Support vector machines

Fraida Fund

Contents

In this lecture

- Maximal margin classifier
- Support vector classifier
- Solving constrained optimization to find coefficients
- Support vector machine with non-linear kernel

Recap

Classifying data that is not linearly separable

- Decision tree - complex decision boundary, fast prediction, often works best as part of ensemble
- KNN - complex decision boundary, slow prediction
- Logistic regression - only if you use basis function $\phi()$ to transform data before applying model

Maximal margin classifier

Binary classification problem

- N training samples $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^p$
- Class labels $y_1, \ldots, y_N \in \{-1, 1\}$

Linear separability

The problem is **perfectly linearly separable** if there exists a **separating hyperplane** H_i such that

- all $\mathbf{x} \in C_i$ lie on its positive side, and
- all $\mathbf{x} \in C_j, j \neq i$ lie on its negative side.

Separating hyperplane (1)

The separating hyperplane has the property that for all $i = 1, \ldots, N$,

$$\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} > 0 \text{ if } y_i = 1$$

$$\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} < 0 \text{ if } y_i = -1$$

Separating hyperplane (2)

Equivalently:

$$y_i \left(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} \right) > 0 \tag{1}$$

Using the hyperplane to classify

Then, we can classify a new sample $\mathbf{x}$ using the sign of

$$z = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}$$

and we can use the magnitude of z to determine how confident we are about our classification. (Larger z = farther from hyperplane = more confident about classification.)

Non-uniqueness

If a separating hyperplane exists, there will be an infinite number of separating hyperplanes.

Which separating hyperplane is best?

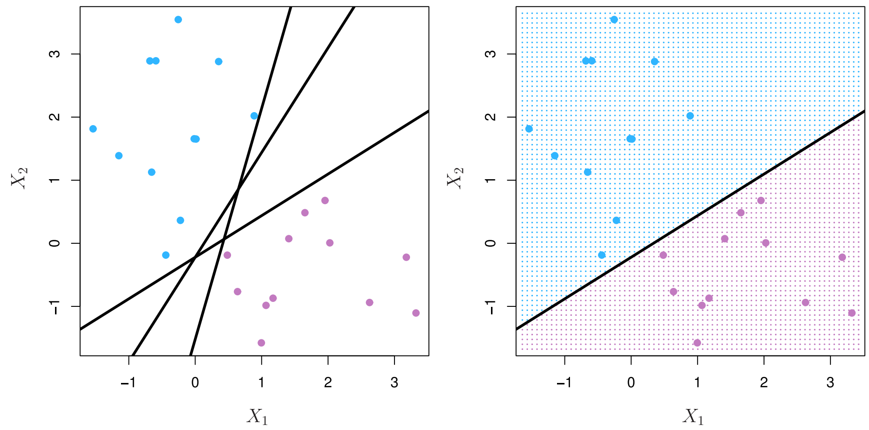

Figure 1: Fig. 9.2 from ISLR.

Margin

- Compute distance from each training sample to the separating hyperplane.
- Smallest distance among all samples is called the **margin**.

Maximal margin classifier

- For classifier to be more robust to noise, we should maximize the margin.
- Find the widest "slab" we can fit between the two classes.
- Choose the midline of this "slab" as the decision boundary.

Maximal margin classifier - illustration

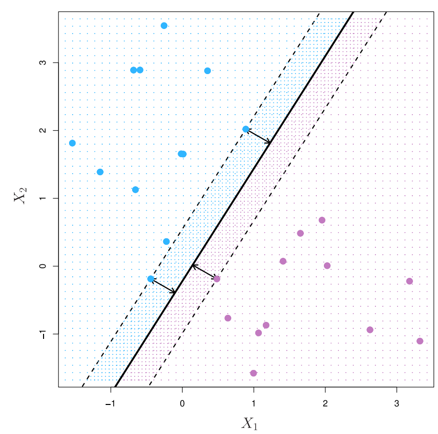

Figure 2: Fig. 9.3 from ISLR.

Support vectors

- Points that lie on the border of maximal margin hyperplane are **support vectors**
- They "support" the maximal margin hyperplane: if these points move, then the maximal margin hyperplane moves
- Maximal margin hyperplane is not affected by movement of any other point, as long as it doesn't cross borders!

Constructing the maximal margin classifier (1)

$$\underset{\boldsymbol{\beta},\gamma}{\text{maximize}}\ \gamma \tag{2}$$

$$\text{subject to: } \sum_{j=1}^{p} \beta_j^2 = 1 \tag{3}$$

$$\text{and } y_i \left(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} \right) \geq \gamma, \forall i = 1, \ldots, N \tag{4}$$

Constructing the maximal margin classifier (2)

The constraint

$$y_i \left(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} \right) \geq \gamma, \forall i = 1, \ldots, N$$

guarantees that each observation is on the correct side of the hyperplane *and* on the correct side of the margin, if margin γ is positive. (This is analogous to Equation 1, but we have added a margin.)

Constructing the maximal margin classifier (3)

The constraint

$$\text{and } \sum_{j=1}^{p} \beta_j^2 = 1$$

is not really a constraint: if a separating hyperplane is defined by $\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} = 0$, then for any $k \neq 0$, $k \left(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} \right) = 0$ is also a separating hyperplane.

This "constraint" just scales weights so that distance from ith sample to the hyperplane is given by $y_i \left(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} \right)$. This is what make the previous constraint meaningful!

Constructing the maximal margin classifier (4)

Therefore, the constraints ensure that

- Each observation is on the correct side of the hyperplane, and
- at least γ away from the hyperplane

and γ is maximized.

Problems with MM classifier (1)

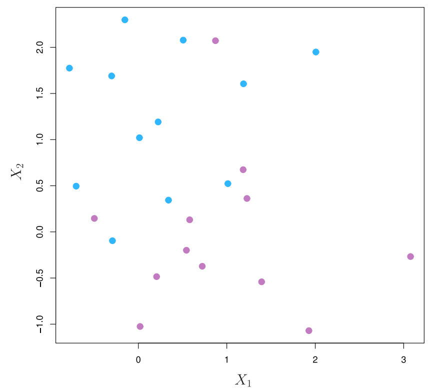

Figure 3: ISLR Fig. 9.4: data may not be separable. Optimization problem has no solution with $\gamma > 0$.

Problems with MM classifier (2)

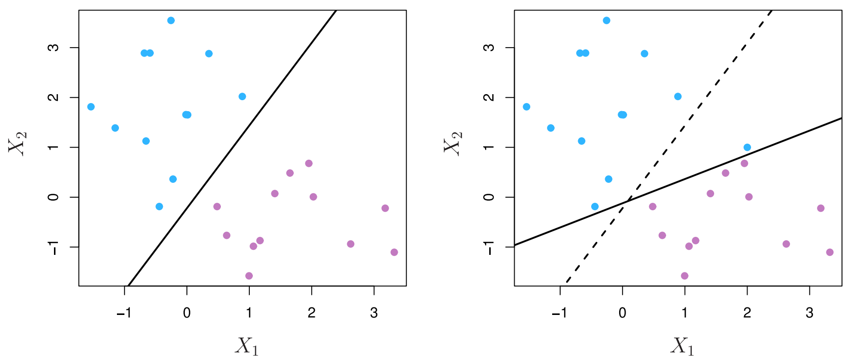

Figure 4: ISLR Fig. 9.5: MM classifier is not robust.

Support vector classifier

Basic idea

- Generalization of MM classifier to non-separable case
- Use a hyperplane that *almost* separates the data
- "Soft margin"

Constructing the support vector classifier

$$\underset{\boldsymbol{\beta},\boldsymbol{\epsilon},\gamma}{\text{maximize}}\, \gamma \tag{5}$$

$$\text{subject to: } \sum_{j=1}^{p} \beta_j^2 = 1 \tag{6}$$

$$y_i \left(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} \right) \geq \gamma(1 - \epsilon_i), \forall i = 1, \ldots, N \tag{7}$$

$$\epsilon_i \geq 0, \sum_{i=1}^{N} \epsilon_i \leq C \tag{8}$$

C is a non-negative tuning parameter.

Constructing the support vector classifier (3)

Slack variable ϵ_i determines where a point lies:

- If $\epsilon_i = 0$, point is on the correct side of margin
- If $\epsilon_i > 0$, point has *violated* the margin (wrong side of margin)
- If $\epsilon_i > 1$, point is on wrong side of hyperplane and is misclassified

Constructing the support vector classifier (4)

C is the **budget** that determines the number and severity of margin violations we will tolerate.

- $C = 0 \to$ same as MM classifier
- $C > 0$, no more than C observations may be on wrong side of hyperplane
- As C increases, margin widens; as C decreases, margin narrows.

Illustration of effect of C

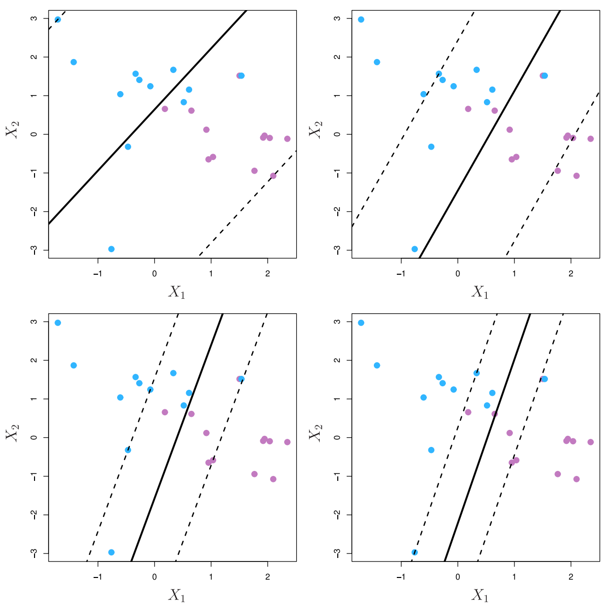

Figure 5: ISLR Fig. 9.7: Margin shrinks as C decreases.

Support vector

For a support vector classifier, the only points that affect the classifier are:

- Points that lie on the margin boundary
- Points that violate margin

These are the *support vectors*.

C controls bias-variance tradeoff

- When C is large: many support vectors, variance is low, but bias may be high.
- When C is small: few support vectors, high variance, but low bias.

Important terminology note

In ISLR and in these notes, meaning of C is opposite its meaning in Python `sklearn`:

- ISLR and these notes: Large C, wide margin.
- Python `sklearn`: Large C, small margin.

Constrained vs. Lagrange forms

In general, we may see a model expressed in **constrained form**, with tuning parameter $t \in \mathbb{R}$:

$$\underset{x \in \mathbb{R}^n}{\text{minimize}}\, f(x) \text{ subject to } h(x) \leq t$$

and also in **Lagrange form**, with tuning parameter $\lambda \geq 0$:

$$\underset{x \in \mathbb{R}^n}{\text{minimize}}\, f(x) + \lambda h(x)$$

Loss + penalty expression (1)

Equivalent expression for fitting support vector classifier using *hinge loss*:

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \left(\sum_{i=1}^{N} \text{max}[0, 1 - y_i f(x_i)] + \lambda \sum_{j=1}^{p} \beta_j^2 \right)$$

where λ is non-negative tuning parameter similar to C (large λ means wider margin) and $f(x_i) = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}$.

Loss + penalty representation (2)

With this representation: Zero loss for observations where

$$y_i \left(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} \right) \geq 1$$

and width of margin depends on $\sum \beta_j^2$.

Loss + penalty representation (3)

This is in contrast to previous representation, where: Zero loss for observations where

$$y_i \left(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} \right) \geq \gamma$$

and $\sum \beta_j^2 = 1$.

Compared to logistic regression

- **Hinge loss**: zero for points on correct side of margin.
- **Logistic regression loss**: small for points that are far from decision boundary.

Hinge loss vs. logistic regression

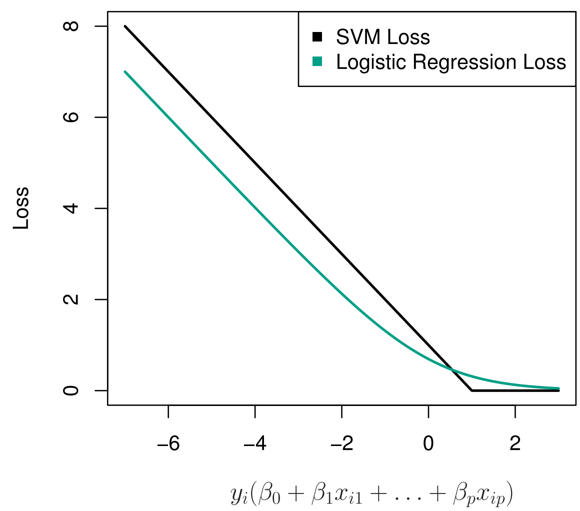

Figure 6: ISLR 9.12. Hinge loss is zero for points on correct side of margin.

Maximizing the margin

Optimization review

Reference: Appendix C.3 of Boyd and Vandenberghe, "Introduction to Applied Linear Algebra".

Constrained optimization

Basic formulation of contrained optimization problem:

- **Objective**: Minimize $f(x)$
- **Constraint(s)**: subject to $g(x) \leq 0$

Find a point $\hat{x}$ that satisfies $g(\hat{x}) \leq 0$ and, for any other x that satisfies $g(x) \leq 0$, $f(x) \geq f(\hat{x})$.

Definition of Lagrangian

Define the Lagrangian as the weighted sum of all constraints:

$$L(x, \lambda) = f(x) + \lambda_1 g_1(x) + \cdots + \lambda_p g_p(x)$$

$$= f(x) + g(x)^T \lambda$$

where λ is the *Lagrange multiplier*. $g(x)^T \lambda$ "attracts" toward the feasible set, away from the non-feasible set.

Dual problem (with extra details not shown in class)

Expressed in terms of $L(x, \lambda)$, the primal problem is equivalent to

$$\min_x \max_{\lambda \geq 0} L(x, \lambda)$$

The dual problem is

$$\max_{\lambda \geq 0} \min_x L(x, \lambda)$$

KKT conditions (1)

Under some technical conditions: if $\hat{x}$ is a local minima, then there is a vector $\hat{\lambda}$ that satisfies:

$$\frac{\partial L}{\partial x_i}(\hat{x}, \hat{\lambda}) = 0, i = 1 \ldots, n$$

$$\frac{\partial L}{\partial \lambda_i}(\hat{x}, \hat{\lambda}) = 0, i = 1 \ldots, p$$

(produces as many equations as there are unknowns!)

KKT conditions (2)

$$g_i(x) \leq 0, \quad i = 1, \ldots, p$$

$$\lambda_i \geq 0, \quad i = 1, \ldots, p$$

$$\lambda_i g_i(x) = 0, \quad i = 1, \ldots, p$$

Active vs. inactive constraints

At the optimal point, some constraints will be "binding" and some will be "slack" - either:

- $g_i(\hat{x}) < 0$ and $\hat{\lambda}_i = 0$ (optimum is inside feasible set, constraint is inactive)
- $g_i(\hat{x}) = 0$ and $\hat{\lambda}_i \geq 0$ (optimum is outside feasible set, constraint is active)

Active vs. inactive constraints (illustration)

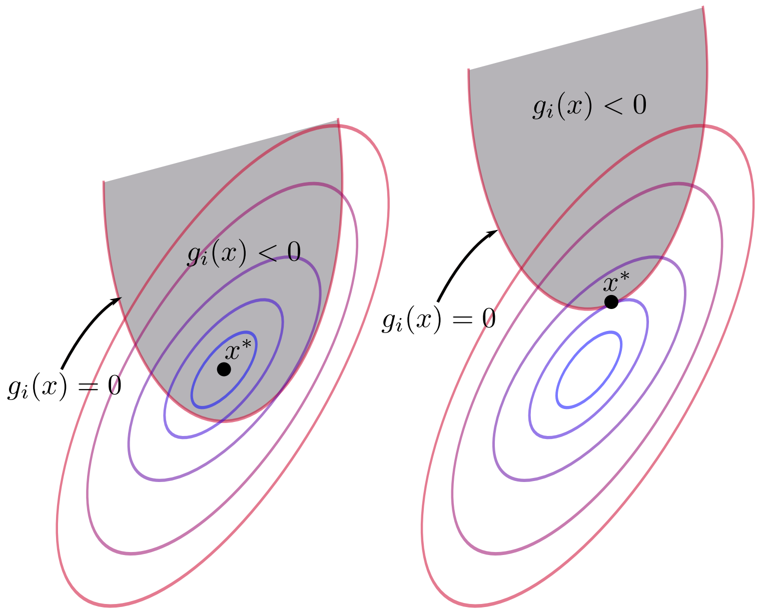

Figure 7: Image via Wikipedia

Comment on notation

For the following section, we use `sklearn` notation, with opposite meaning of C -

- in the previous formulation we had a tuning parameter λ that multiplied the penalty term, and increasing this parameter widens the margin
- now C multiplies the loss term, and increasing this parameter narrows the margin.

Support vector classifier as constrained optimization (1)

The support vector classifier problem is:

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \left(C \sum_{i=1}^{N} \epsilon_i + \frac{1}{2} \sum_{j=1}^{p} \beta_j^2 \right)$$

subject to:

$$y_i(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}) \geq 1 - \epsilon_i \text{ and } \epsilon_i \geq 0, \quad \forall i = 1, \ldots, N$$

Support vector classifier as constrained optimization (extra details)

Construct Lagrange function $L(\beta, \alpha, \mu)$ where

- α is the vector of Lagrange multipliers for the set of constraints $y_i(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}) \geq 1 - \epsilon_i$
- μ is the vector of Lagrange multipliers for the set of constraints $\epsilon_i \geq 0$

Then the dual problem is:

$$\max_{\alpha,\mu} \min_{\beta} L(\beta, \alpha, \mu)$$

subject to

$$\alpha_i \geq 0, \quad \mu_i \geq 0, \quad \forall i$$

which becomes:

$$\underset{\alpha}{\text{maximize}} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j$$

subject to:

$$\sum_i \alpha_i y_i = 0, \quad C \geq \alpha_i \geq 0, \quad \forall i$$

Support vector classifier as constrained optimization (2)

Optimal coefficients for $j = 1, \ldots, p$ are:

$$\beta_j = \sum_{i=1}^{N} \alpha_i y_i x_{ij}$$

where α_i come from the solution to the dual problem.

Support vector classifier as constrained optimization (3)

- $\alpha_i > 0$ only when x_i is a support vector (active constraint).
- Otherwise, $\alpha_i = 0$ (inactive constraint).

Support vector classifier as constrained optimization (4)

That leaves β_0 - for any i where $\alpha_i > 0$, we can find β_0 from

$$\beta_0 = y_i - \sum_{j=1}^{p} \beta_j x_{ij}$$

Why solve dual problem?

For high-dimension problems (many features), dual problem can be much faster to solve than primal problem:

- Primal problem: optimize over $p + 1$ coefficients.
- Dual problem: optimize over n dual variables, but there are only as many non-zero ones as there are support vectors.

Correlation interpretation (1)

Given a new sample $\mathbf{x}$ to classify, compute

$$\hat{z}(\mathbf{x}) = \beta_0 + \sum_{j=1}^{p} \beta_j x_j = \beta_0 + \sum_{i=1}^{N} \alpha_i y_i \sum_{j=1}^{p} x_{ij} x_j$$

Measures inner product (a kind of "correlation") between new sample and each support vector.

Correlation interpretation (2)

Classifier output (assuming -1,1 labels):

$$\hat{y}(\mathbf{x}) = \text{sign}(\hat{z}(\mathbf{x}))$$

Predicted label is weighted average of labels for support vectors, with weights proportional to "correlation" of test sample and support vector.

Support vector machines

Extension to non-linear decision boundary

- For logistic regression: we used functions of $\mathbf{x}$ to increase the feature space to classify data that is not linearly separable.
- Could use similar approach here.

SVM in transformed form (1)

Coefficients:

$$\beta_j = \sum_{i=1}^{N} \alpha_i y_i \phi(\mathbf{x}_{ij})$$

Classifier discriminant:

$$z = \beta_0 + \sum_{i=1}^{N} \alpha_i y_i \phi(\mathbf{x}_i) \phi(\mathbf{x})$$

SVM in transformed form (2)

Classifier output:

$$\hat{y} = \text{sign}(z)$$

Important: solution uses inner product of transformed samples, not necessarily transformed samples themselves.

Kernel trick

$K(\mathbf{x}_i, \mathbf{x}) = \phi(\mathbf{x}_i)\phi(\mathbf{x})$ is a "kernel".
Classifier discriminant with kernel:

$$z = \beta_0 + \sum_{i=1}^{N} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})$$

Can directly compute $K(\mathbf{x}_i, \mathbf{x})$ **without explicitly computing** $\phi(\mathbf{x})$!
(For more details: Mercer's theorem)

Kernel trick example

Kernel can be inexpensive to compute, even if basis function itself is expensive. For example, consider:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \phi(\mathbf{x}) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{bmatrix}$$

Kernel trick example - direct computation

Direct computation of $\phi(\mathbf{x}_n)\phi(\mathbf{x}_m)$: square or multiply 3 components of two vectors (6 operations), then compute inner product in $\mathbb{R}^3$ (3 multiplications, 1 sum).

$$\begin{aligned} \phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m) &= \begin{bmatrix} x_{n,1}^2 & x_{n,2}^2 & \sqrt{2}x_{n,1}x_{n,2} \end{bmatrix} \cdot \begin{bmatrix} x_{m,1}^2 \\ x_{m,2}^2 \\ \sqrt{2}x_{m,1}x_{m,2} \end{bmatrix} \\ &= x_{n,1}^2 x_{m,1}^2 + x_{n,2}^2 x_{m,2}^2 + 2x_{n,1}x_{n,2}x_{m,1}x_{m,2}. \end{aligned}$$

Kernel trick example - computation using kernel

Using kernel $K(x_n, x_m) = (x_n^T x_m)^2$: compute inner product in $\mathbb{R}^2$ (2 multiplications, 1 sum) and then square of scalar (1 square).

$$\begin{aligned} (\mathbf{x}_m^\top \mathbf{x}_m)^2 &= \left(\begin{bmatrix} x_{n,1} & x_{n,2} \end{bmatrix} \cdot \begin{bmatrix} x_{m,1} \\ x_{m,2} \end{bmatrix} \right)^2 \\ &= (x_{n,1}x_{m,1} + x_{n,2}x_{m,2})^2 \\ &= (x_{n,1}x_{m,1})^2 + (x_{n,2}x_{m,2})^2 + 2(x_{n,1}x_{m,1})(x_{n,2}x_{m,2}) \\ &= \phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m). \end{aligned}$$

Kernel intution

$K(\mathbf{x}_i, \mathbf{x})$ measures "similarity" between training sample $\mathbf{x}_i$ and new sample $\mathbf{x}$.

- Large K, more similarity

- K close to zero, not much similarity

$z = \beta_0 + \sum_{i=1}^{N} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})$ gives higher weight to training samples that are close to new sample.

Linear kernel

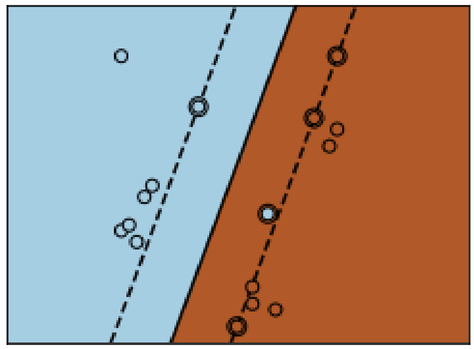

Figure 8: Linear kernel: $K(x, y) = x^T y$

Polynomial kernel

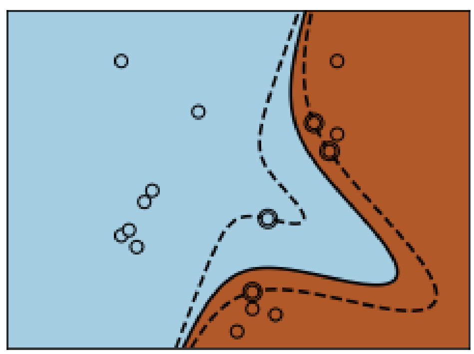

Figure 9: Polynomial kernel: $K(x, y) = (\gamma x^T y + c_0)^d$

Radial basis function kernel

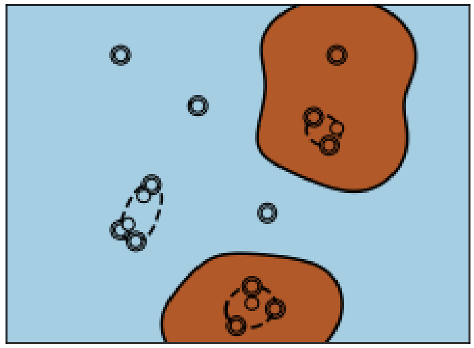

Figure 10: Radial basis function: $K(x, y) = \text{exp}(-\gamma||x-y||^2)$. If $\gamma = \frac{1}{\sigma^2}$, this is known as the Gaussian kernel with variance σ^2.

Infinite-dimensional feature space

With kernel method, can operate in infinite-dimensional feature space! Take for example the RBF kernel:

$$K_{\text{RBF}}(\mathbf{x}, \mathbf{y}) = \text{exp}\left(-\gamma\|\mathbf{x}-\mathbf{y}\|^2\right)$$

Let $\gamma = \frac{1}{2}$ and let $K_{\texttt{poly}(r)}$ be the polynimal kernel of degree r. Then

Infinite-dimensional feature space (extra steps not shown in class)

$$\begin{aligned}
K_{\text{RBF}}(\mathbf{x}, \mathbf{y}) &= \text{exp}\left(-\frac{1}{2}\|\mathbf{x}-\mathbf{y}\|^2\right) \\
&= \text{exp}\left(-\frac{1}{2}\langle\mathbf{x}-\mathbf{y}, \mathbf{x}-\mathbf{y}\rangle\right) \\
&\stackrel{\star}{=} \text{exp}\left(-\frac{1}{2}\langle\mathbf{x}, \mathbf{x}-\mathbf{y}\rangle - \langle\mathbf{y}, \mathbf{x}-\mathbf{y}\rangle\right) \\
&\stackrel{\star}{=} \text{exp}\left(-\frac{1}{2}\langle\mathbf{x}, \mathbf{x}\rangle - \langle\mathbf{x}, \mathbf{y}\rangle - [\langle\mathbf{y}, \mathbf{x}\rangle - \langle\mathbf{y}, \mathbf{y}\rangle]\rangle\right) \\
&= \text{exp}\left(-\frac{1}{2}\langle\mathbf{x}, \mathbf{x}\rangle + \langle\mathbf{y}, \mathbf{y}\rangle - 2\langle\mathbf{x}, \mathbf{y}\rangle\right) \\
&= \text{exp}\left(-\frac{1}{2}\|\mathbf{x}\|^2\right)\text{exp}\left(-\frac{1}{2}\|\mathbf{y}\|^2\right)\text{exp}\left(-2\langle\mathbf{x}, \mathbf{y}\rangle\right)
\end{aligned}$$

where the steps marked with a star use the fact that for inner products, $\langle\mathbf{u}+\mathbf{v}, \mathbf{w}\rangle = \langle\mathbf{u}, \mathbf{w}\rangle + \langle\mathbf{v}, \mathbf{w}\rangle$.

Infinite-dimensional feature space (2)

Let C be a constant

$$C \equiv \exp\Big(-\frac{1}{2}\|\mathbf{x}\|^2\Big)\exp\Big(-\frac{1}{2}\|\mathbf{y}\|^2\Big)$$

And note that the Taylor expansion of $e^{f(x)}$ is:

$$e^{f(x)} = \sum_{r=0}^{\infty}\frac{[f(x)]^r}{r!}$$

Finally, the RBF kernel can be viewed as an infinite sum over polynomial kernels:

$$\begin{aligned}
K_{\texttt{RBF}}(\mathbf{x},\mathbf{y}) &= C\exp\big(-2\langle\mathbf{x},\mathbf{y}\rangle\big)\\
&= C\sum_{r=0}^{\infty}\frac{\langle\mathbf{x},\mathbf{y}\rangle^r}{r!}\\
&= C\sum_{r}^{\infty}\frac{K_{\texttt{poly(r)}}(\mathbf{x},\mathbf{y})}{r!}
\end{aligned}$$

Extension to regression

- Similar idea
- Only points outside the margin contribute to final cost

SVR illustration

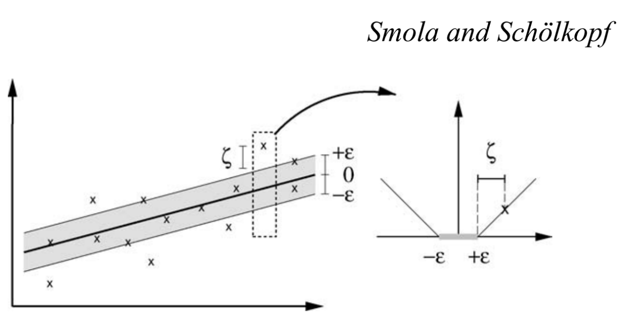

Figure 11: Support vector regression.

Summary: SVM

Key expression

Discriminant can be computed using an inexpensive kernel function on a small number of support vector points ($i \in S$ are the subset of training samples that are support vectors):

$$z = \beta_0 + \sum_{i \in S} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})$$

Key ideas

- Defines boundary with greatest separation between classes
- Tuning parameter controls complexity (which direction depends on notation/“meaning” of C)
- Kernel trick allows efficient extension to higher-dimension space: non-linear decision boundary through transformation of features, but without explicitly computing high-dimensional features.