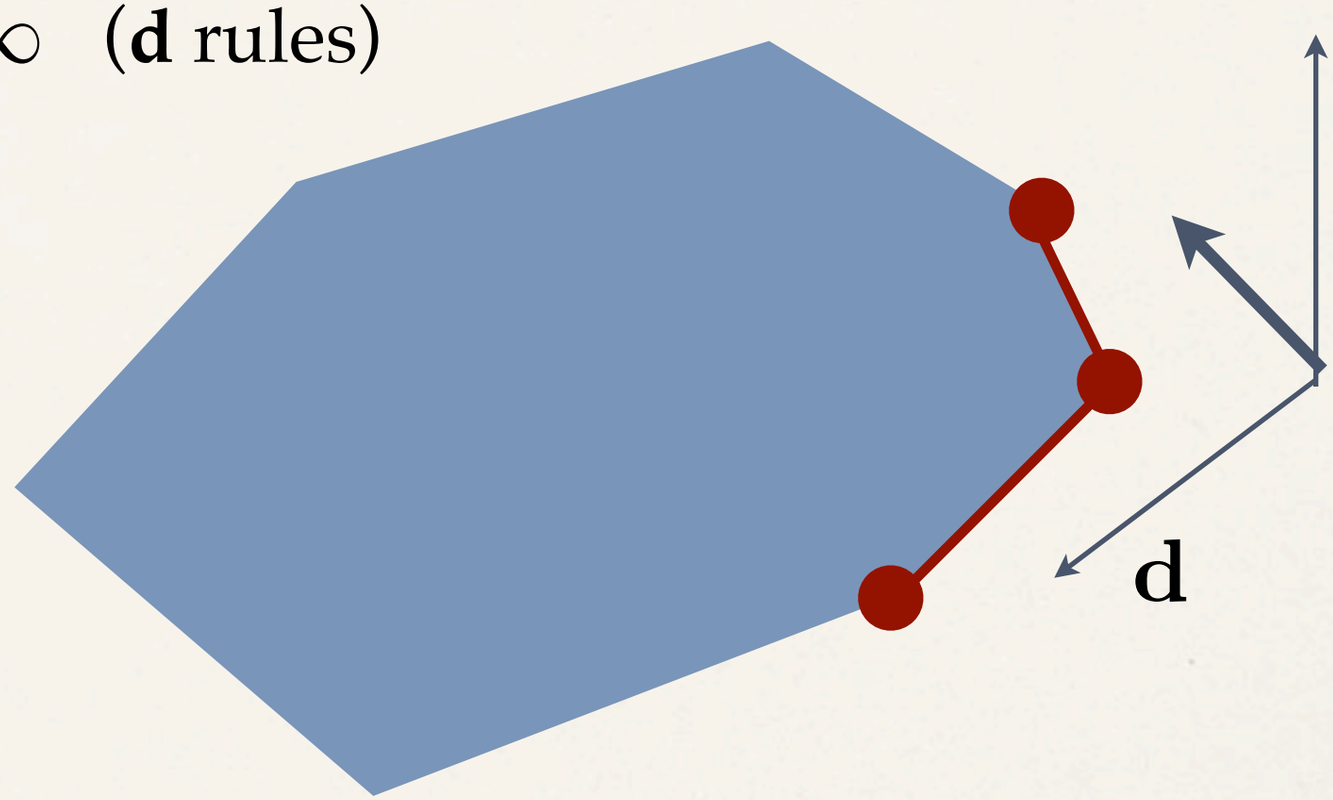


● = opt

Parametric Linear Programs
The Solution Path

$$\min\ (\mathbf{c}^T + \nu\mathbf{d})^T\mathbf{x}$$

- Start with $\nu = \infty$ (**d** rules)
- Decrease ν...

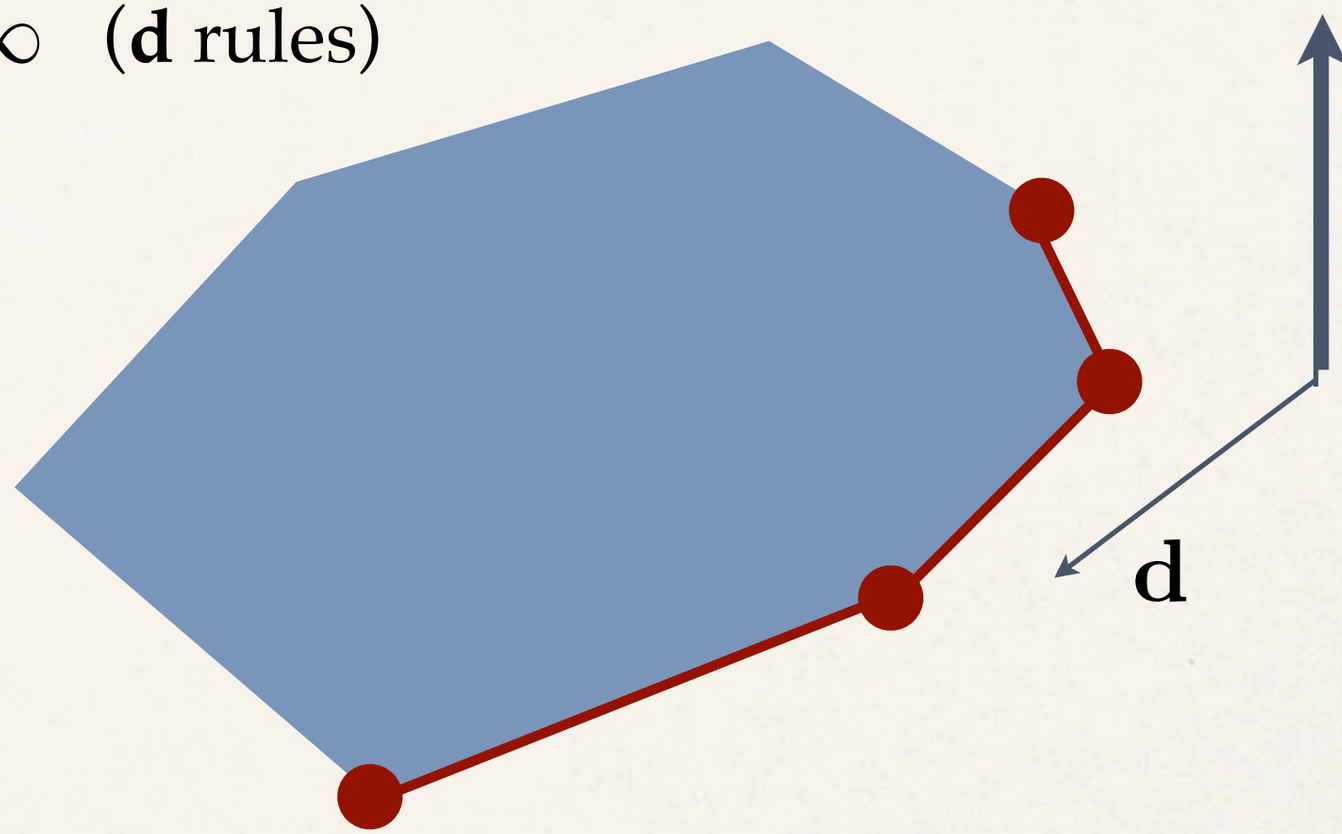


● = opt

Parametric Linear Programs
The Solution Path

$$\min\ (\mathbf{c}^T + \nu\mathbf{d})^T\mathbf{x}$$

- Start with $\nu = \infty$ (**d** rules)
- Decrease ν...
- until $\nu = 0$

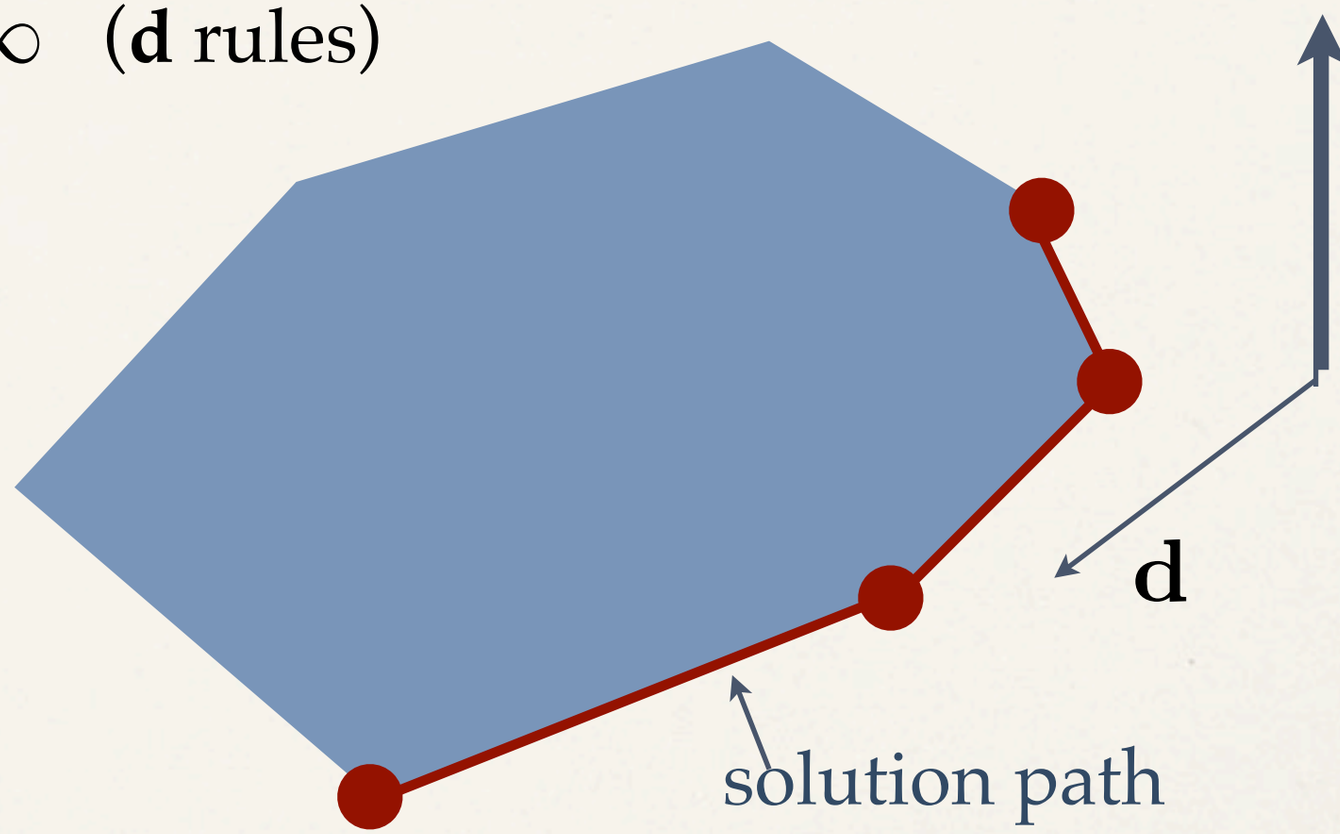


● = opt

Parametric Linear Programs
The Solution Path

$$\min\ (\mathbf{c}^T + \nu\mathbf{d})^T\mathbf{x}$$

- Start with $\nu = \infty$ (**d** rules)
- Decrease ν...
- until $\nu = 0$

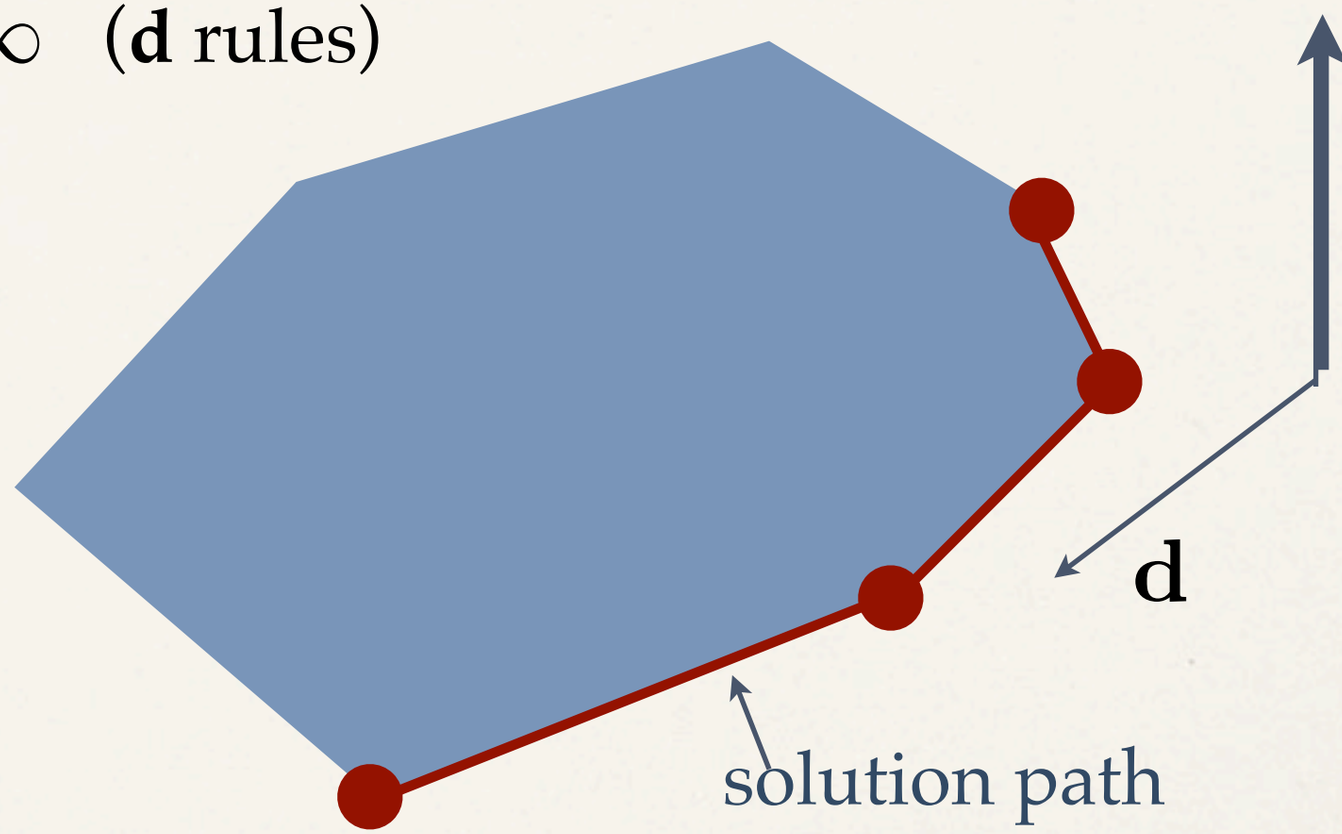


● = opt

Parametric Linear Programs
The Solution Path

$$\min\ (\mathbf{c}^T + \nu\mathbf{d})^T\mathbf{x}$$

- Start with $\nu = \infty$ (**d** rules)
- Decrease ν...
- until $\nu = 0$

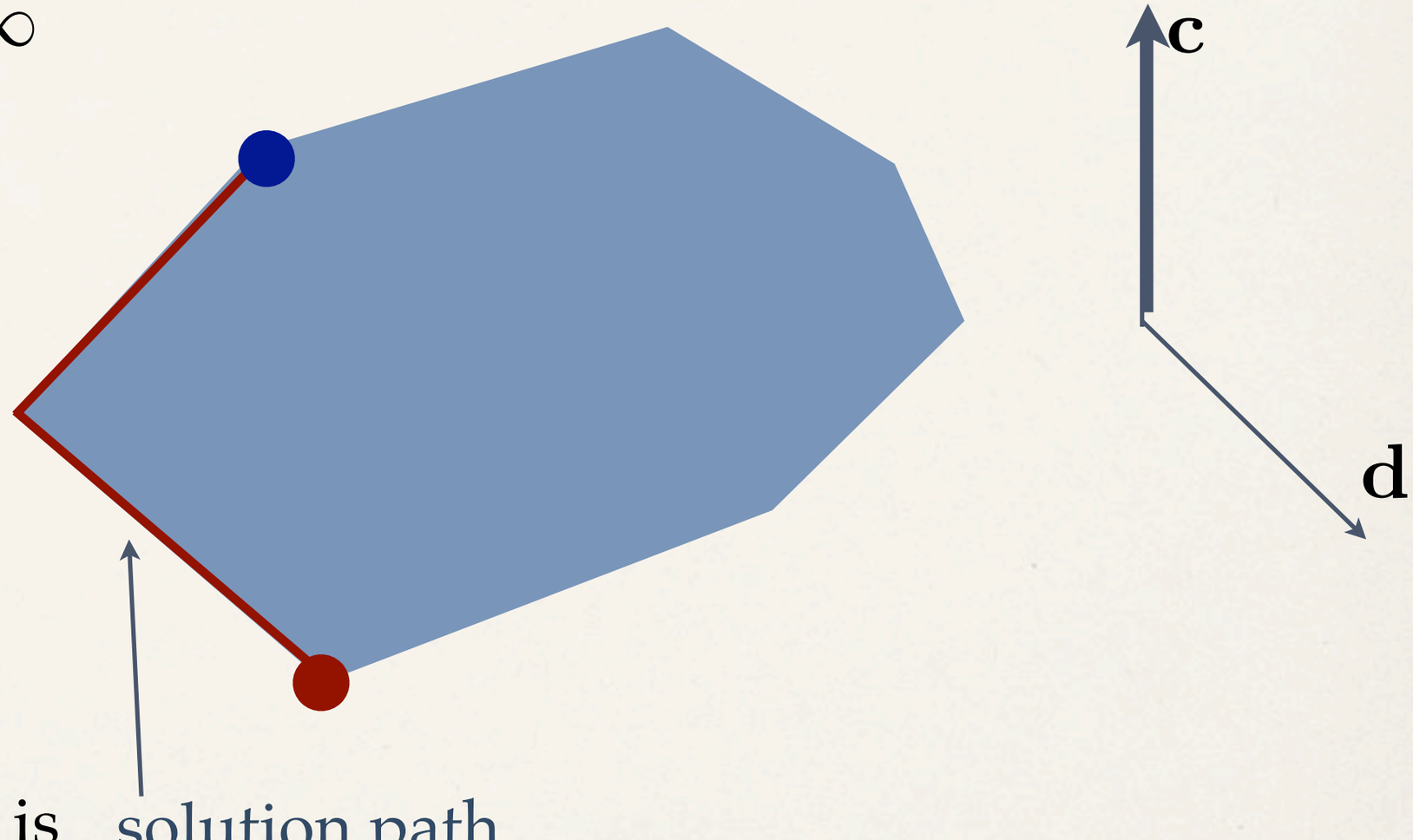


● = opt

- Same picture in higher *d*, with blue polygon the *shadow* of the LP

The Shadow Vertex / Parametric Objective Simplex Algorithm $\min \mathbf{c}^T\mathbf{x}$

- Start with $\nu = \infty$ (and artificial $\mathbf{d}$ such that start vertex ● is optimal)
- Decrease ν...
- until $\nu = 0$
- Vertex ● obtained is optimal under $\mathbf{c}$

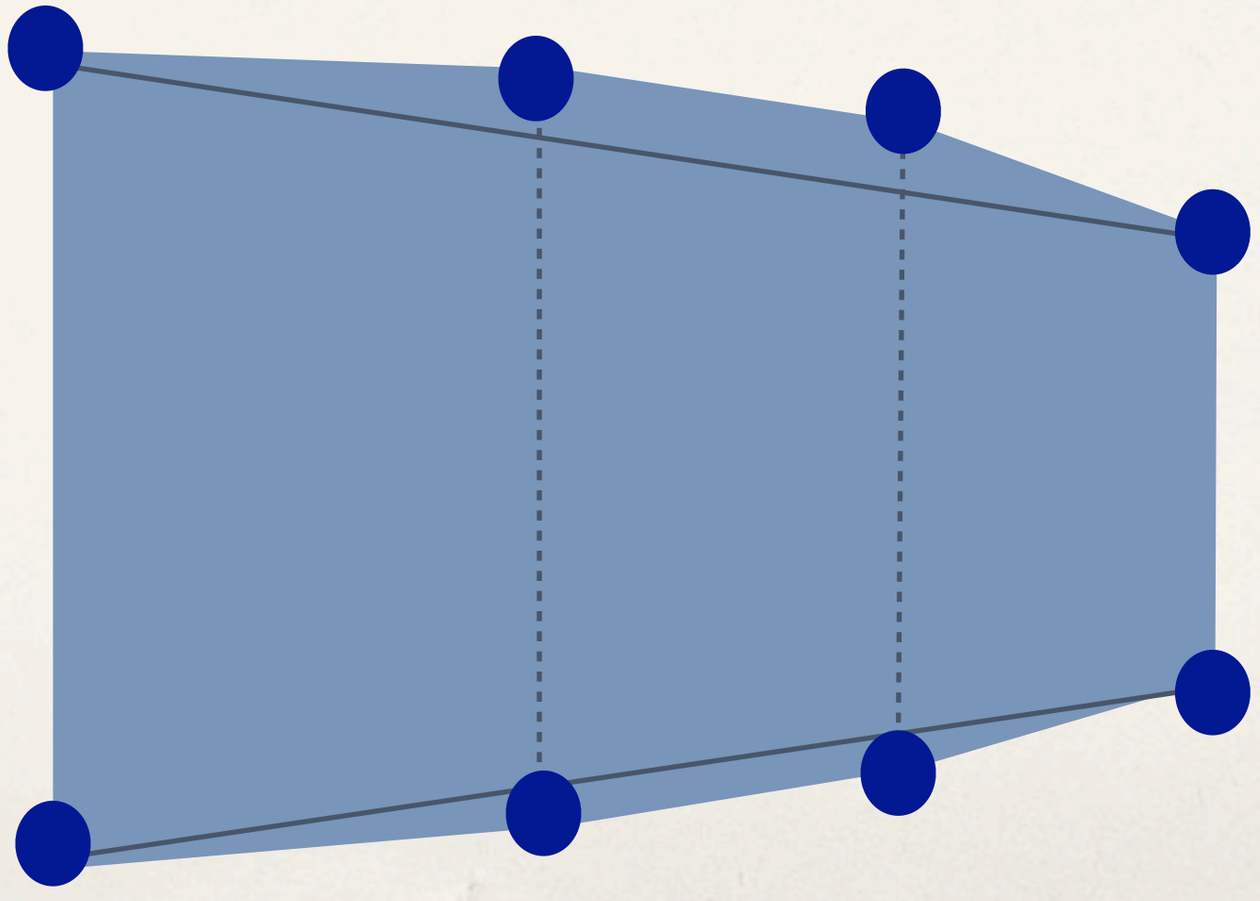


The Goldfarb Cubes

- Worst case inputs for the shadow vertex simplex algorithm
- The d-dimensional Goldfarb cube is a deformed unit cube with all its 2^d vertices appearing on a 2-dimensional shadow

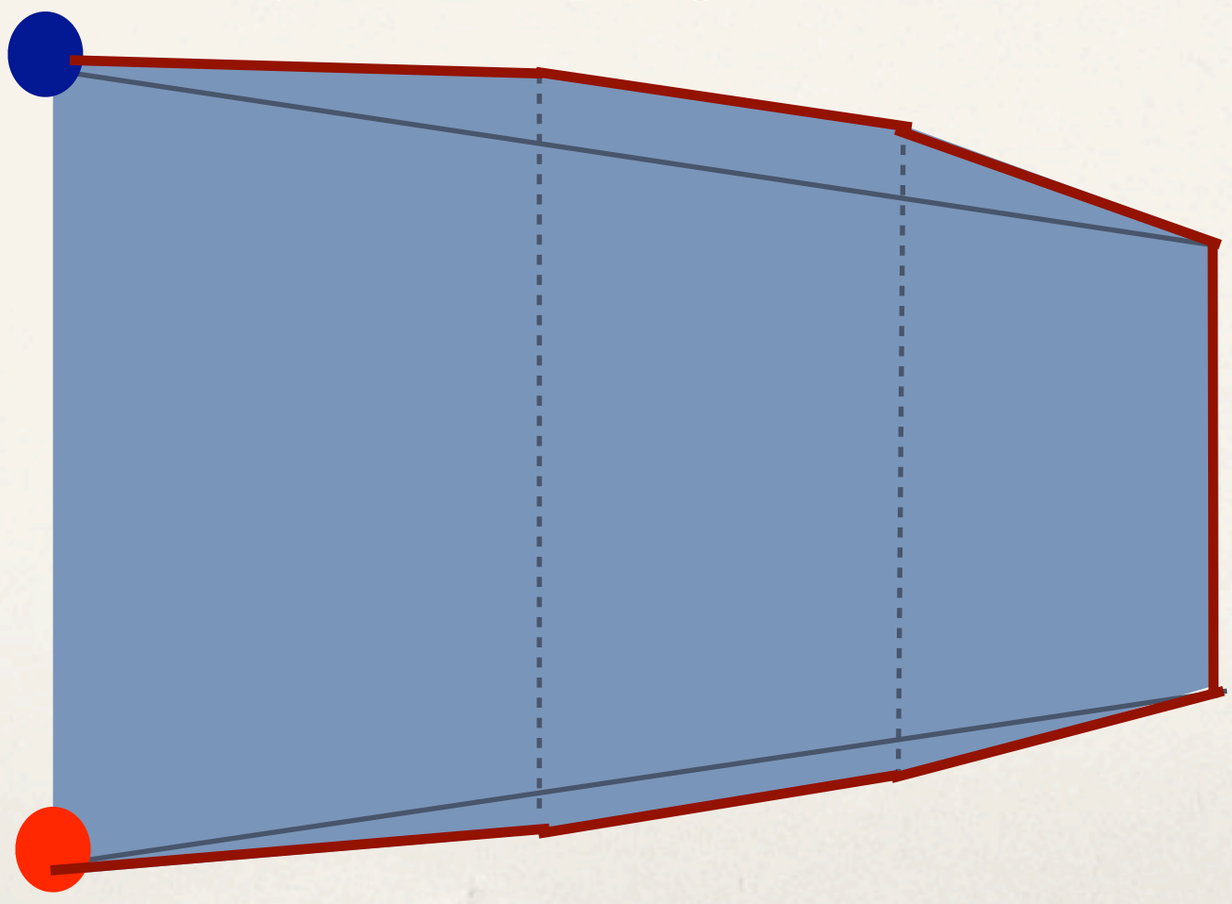
The Goldfarb Cubes

- Worst case inputs for the shadow vertex simplex algorithm
- The d-dimensional Goldfarb cube is a deformed unit cube with all its 2^d vertices appearing on a 2-dimensional shadow



Solution Paths of Parametric Linear Programs...

- can be exponentially long in the worst case
- It is not implausible that a similar result holds for the solution path of SVM (specific parametric *quadratic* programs)



Solution Paths of Parametric Linear Programs...

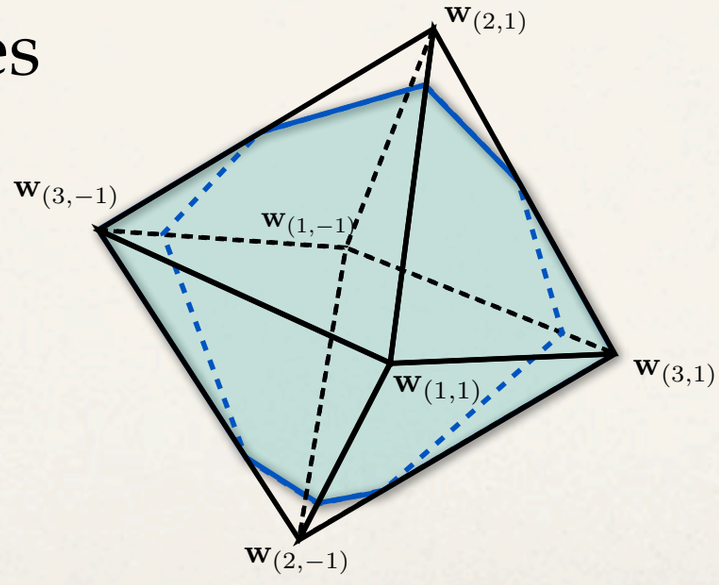
- can be exponentially long in the worst case
- It is not implausible that a similar result holds for the solution path of SVM (specific parametric *quadratic* programs)
- **Obstacles:**
  - SVM are very special parametric quadratic programs
  - Goldfarb cube is not an instance of an SVM
  - we want a *nondegenerate* SVM with a long solution path

An SVM with an Exponentially Long Solution Path

- **Outline:**
  - Work in the dual (μ-SVM: distance between reduced convex hulls)

An SVM with an Exponentially Long Solution Path

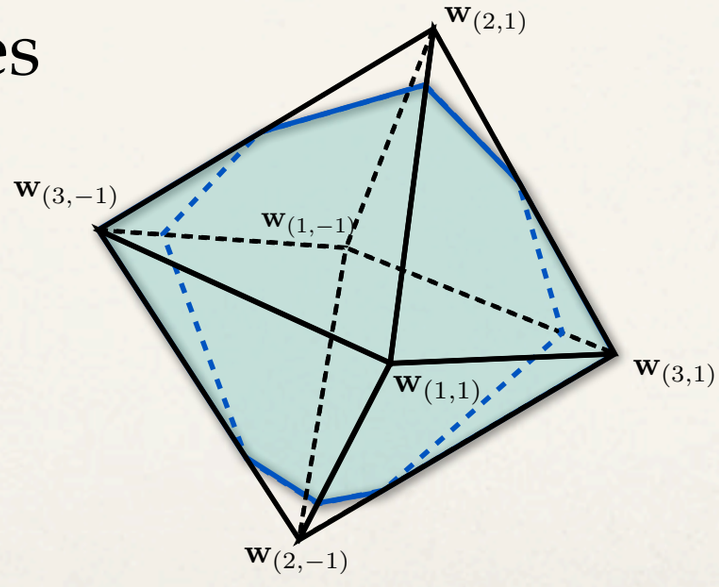
- **Outline:**
  - Work in the dual (μ-SVM: distance between reduced convex hulls)
  - First convex hull: **dual Goldfarb cube** ($2d$ points); there exists a $2d$ plane S such that the dual Goldfarb cube intersects S in a polygon with 2^d edges



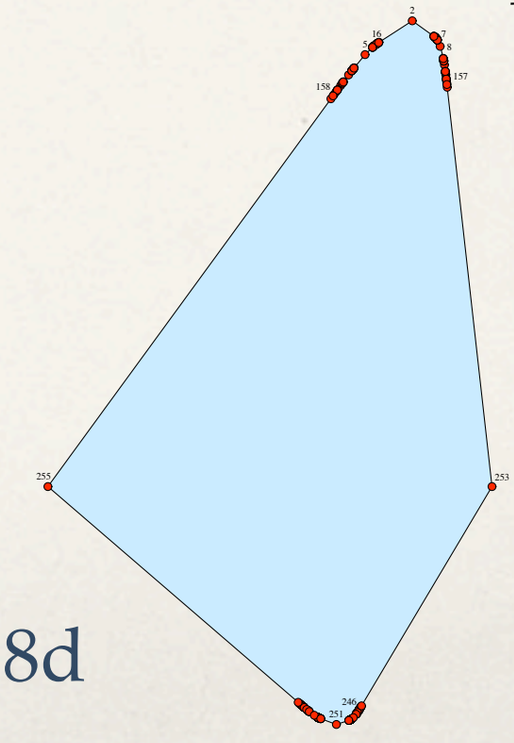
3d

An SVM with an Exponentially Long Solution Path

- **Outline:**
  - Work in the dual (μ-SVM: distance between reduced convex hulls)
  - First convex hull: **dual Goldfarb cube** ($2d$ points); there exists a $2d$ plane S such that the dual Goldfarb cube intersects S in a polygon with 2^d edges



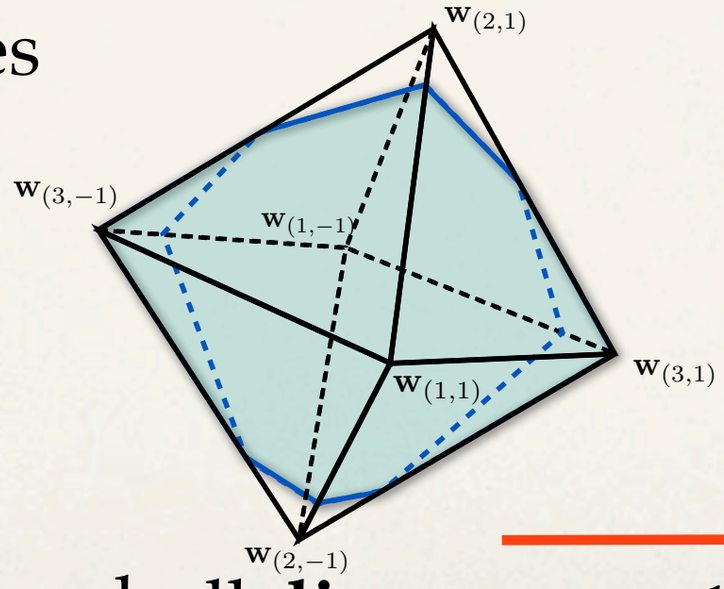
3d



8d

An SVM with an Exponentially Long Solution Path

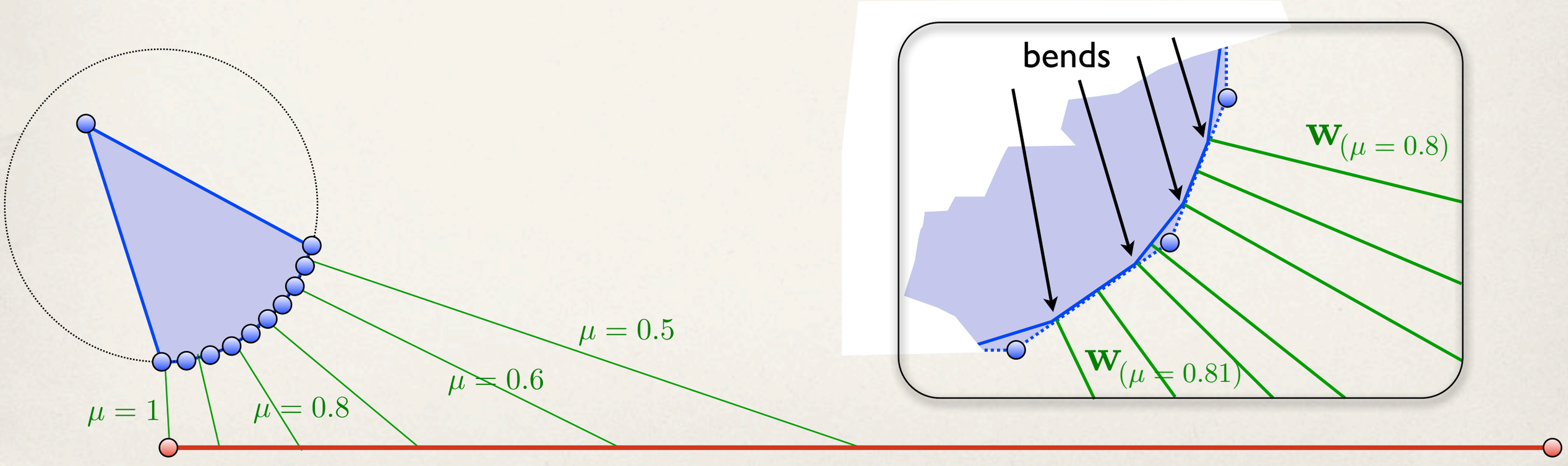
- **Outline:**
  - Work in the dual (μ-SVM: distance between reduced convex hulls)
  - First convex hull: **dual Goldfarb cube** ($2d$ points); there exists a $2d$ plane S such that the dual Goldfarb cube intersects S in a polygon with 2^d edges



  - Second convex hull: **line segment in S**

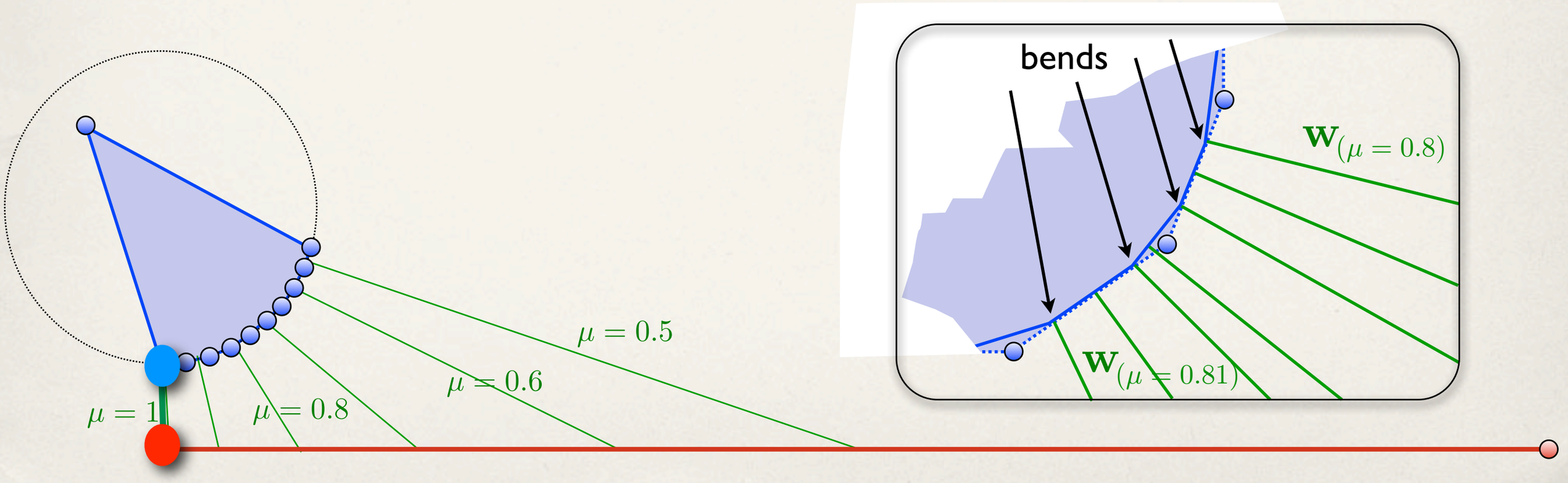
An SVM with an Exponentially Long Solution Path

- **Outline:**
  - While μ decreases, the left segment endpoint moves to the right (reduction!). The shortest distance vector traverses 2^d / 4 many edges of the dual Goldfarb cube intersection with S



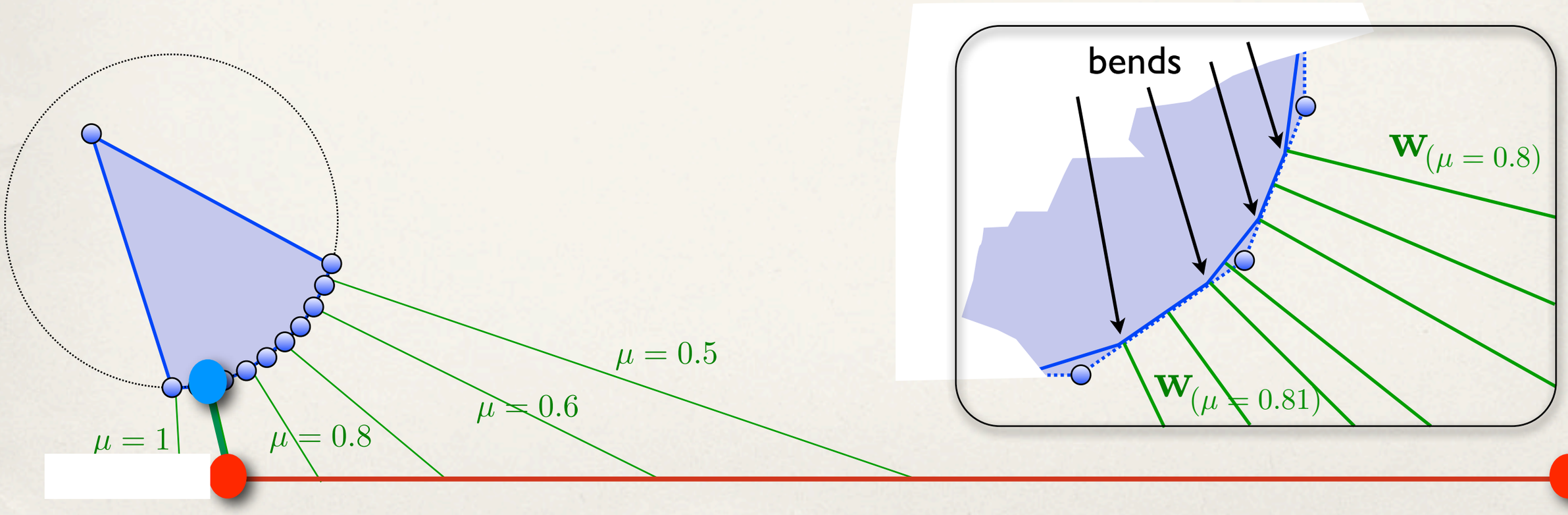
An SVM with an Exponentially Long Solution Path

- **Outline:**
  - While μ decreases, the left segment endpoint moves to the right (reduction!). The shortest distance vector traverses 2^d / 4 many edges of the dual Goldfarb cube intersection with S



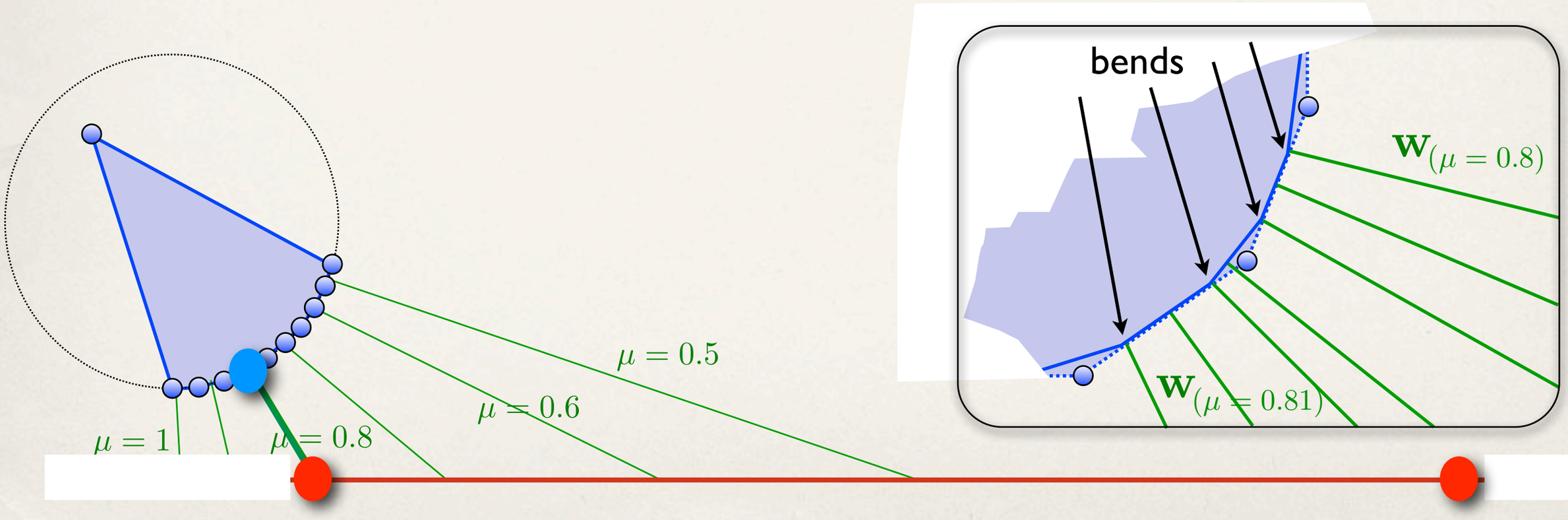
An SVM with an Exponentially Long Solution Path

- **Outline:**
  - While μ decreases, the left segment endpoint moves to the right (reduction!). The shortest distance vector traverses 2^d / 4 many edges of the dual Goldfarb cube intersection with S



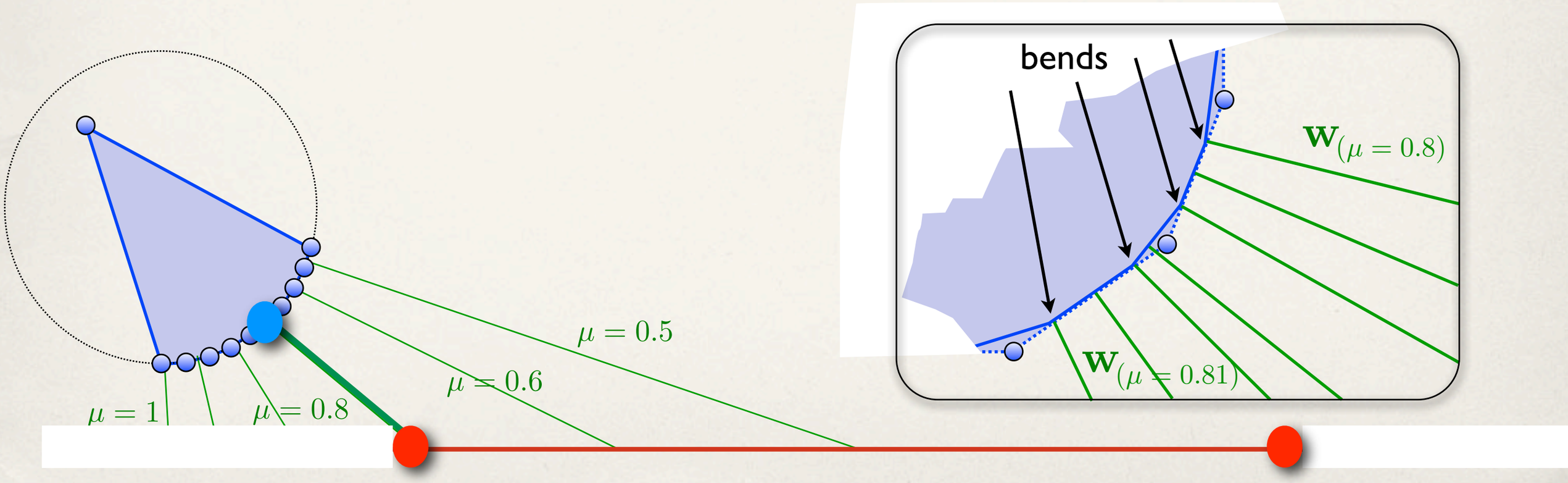
An SVM with an Exponentially Long Solution Path

- **Outline:**
  - While μ decreases, the left segment endpoint moves to the right (reduction!). The shortest distance vector traverses 2^d / 4 many edges of the dual Goldfarb cube intersection with S



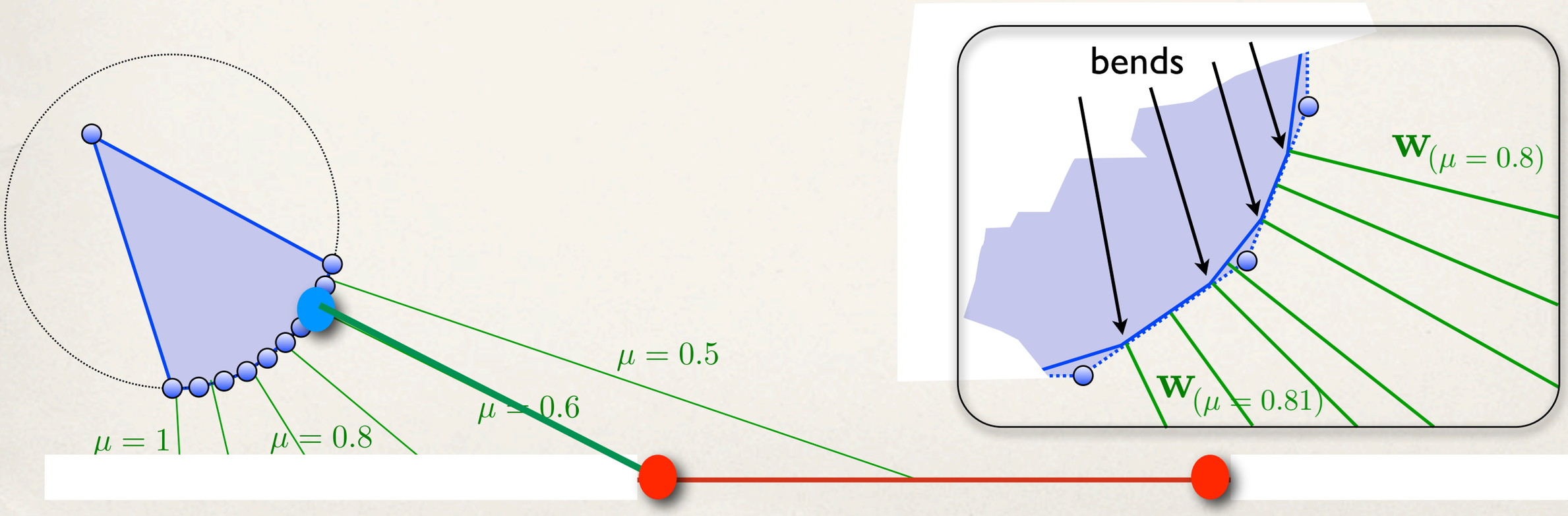
An SVM with an Exponentially Long Solution Path

- **Outline:**
  - While μ decreases, the left segment endpoint moves to the right (reduction!). The shortest distance vector traverses 2^d / 4 many edges of the dual Goldfarb cube intersection with S



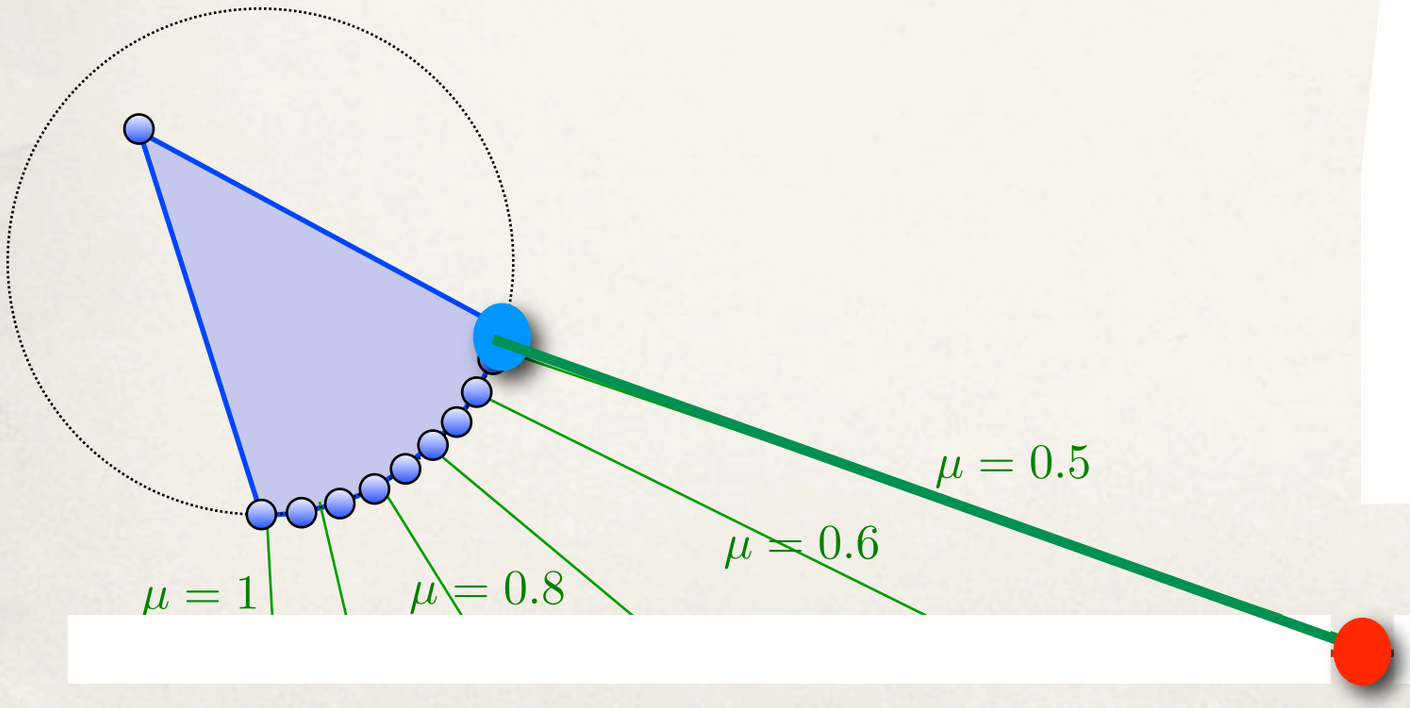
An SVM with an Exponentially Long Solution Path

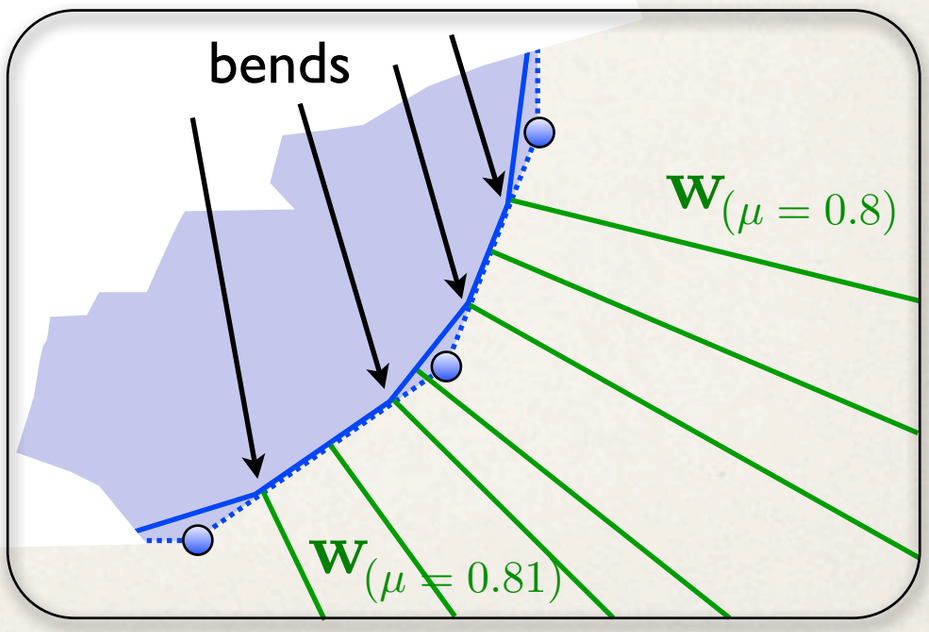
- **Outline:**
  - While μ decreases, the left segment endpoint moves to the right (reduction!). The shortest distance vector traverses 2^d / 4 many edges of the dual Goldfarb cube intersection with S



An SVM with an Exponentially Long Solution Path

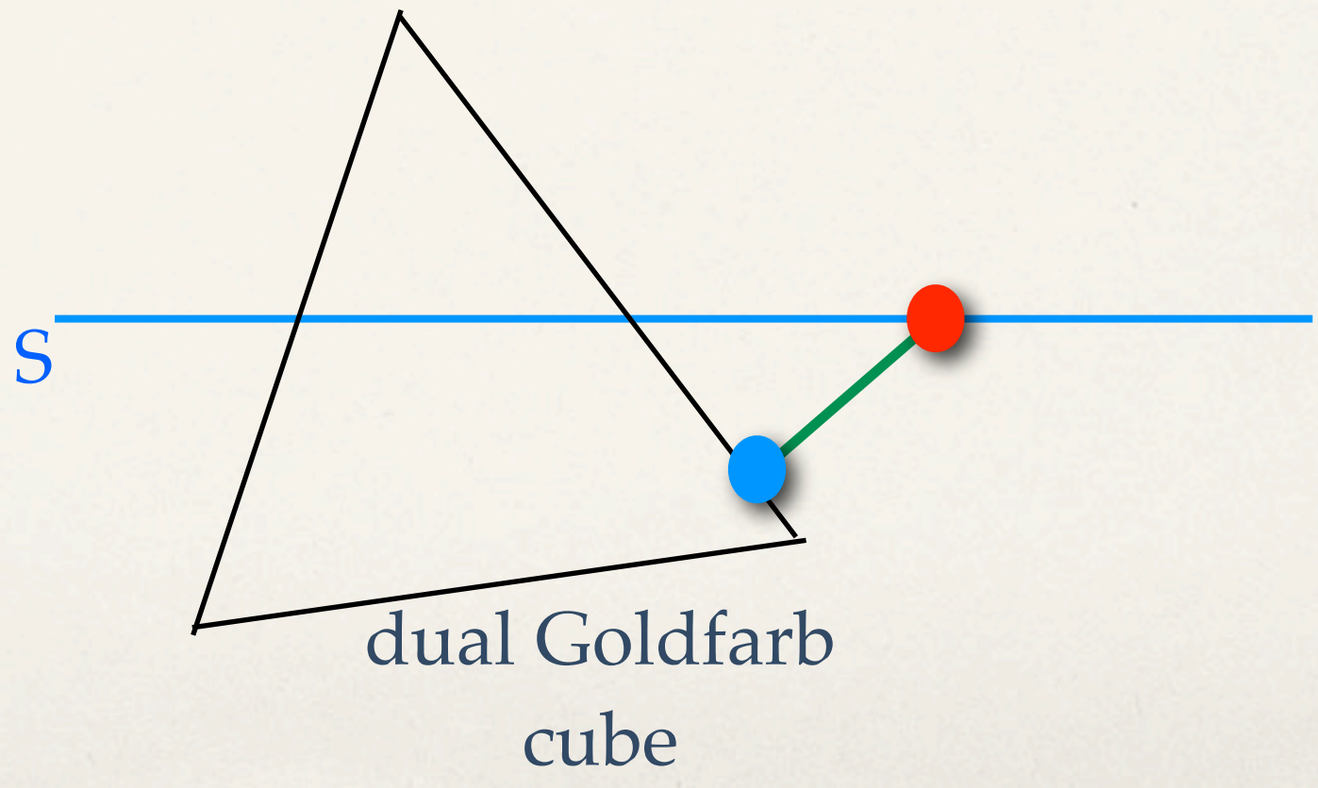
- **Outline:**
  - While μ decreases, the left segment endpoint moves to the right (reduction!). The shortest distance vector traverses 2^d / 4 many edges of the dual Goldfarb cube intersection with S





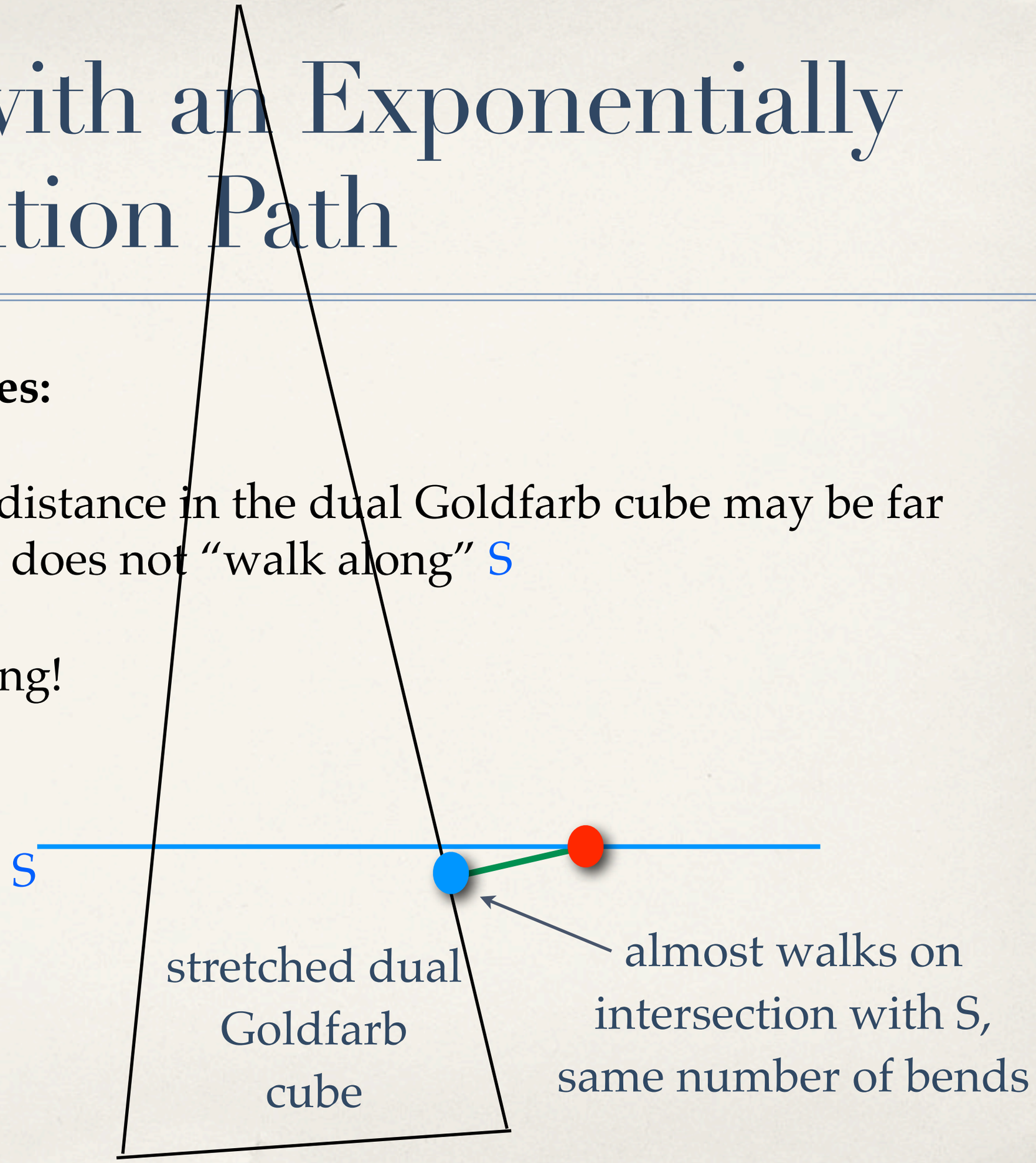
An SVM with an Exponentially Long Solution Path

- **Technical difficulties:**
  - Point of optimal distance in the dual Goldfarb cube may be far away from S and does not "walk along" S



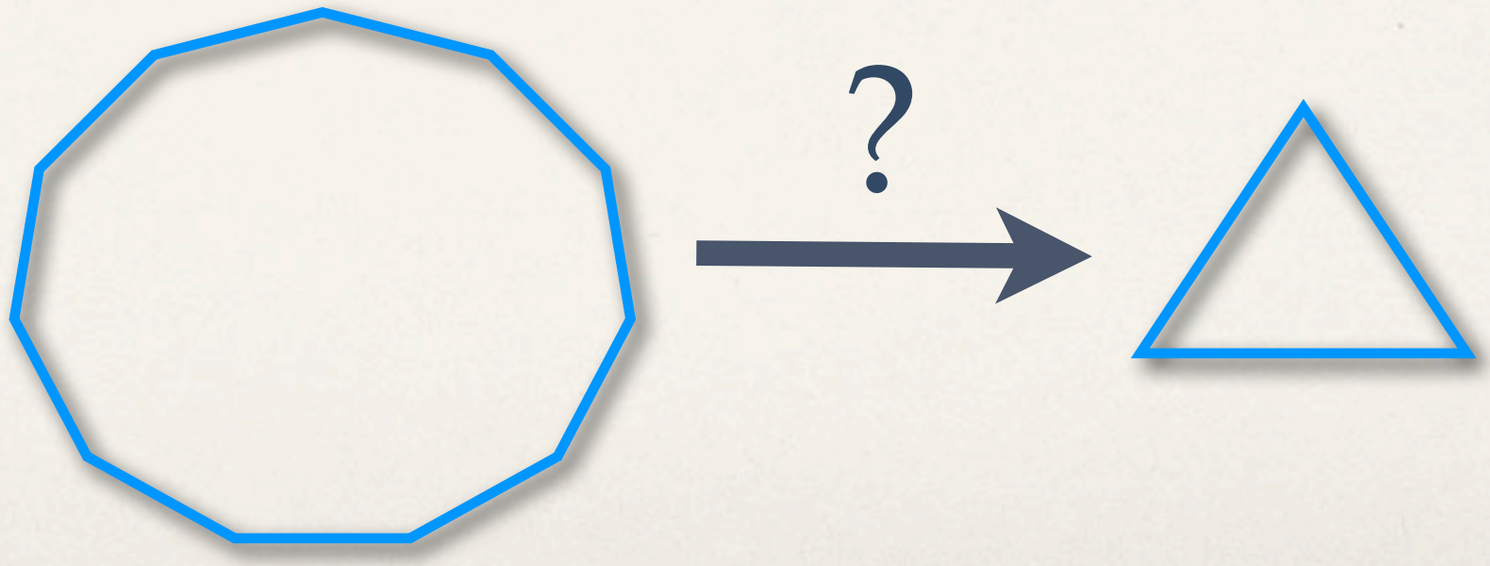
An SVM with an Exponentially Long Solution Path

- **Technical difficulties:**
  - Point of optimal distance in the dual Goldfarb cube may be far away from S and does not "walk along" S
- Solution: stretching!



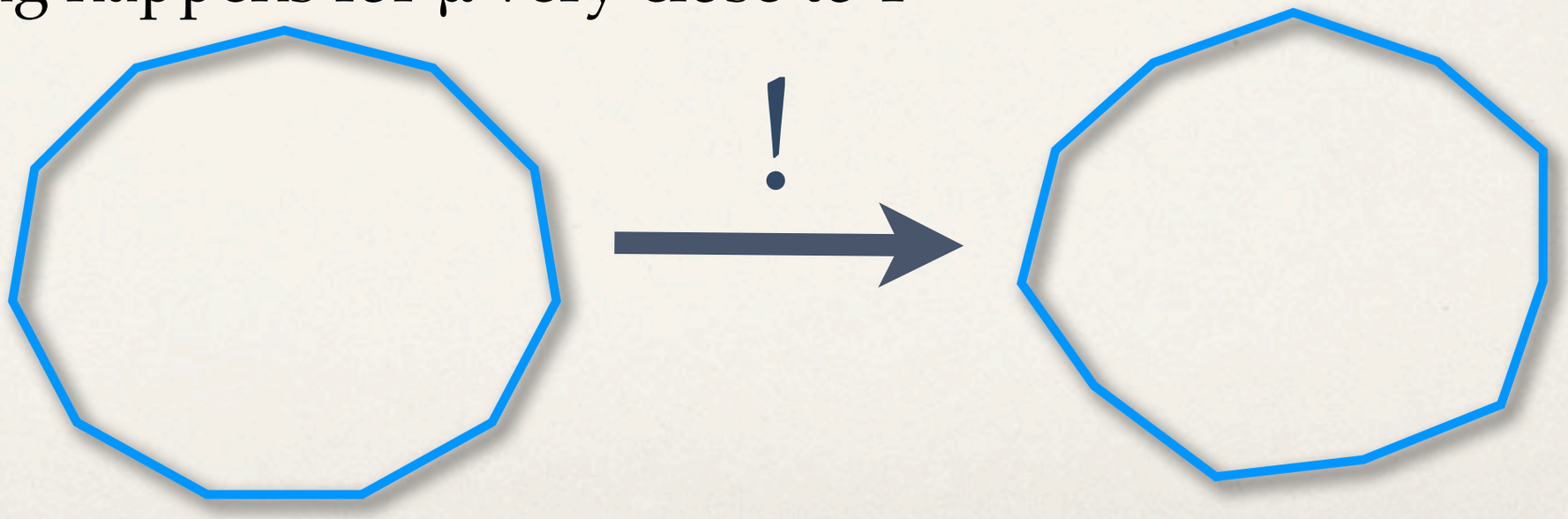
An SVM with an Exponentially Long Solution Path

- **Technical difficulties:**
  - Reduction of stretched dual Goldfarb cube may affect the intersection with S



An SVM with an Exponentially Long Solution Path

- **Technical difficulties:**
  - Reduction of stretched dual Goldfarb cube may affect the intersection with S
  - Solution: make line segment ——— so long that everything happens for μ very close to 1



Summary

- The solution path of a support vector machine may be exponentially long in the number of training points; the linear-length conjecture of Hastie et al. fails dramatically

Summary

- The solution path of a support vector machine may be exponentially long in the number of training points; the linear-length conjecture of Hastie et al. fails dramatically

- For “us” polytope people, this result is what we would expect, knowing the exponential lower bounds for solution paths of parametric *linear* programs (Goldfarb cubes)

Summary

- The solution path of a support vector machine may be exponentially long in the number of training points; the linear-length conjecture of Hastie et al. fails dramatically

- For "us" polytope people, this result is what we would expect, knowing the exponential lower bounds for solution paths of parametric *linear* programs (Goldfarb cubes)

- *Approximate* solution paths are always short (Giesen, Jaggi, Laue)

Summary

- The solution path of a support vector machine may be exponentially long in the number of training points; the linear-length conjecture of Hastie et al. fails dramatically

- For “us” polytope people, this result is what we would expect, knowing the exponential lower bounds for solution paths of parametric *linear* programs (Goldfarb cubes)

- *Approximate* solution paths are always short (Giesen, Jaggi, Laue)

- The machine learning community is proficient in statistics, but not in geometry; it can be beneficial to look at some of their problems from a geometric viewpoint