Optimization over Zonotopes and Training Support Vector Machines

Marshall Bern

Xerox Palo Alto Research Ctr.

David Eppstein

Univ. of California, Irvine
Dept. of Information and Computer Science

Support Vector Machines (SVM)

Machine learning technique for classification problems
i.e. given a large number of labeled yes/no instances,
predict yes/no value of additional instances

Lift data values to moderate- or high-dimensional Euclidean space
may be implicit, using "kernel functions" to replace dot products

Find hyperplane separating lifted yes and no instances
depending on only few "support vectors"

Predict future values by lifting and using same hyperplane

Support Vector Machines (SVM)

Machine learning technique for classification problems
i.e. given a large number of labeled yes/no instances,
predict yes/no value of additional instances

Lift data values to moderate- or high-dimensional Euclidean space
may be implicit, using "kernel functions" to replace dot products

Find hyperplane separating lifted yes and no instances
depending on only few "support vectors"

Predict future values by lifting and using same hyperplane

Mathematical optimization problem
Using linear or convex programming algorithms

Directions of SVM Research

Apply SVM techniques to machine learning applications

Compare SVM techniques to other classifiers

Modify SVM to produce better classifiers

Derive efficient practical algorithms for SVM optimization

Do theoretical analysis of hyperplane separation algorithms

Directions of SVM Research

Apply SVM techniques to machine learning applications

Compare SVM techniques to other classifiers

Modify SVM to produce better classifiers

Derive efficient practical algorithms for SVM optimization

Do theoretical analysis of hyperplane separation algorithms

Our interests

Directions of SVM Research

Apply SVM techniques to machine learning applications

Compare SVM techniques to other classifiers

Modify SVM to produce better classifiers

Derive efficient practical algorithms for SVM optimization

Do theoretical analysis of hyperplane separation algorithms

This talk

Isn't it just linear programming?

find $\mathbf{v}$, c defining separating hyperplane $\mathbf{v} \cdot \mathbf{x} + c = 0$
satisfying constraints $\mathbf{v} \cdot \mathbf{Y_i} + c \geq 0$, for yes-instances,
$\mathbf{v} \cdot \mathbf{N_i} + c \leq 0$ for no-instances

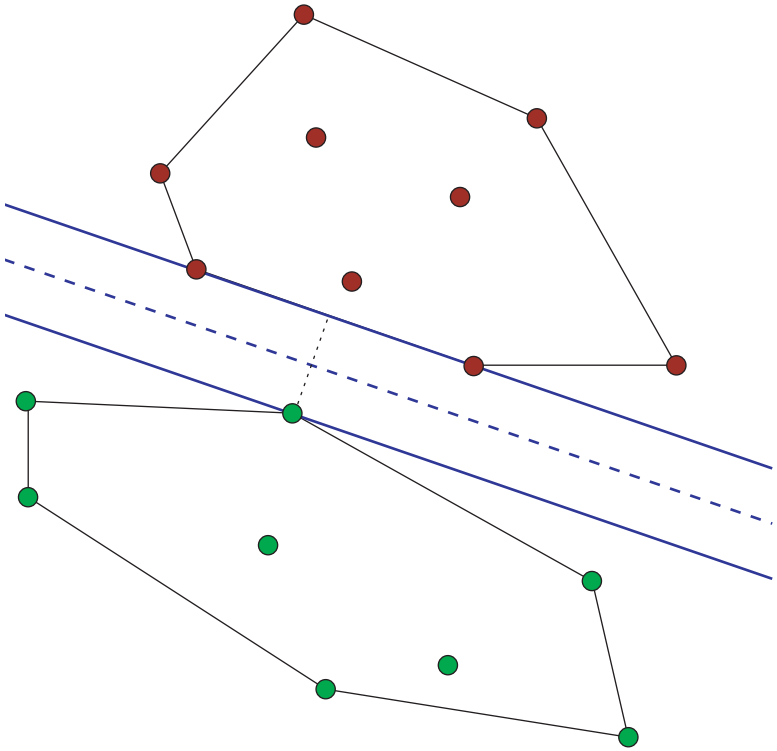
From computational geometry we know LP is efficient when $n >> d$

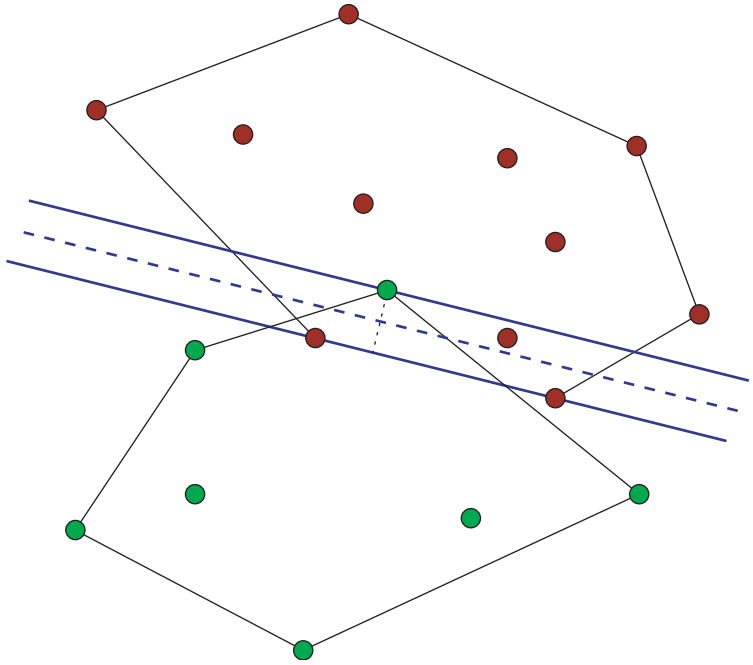
No, because...

Many feasible solutions, need to choose one
"maximum margin classifier" leads to quadratic program, still not so hard

Use "soft margin classifier" to avoid dependence on outliers
blows up dimension from d to $n + d$ if expressed as LP
so want algorithms that stay in low dimension

Maximum margin classifier





Choose hyperplane at maximum distance from both convex hulls

Works well (but so do many other choices) when sets well-separated

When sets overlap, distance from hulls is negative
Maximum margin unpopular in this case
due to sensitive dependence on the most extreme points (outliers)

Soft Convex Hull

Idea: shrink the two convex hulls so they are well separated

Usual hull: sum $a_i p_i$, $0 \le a_i \le 1$, sum $a_i = 1$

Centroid: sum $a_i p_i$, $0 \le a_i \le 1/n$, sum $a_i = 1$

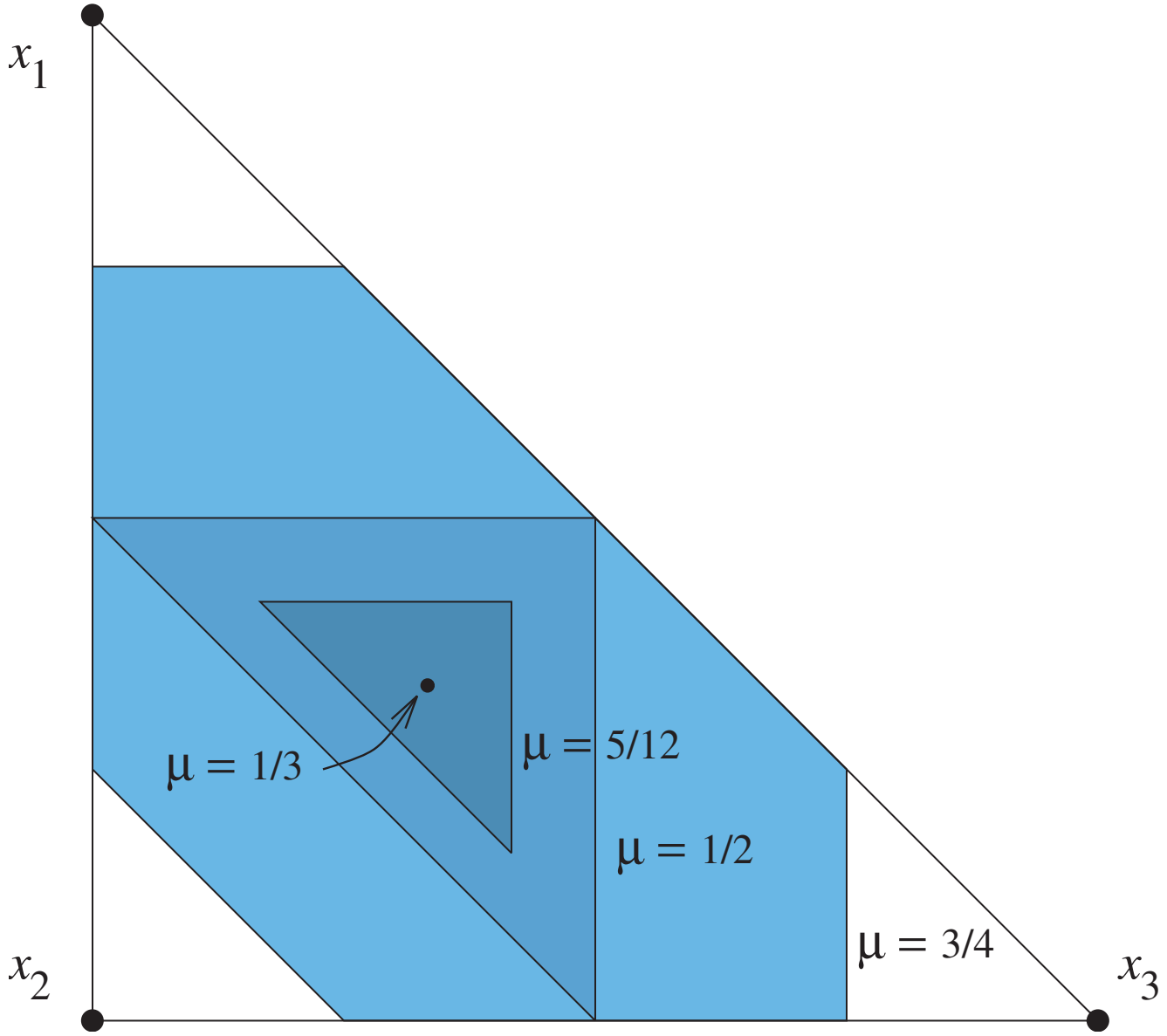
Soft convex hull: sum $a_i p_i$, $0 \le a_i \le \mu$, sum $a_i = 1$

Choose parameter $1/n \le \mu \le 1$ to shrink hull towards centroid

Result is a "centroid polytope" [Bern et al., ESA '95]:
weighted average of points where weights vary in interval $[0,\mu]$

Formed by intersecting zonotope sum $a_i p_i$, $0 \le a_i \le \mu$
with hyperplane sum $a_i = 1$

Soft Convex Hulls



Soft Margin Classifiers

If μ is large, optimal separating hyperplane depends only on few "support vectors" rather than on entire data set

If μ is small, soft hulls will be well separated

Choose μ automatically to largest value for which hulls are separated

Geometrically: find lowest point in intersection of two zonotopes

or...

Choose μ empirically (e.g. by cross-validating to find best classifier)

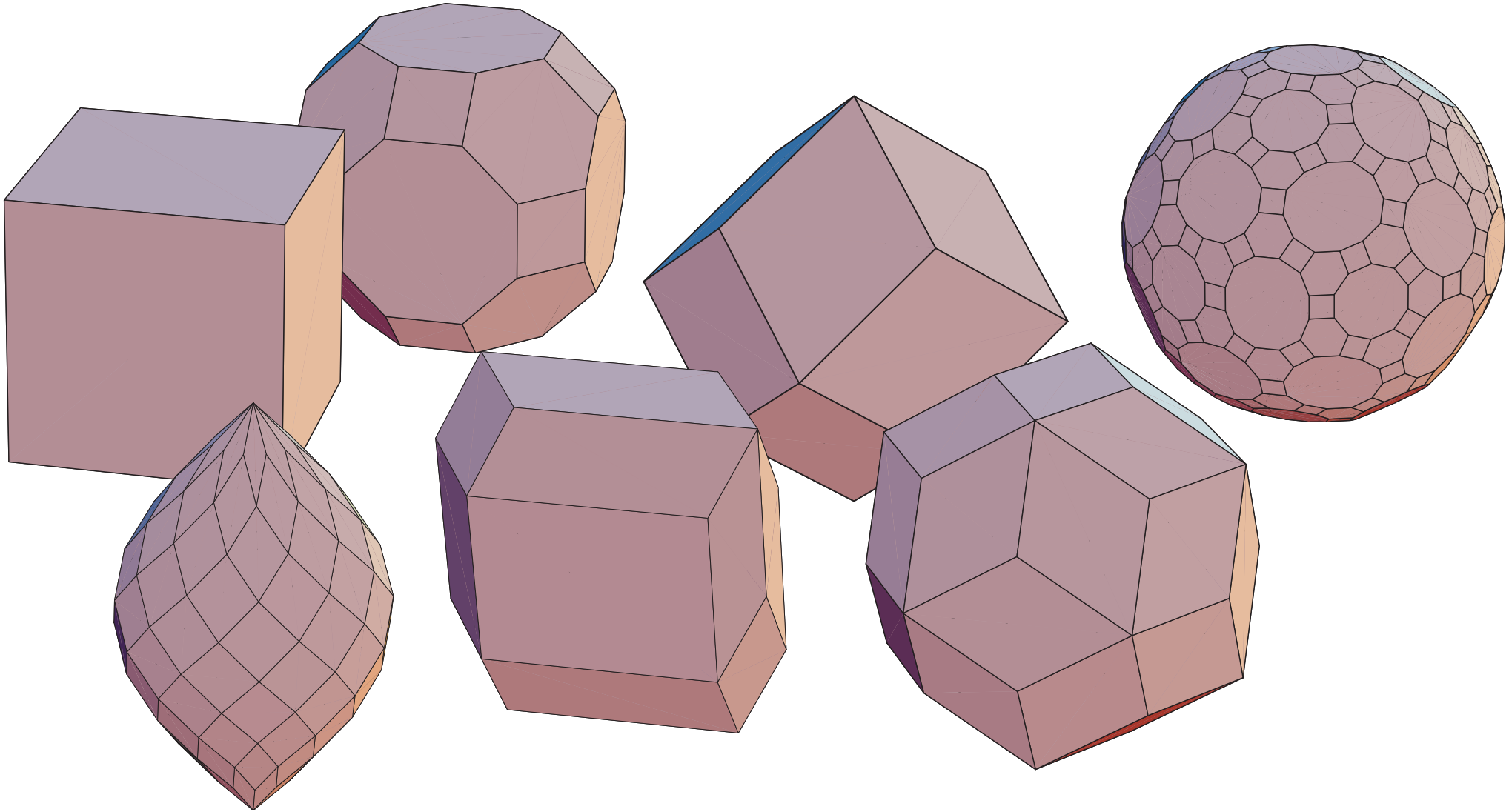
Perform maximum margin classification for chosen value

Geometrically: find closest points in two zonotope cross-sections

Our techniques apply to both problems

Zonotopes: Minkowski sums of line segments

Choose one point from each segment, add the coordinates



Typically $\Theta(n^{d-1})$ facets

corresponding to hyperplane arrangement in $d-1$ dimensions

Optimization over Zonotopes

Given a collection of zonotopes generated by n line segments

and given a linear objective function f

find the point $\mathbf{x}$ in the intersection of the zonotopes minimizing $f(\mathbf{x})$

Like linear programming with zonotope instead of halfspace constraints

Could be turned into an explicit LP but number of constraints blows up

This solves automatic choice of μ, fixed-μ variant is similar

Goals:

scalable algorithm (linear or near-linear in n)

low dependence on d — typical CG alg. is exponential, we prefer polynomial

Optimization over one zonotope

Given zonotope and linear function, what is best vertex?

Very easy: optimize independently over each line segment

Zonotope intersect hyperplane almost as easy: fractional knapsack
(solved by a greedy algorithm)

But how to extend to more than one zonotope?

Ellipsoid Method

General technique for linear or convex optimization
Not very practical

Converts separation into optimization

Needs as input a "separation oracle"
that tests if a point is in feasible region,
if not finds hyperplane separating it from feasible region

Dually, converts optimization into separation

Separation on a convex set = optimization on its polar, vice versa

Can solve separation problem using as input an "optimization oracle"
that finds extreme vertex for a linear objective function

Zonotope optimization algorithm

Use ellipsoid to convert single-zonotope optimization to separation

Multi-zonotope separation solved by testing each zonotope independently

Use ellipsoid again to convert separation to multi-zonotope optimization

Analysis:

Two levels of recursive calls in ellipsoid methods

Each level multiplies time by poly(*d*, precision)

Required precision can be shown to be small: polylog(*n*) times initial precision

No blowup in dependence on n

Total time: O(*n* poly(*d*, log *n*, precision))

Conclusions

Can solve SVM optimization in time O(*n* polylog)

Scalable (near-linear dependence on *n*)
Polynomial dependence on *d*

Alternatives?

Typical computational geometry approach: parametric search

Converts decision problem into optimization, similarly to ellipsoid
so again need two levels of recursion

Seems to lead to O(n polylog), no dependence on precision
but exponential dependence on dimension

What about a practical polynomial time algorithm?