Support Vector Machines & Kernels

Doing *really* well with linear decision surfaces

These slides were assembled by Byron Boots, with only minor modifications from Eric Eaton's slides and grateful acknowledgement to the many others who made their course materials freely available online. Feel free to reuse or adapt these slides for your own academic purposes, provided that you include proper attribution.

Adapted from slides by Tim Oates

One Slide Summary

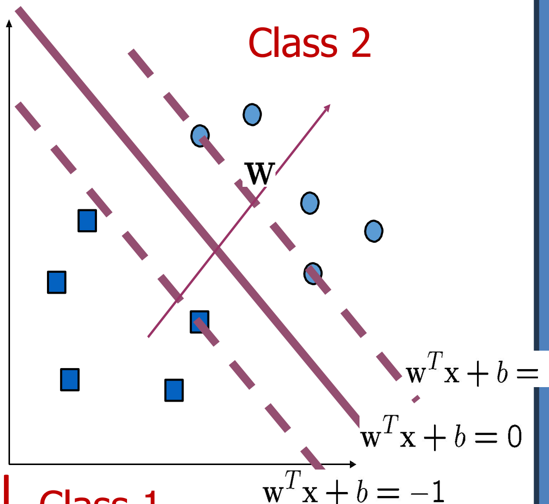
Minimize $\frac{1}{2}\boldsymbol{\theta}^{\top}\boldsymbol{\theta}$
subject to $y_n(\boldsymbol{\theta}^{\top}x_n) \geq 1$, for $n = 1,2,\ldots N$

Maximize Margin
Learn θ (hyperplane)

Minimize **Primal**

$$L(\boldsymbol{\theta},\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\theta}^{\top}\boldsymbol{\theta} - \sum_{i=1}^{n} \alpha_i (y_i(\boldsymbol{\theta}^T x_i) - 1)$$

$$\text{s.t.}\ \forall i \ \ \alpha_i \geq 0$$



Dual

Maximize $J(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$

s.t. $\alpha_i \geq 0 \ \forall i$ (comes from Lagrangian Assumptions)

$$\sum_i \alpha_i y_i = 0$$

(comes from differentiating w.r.t θ_0)

Maximize Margin
Learn α (weight
of support vectors)

SVM Dual

Maximize $J(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$

s. t. $\alpha_i \geq 0 \; \forall i$

$$\sum_i \alpha_i y_i = 0$$

The decision function is given by

$$h(\mathbf{x}) = \text{sign}\left(\sum_{i \in \mathcal{SV}} \alpha_i y_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b \right)$$

$$\text{where} \quad b = \frac{1}{|\mathcal{SV}|} \sum_{i \in \mathcal{SV}} \left(y_i - \sum_{j \in \mathcal{SV}} \alpha_j y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \right)$$

Interesting Twist: Many α_i's are zero. Only SVs have non-zero α_i's

Understanding the Dual

$$\text{Maximize } J(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boxed{\langle \mathbf{x}_i, \mathbf{x}_j \rangle}$$

$$\text{s. t. } \alpha_i \geq 0 \ \forall i$$

$$\sum_i \alpha_i y_i = 0$$

In the solution, either:

- $\alpha_i > 0$ and the constraint is tight $(y_i(\boldsymbol{\theta}^\top x_i) = 1)$
  - point is a support vector
- $\alpha_i = 0$
  - point is not a support vector

What if Data Are Not Linearly Separable?

Cannot find $\boldsymbol{\theta}$ that satisfies. $y_i(\theta^T x_i) \geq 1 \; \forall i$

Introduce slack variables ξ_i

$$y_i(\theta^T x_i) \geq 1 - \xi_i \; \forall i$$

New Problem

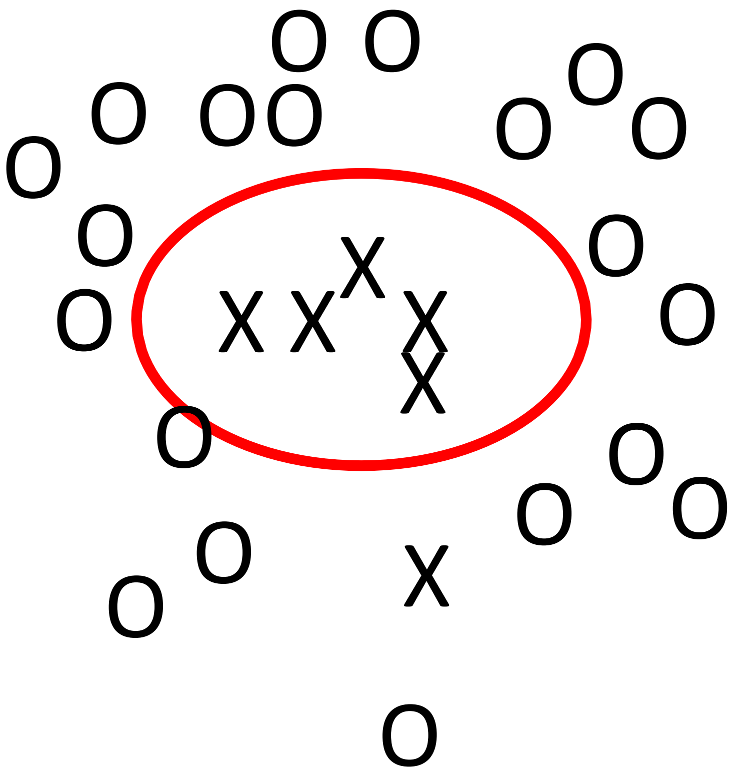
$$\min_{\theta} \frac{1}{2} \sum_{j=1}^{d} \theta_j^2 + C \sum_{i} \xi_i$$

$$\text{s.t. } y_i(\theta^T x_i) \geq 1 - \xi_i, \quad \text{if } \forall i$$

Strengths of SVMs

- Good generalization in theory
- Good generalization in practice
- Work well with few training instances
- Find globally best model
- Efficient algorithms
- Amenable to the kernel trick …

What if Surface is Non-Linear?



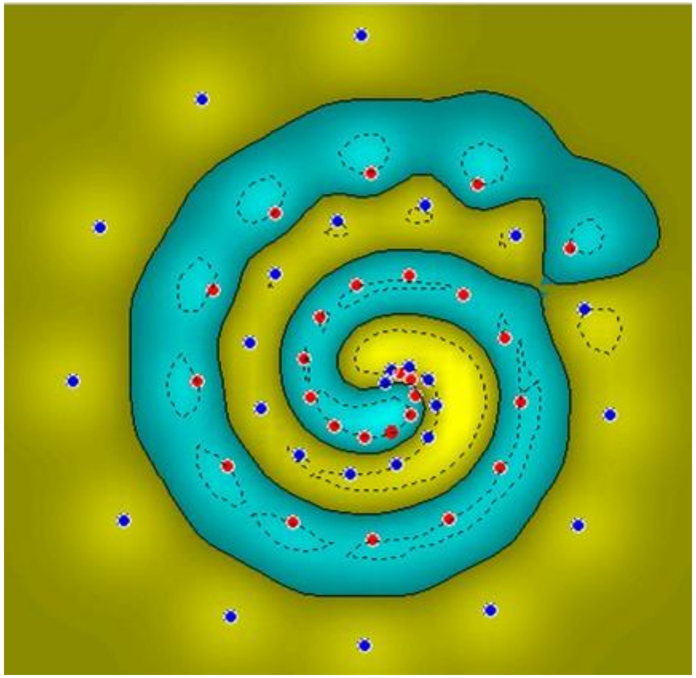
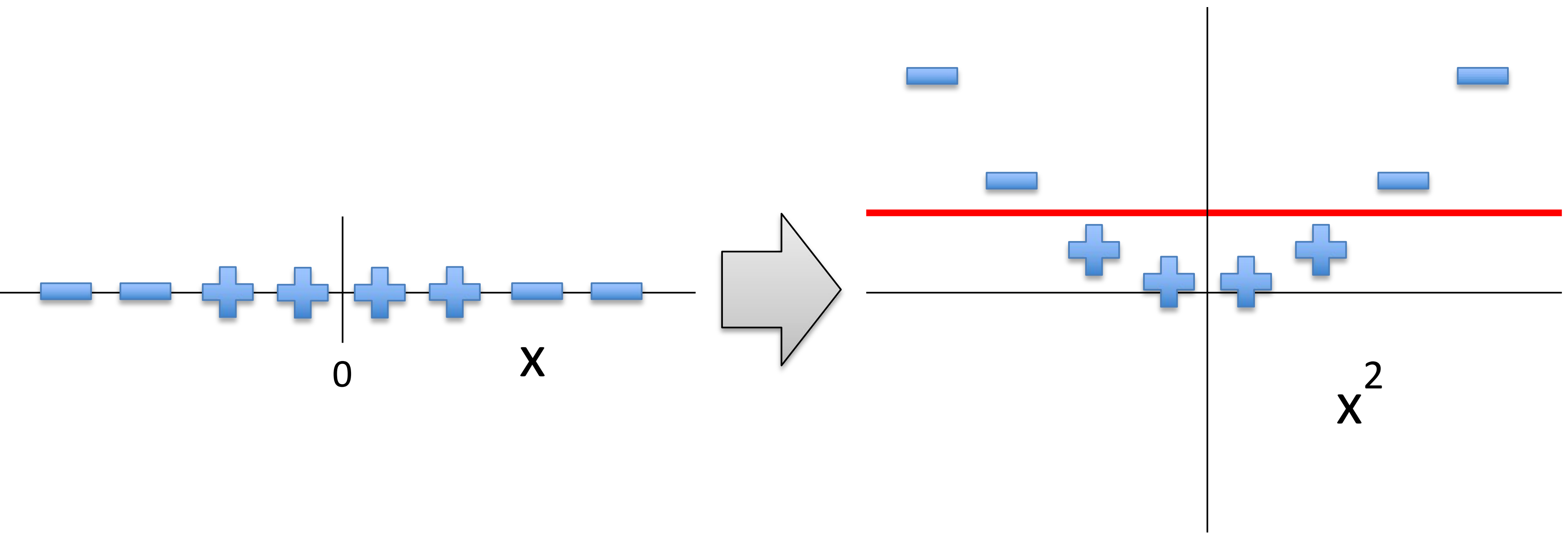


Image from http://www.atrandomresearch.com/iclass/

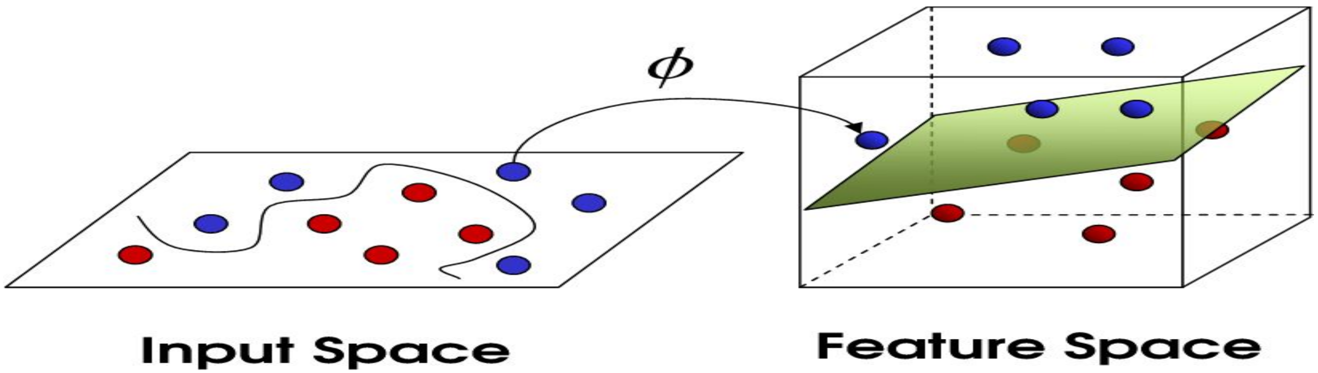
Kernel Methods

Making the Non-Linear Linear

When Linear Separators Fail



Mapping into a New Feature Space



$$\Phi: \mathcal{X} \rightarrow \hat{\mathcal{X}} = \Phi(\mathbf{x})$$

- For example, with $\mathbf{x}_i \in \mathbb{R}^2$

$$\Phi([x_{i1}, x_{i2}]) = [x_{i1}, x_{i2}, x_{i1}x_{i2}, x_{i1}^2, x_{i2}^2]$$

- Rather than run SVM on $\mathbf{x}_i$, run it on $\Phi(\mathbf{x}_\mathrm{i})$
  - Find non-linear separator in input space
- What if $\Phi(\mathbf{x}_\mathrm{i})$ is really big?
- Use kernels to compute it implicitly!

Image from http://web.engr.oregonstate.edu/ ~afern/classes/cs534/

Kernels

- Find kernel K such that
$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$$

- Computing $K(\mathbf{x}_i, \mathbf{x}_j)$ should be efficient, much more so than computing $\Phi(\mathbf{x}_i)$ and $\Phi(\mathbf{x}_j)$

- Use $K(\mathbf{x}_i, \mathbf{x}_j)$ in SVM algorithm rather than $(\mathbf{x}_i, \mathbf{x}_j)$

- Given a function K, it is possible to verify that it is a kernel

Kernel Matrix

- (aka the Gram matrix):

K=

K(1,1)	K(1,2)	K(1,3)	…	K(1,m)
K(2,1)	K(2,2)	K(2,3)	…	K(2,m)
…	…	…	…	…
K(m,1)	K(m,2)	K(m,3)	…	K(m,m)

- The kernel matrix is Symmetric Positive (semi) Definite.
- Any symmetric positive definite matrix can be regarded as a kernel matrix, that is as an inner product matrix in some space

Mercer's Theorem

Every (semi) positive definite, symmetric function is a kernel: i.e. there exists a mapping ϕ s.t. it is possible to write

$$K(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle$$

Positive Definite

$\int K(x,z) f(x) f(z) dx dz \geq 0 , \forall f$ which is square-integrable

More Properties/Kernel Algebra: UMASS CS SVM Lecture Notes

The Polynomial Kernel

Let $\mathbf{x}_i = [x_{i1}, x_{i2}]$ and $\boldsymbol{x}_j = [x_{j1}, x_{j2}]$

Consider the following function:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle^2 = (x_{i1}x_{j1} + x_{i2}x_{j2})^2$$

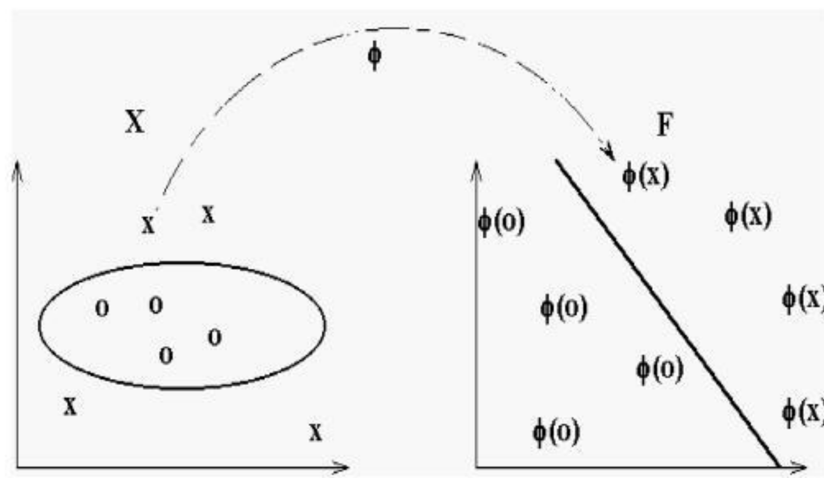
$$= (x_{i1}^2 x_{j1}^2 + x_{i2}^2 x_{j2}^2 + 2x_{i1}x_{i2}x_{j2}x_{j2})$$

$$= \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$$

Where

$$\Phi(\mathbf{x}_i) = [x_{i1}^2, x_{j1}^2, x_{i1}x_{j1}, x_{j1}x_{i1}]$$

$$\Phi(\mathbf{x}_i) = [x_{i2}^2, x_{j2}^2, x_{i2}x_{j2}, x_{j2}x_{i2}]$$



Problems with Feature Space

- Working in high dimensional feature spaces solves the problem of expressing complex functions

But:

- There is a computational problem (working with very large vectors)
- And a generalization theory problem (curse of dimensionality)

The Polynomial Kernel

- Given by $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle^d$
  - Φ(x) contains all monomials of degree d
- Useful in visual pattern recognition
  - Example:
    - 16x16 pixel image
    - 10^{10} monomials of degree 5
    - Never explicitly compute Φ(x) !
- Variation: $K(\mathbf{x}_i, \mathbf{x}_j) = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle + 1)^d$
  - Adds all lower-order monomials (degrees 1,...,d)!

Why Kernels are Efficient

The kernel is essentially a function to perform calculations even in the higher dimensions.

- We use Kernels in SVM to reduce the complexity of calculations.
- A Kernel can do calculations for an even infinite number of dimensions.
  - As the dimensions increase in SVM, it becomes difficult to form a HyperPlane, so we have to resort to Kernels to form a hyperplane.

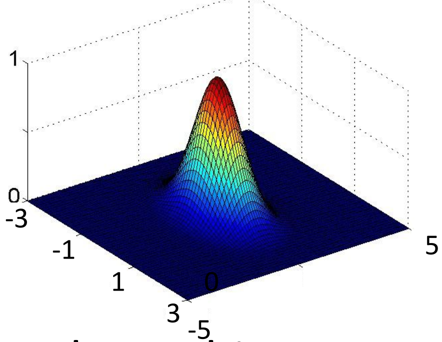
The Gaussian Kernel

- Also called Radial Basis Function (RBF) kernel

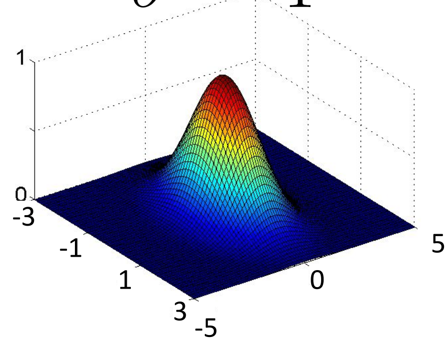
$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}\right)$$

  - Has value 1 when $x_i = x_j$
  - Value falls off to 0 with increasing distance
  - Note: Need to do feature scaling <u>before</u> using Gaussian Kernel

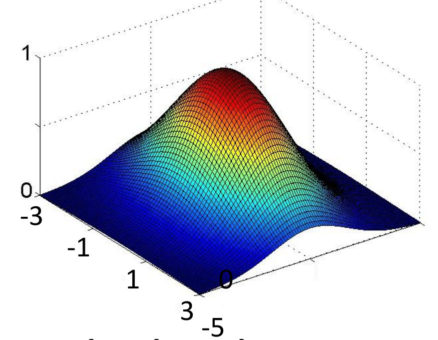
$\sigma^2 = 0.5$



$\sigma^2 = 1$



$\sigma^2 = 3$



lower bias,
higher variance

higher bias,
lower variance

The Kernel Trick

"Given an algorithm which is formulated in terms of a positive definite kernel K_1, one can construct an alternative algorithm by replacing K_1 with another positive definite kernel K_2"

- SVMs can use the kernel trick

Incorporating Kernels into SVM

$$J(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \cancel{\langle \mathbf{x}_i, \mathbf{x}_j \rangle}$$

$$J(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boxed{K(\mathbf{x}_i, \mathbf{x}_j)}$$

$$h(\mathbf{x}) = \operatorname{sign}\left(\sum_{i \in \mathcal{SV}} \alpha_i y_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b\right)$$

A Few Good Kernels...

- Linear Kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$
- Polynomial kernel $K(\mathbf{x}_i, \mathbf{x}_j) = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle + c)^d$
  - $c \geq 0$ trades off influence of lower order terms
- Gaussian kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}\right)$
- Sigmoid kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh\left(\alpha \mathbf{x}_i^\intercal \mathbf{x}_j + c\right)$

Many more...

- Cosine similarity kernel
- Chi-squared kernel
- String/tree/graph/wavelet/etc kernels

Practical Advice for Applying SVMs

- Use SVM software package to solve for parameters
  - e.g., SVMlight, libsvm, cvx (fast!), etc.

- Need to specify:
  - Choice of parameter C
  - Choice of kernel function
    - Associated kernel parameters
    - E.g., $K(\mathbf{x}_i, \mathbf{x}_j) = \left(\langle \mathbf{x}_\mathbf{i}, \mathbf{x}_j \rangle + c\right)^d$
    - $K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}\right)$

SVMs vs Logistic Regression (Advice from Andrew Ng)

n = # training examples   d = # features

If d is large (relative to n) (e.g., d > n with d = 10,000, n = 10-1,000)

- Use logistic regression or SVM with a linear kernel

If d is small (up to 1,000), n is intermediate (up to 10,000)

- Use SVM with Gaussian kernel

If d is small (up to 1,000), n is large (50,000+)

- Create/add more features, then use logistic regression or SVM without a kernel

Neural networks likely to work well for most of these settings, but may be slower to train

Conclusion

- SVMs find optimal linear separator
- The kernel trick makes SVMs learn non-linear decision surfaces

- Strength of SVMs:
  - Good theoretical and empirical performance
  - Supports many types of kernels

- Disadvantages of SVMs:
  - “Slow” to train/predict for huge data sets (but relatively fast!)
  - Need to choose the kernel (and tune its parameters)