AI & ML: STATISTICAL LEARNING & PREDICTION

Professor: PABLO GERVÁS

Professor: PABLO GERVÁS
E-mail: pgervas@faculty.ie.edu

SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

Kernel Methods and SVM

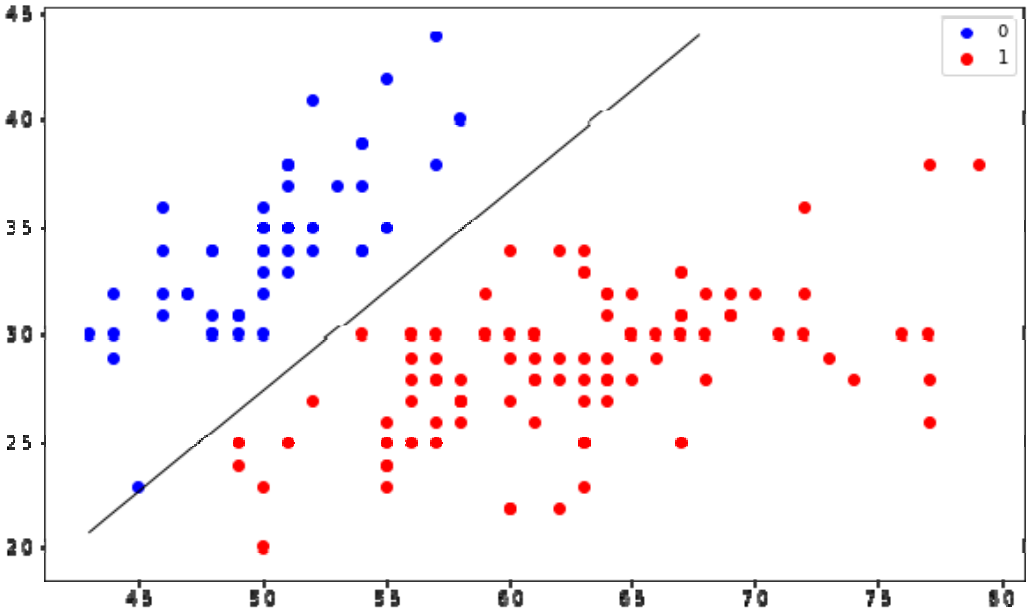
Kernel methods and SVM

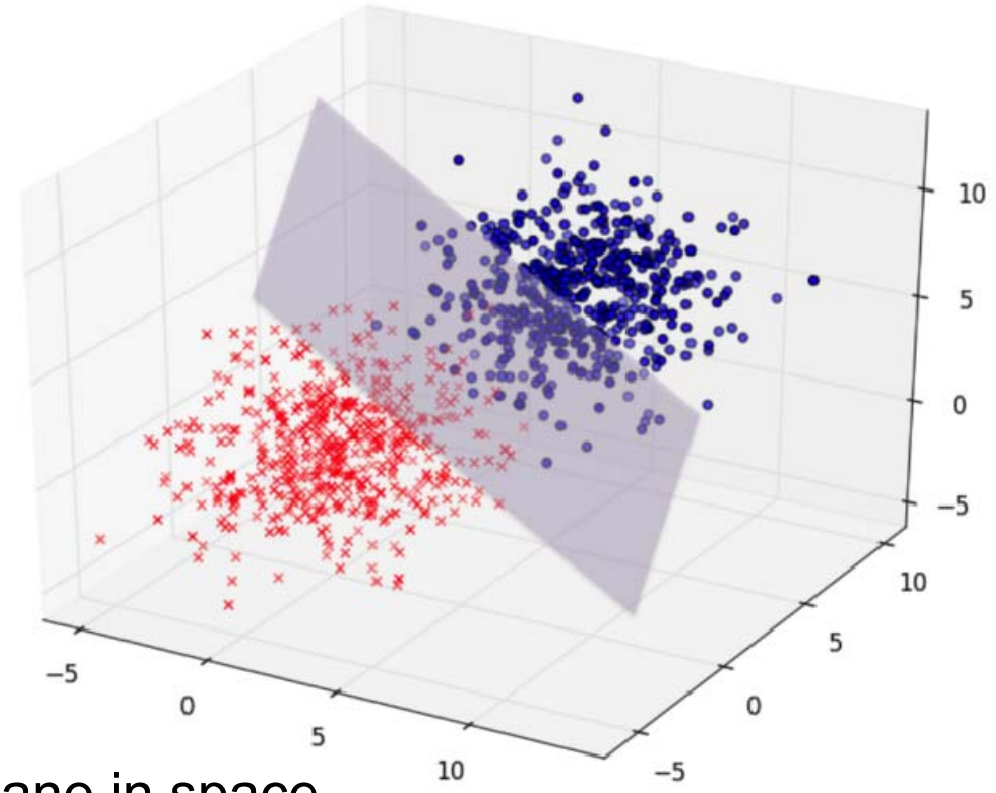
Support Vector Classifier

- Large margin intuition
- Basic math review
- Optimization objective

Kernel methods

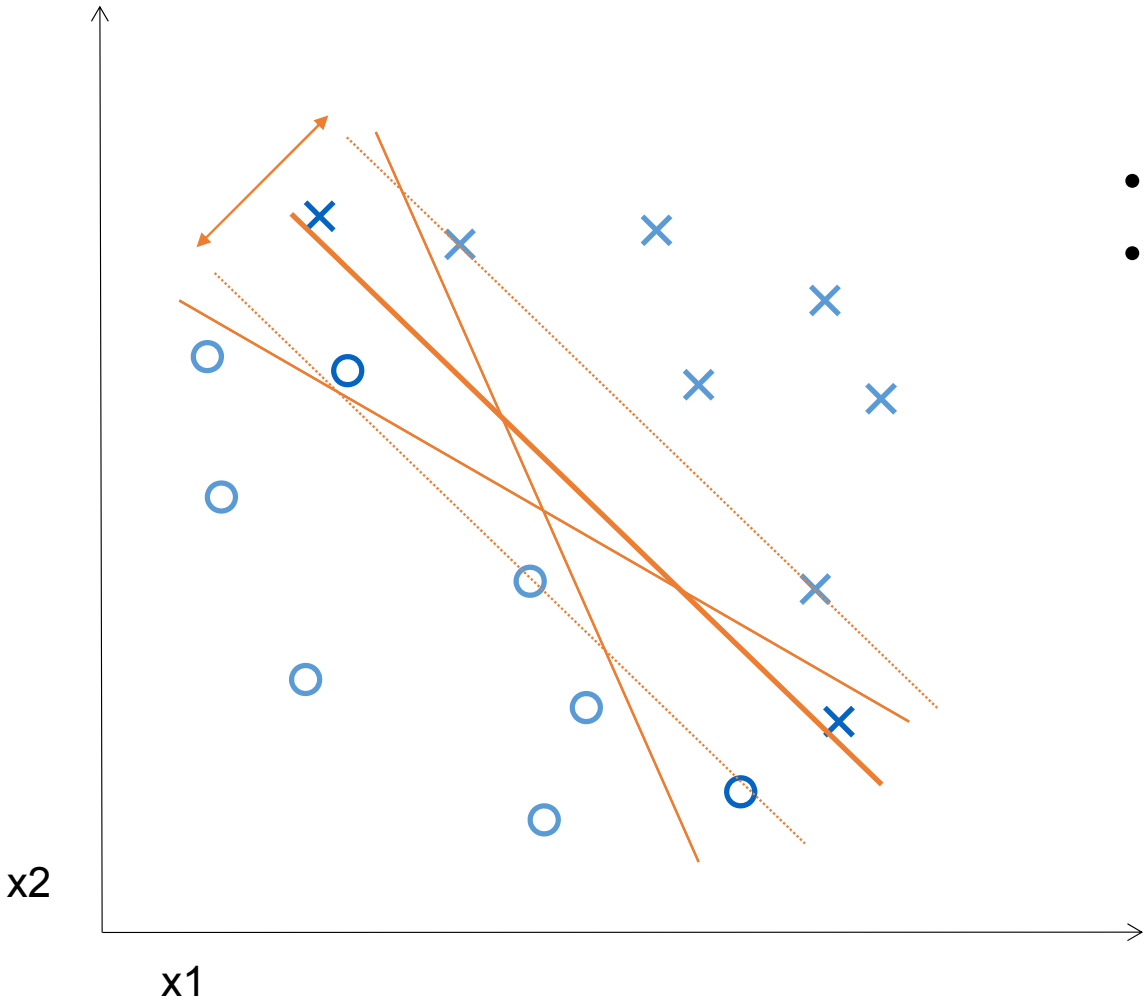
- Non-linear trasnformations
- Intuition behind Kernel methods
- Overall structure
- Using an SVM
- Multi-class





- We think of a linear model as a hyperplane in space
- The margin is the minimum distance of all closest point (misclassified have negative distance)
- The support vector machine is the hyperplane with largest margin

Large Margin Intuition

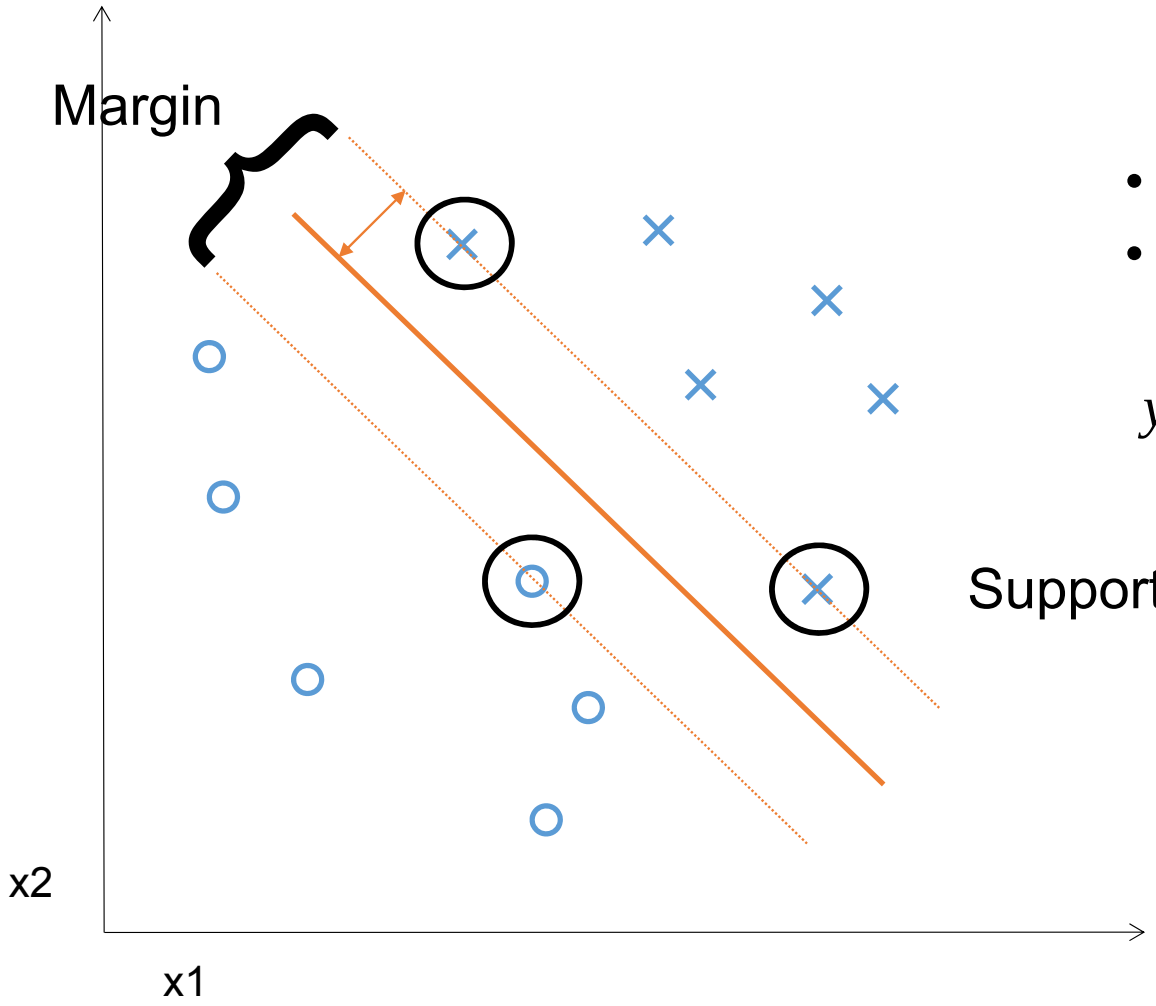


- vectors $\mathbf{x}_i$
- labels $y_i = \pm 1$

× $y_i = +1$

○ $y_i = -1$

Optimal Margin Separating Hyperplane

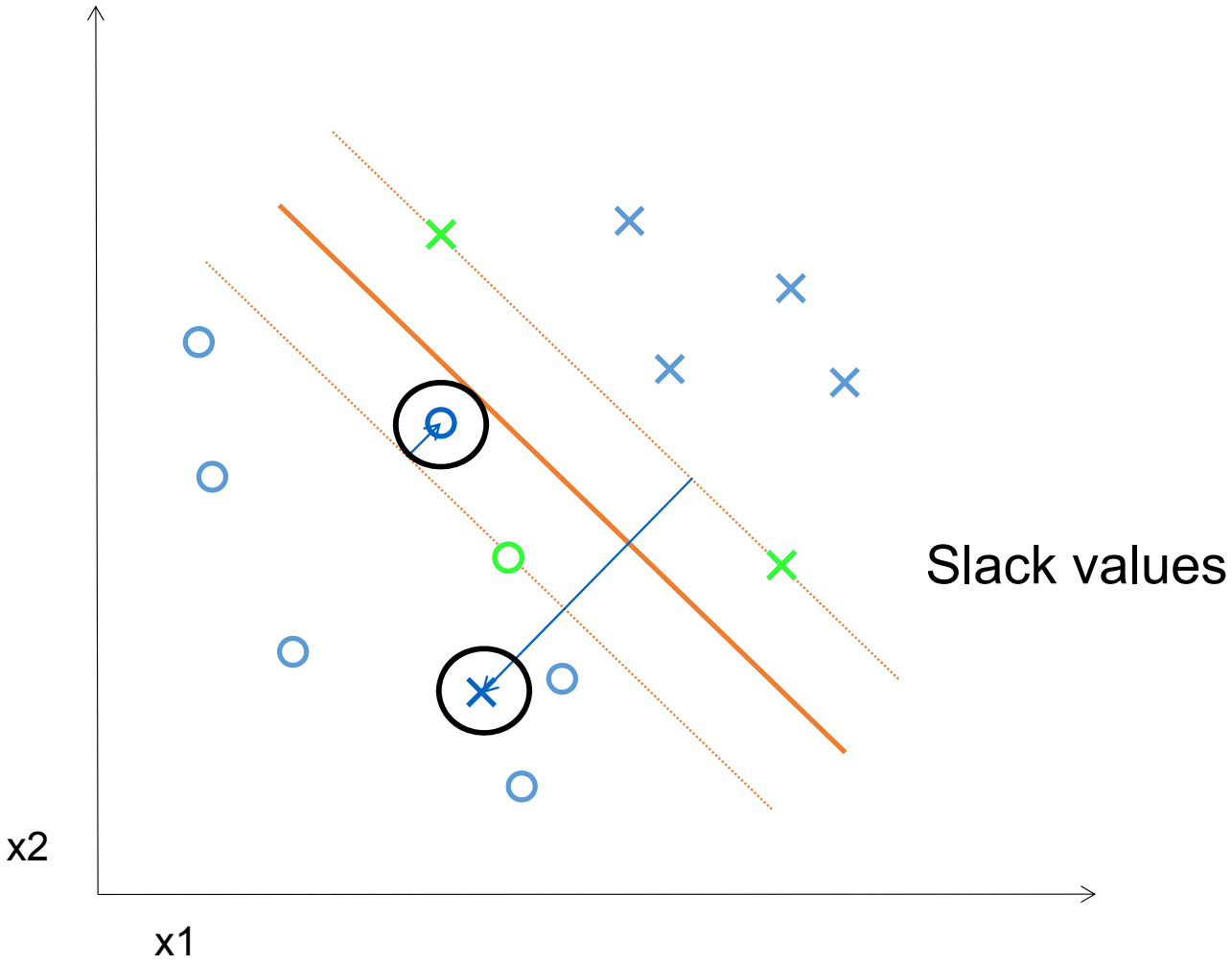


- vectors $\mathbf{X}_i$
- labels $y_i = \pm 1$

$$y = \text{sign}(\mathbf{w} \cdot \mathbf{X} + b)$$

Support vectors

Soft Margin SVM



Slack values

Linear SVM in Python

Google “In-Depth Support Vector Machines”

https://jakevdp.github.io/PythonDataScienceHandbook/
05.07-support-vector-machines.html

Code example 2:	`svm-in-depth-1.py`	Plot blobs data set
Code example 3:	`svm-in-depth-2.py`	Possible separating lines
Code example 4:	`svm-in-depth-3.py`	Plot lines with margins
Code example 5:	`svm-in-depth-4.py`	Plot SVC
Code example 6:	`svm-in-depth-5.py`	Plot for subsets of data set

Linear SVM in Python

Google “Implementing SVM Kernel SVM Python Scikit-Learn”

https://stackabuse.com/
implementing-svm-and-kernel-svm-with-pythons-scikit-learn/

Code example 1: `svm.py`

SVC on bill authentication data set

Basic maths to review

The equation of a line in the form ax + by = c can be written as a dot product:

$$(a,b) \cdot (x,y) = c, \quad \text{or} \quad A \cdot X = c$$

The equation of a plane in the form ax + by + cz = d can be written as a dot product:

$$(a,b,c) \cdot (x,y,z) = d, \quad \text{or} \quad A \cdot X = d$$

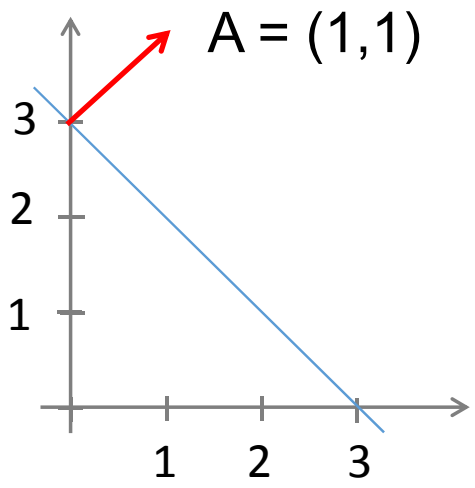
Given the plane with equation $A \cdot X = d$, the vector A is orthogonal to the plane.

$$x + y = 3$$

$$(1,1) \,.\, (x,y) = 3$$

$$A \,.\, X = 3$$

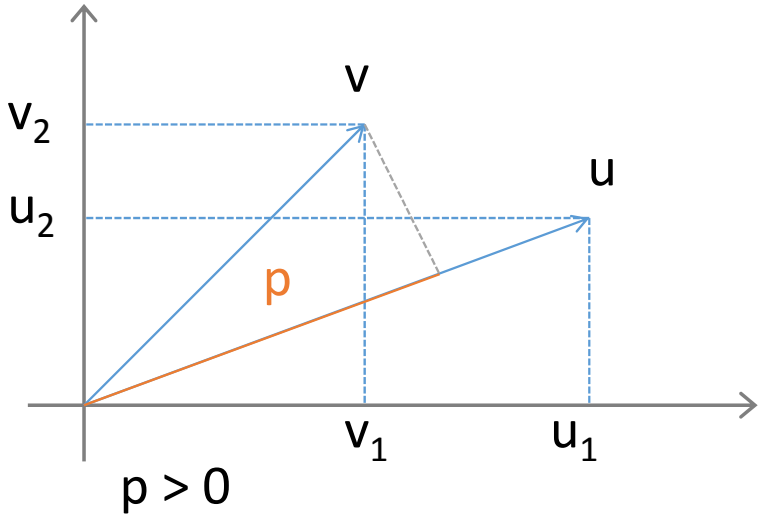
with $A = (1,1)$

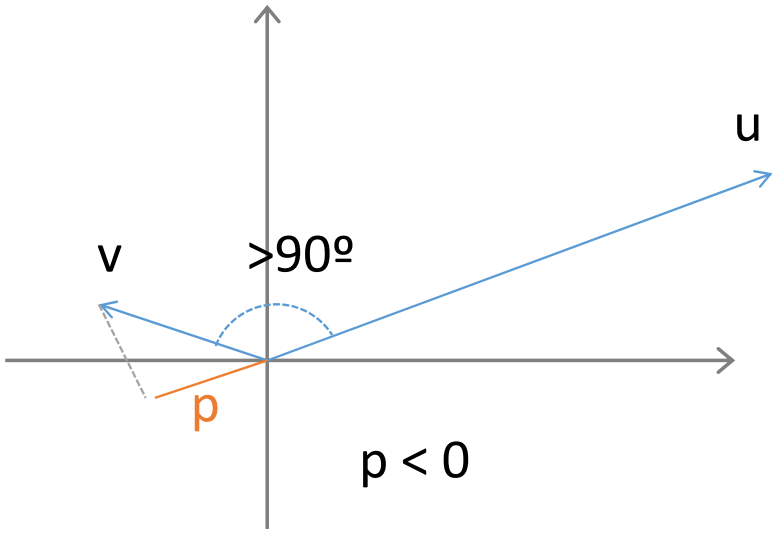


Vector dot Product

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \qquad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$u^T v = u_1 v_1 + u_2 v_2$$





REVIEW SLIDE!

Hypothesis:

Previously: $h_\theta(x) = \theta_0 + \theta_1 x$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

For convenience of notation, define $x_0 = 1$.

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \Re^{n+1} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \Re^{n+1} \qquad h_\theta(x) = \theta^T x$$

Logistic Regression Model

REVIEW SLIDE!

Want $0 \leq h_\theta(x) \leq 1$

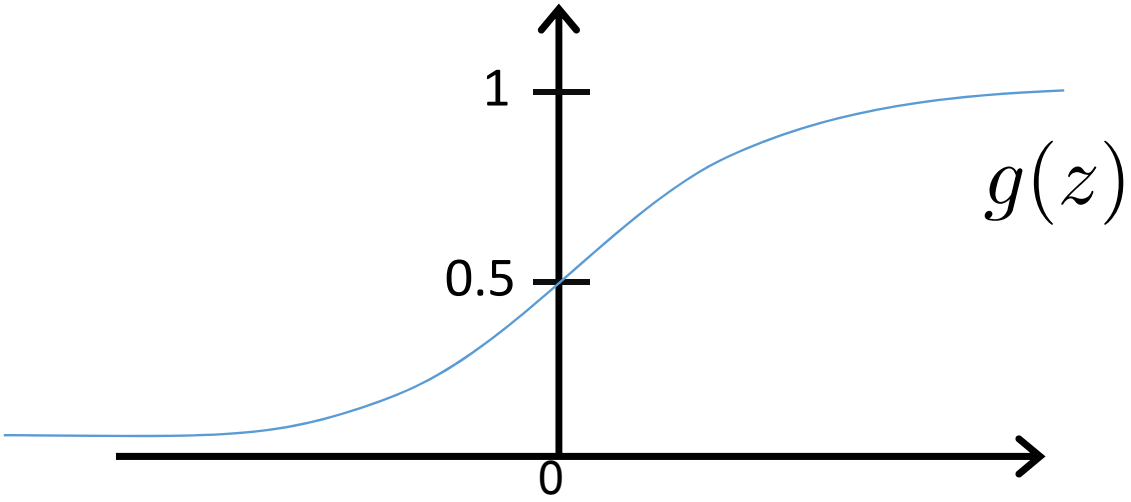
$h_\theta(x) =$?

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

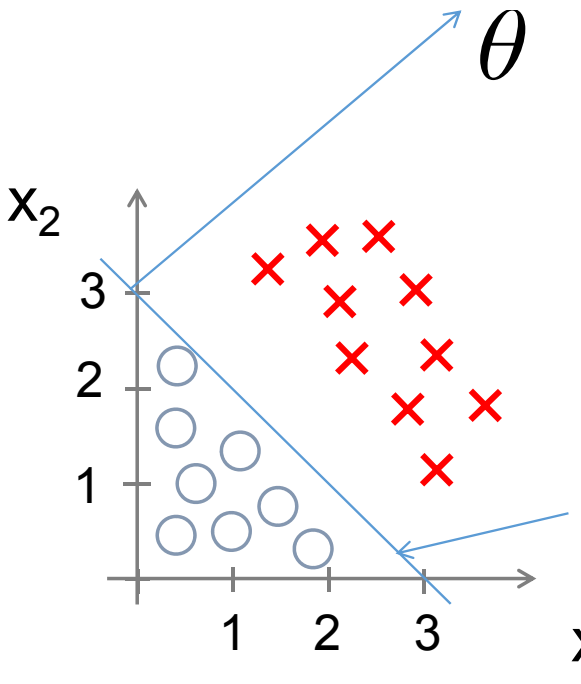
$$g(z) = \frac{1}{1+e^{-z}}$$

Sigmoid function

Logistic function



Decision Boundary in Logistic Regression



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

$$\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

Predict "$y = 1$ " if $-3 + x_1 + x_2 \geq 0$

$\theta^T x$

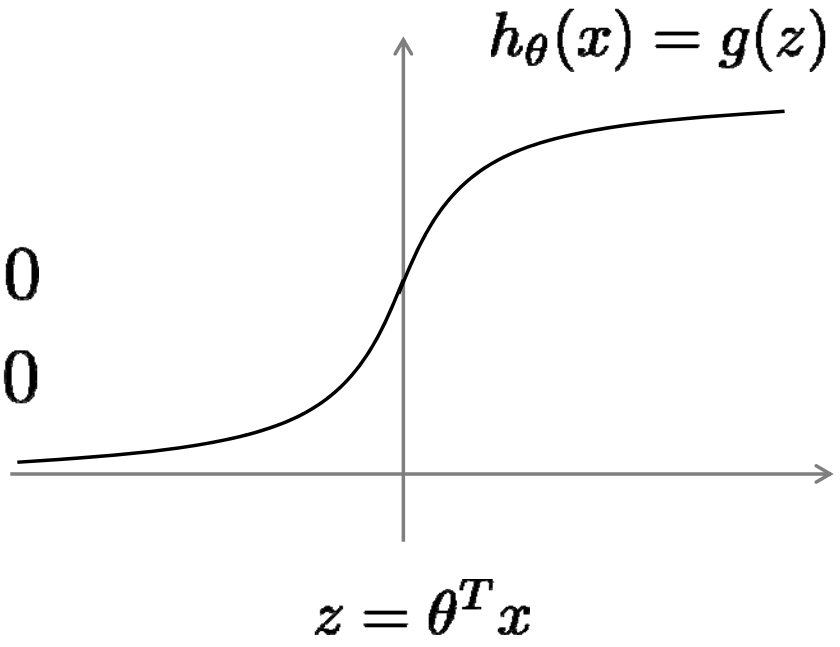
Optimization Objective

Logistic regression hypothesis

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

If $y = 1$, we want $h_\theta(x) \approx 1, \quad \theta^T x \gg 0$

If $y = 0$, we want $h_\theta(x) \approx 0, \quad \theta^T x \ll 0$



SVM hypothesis

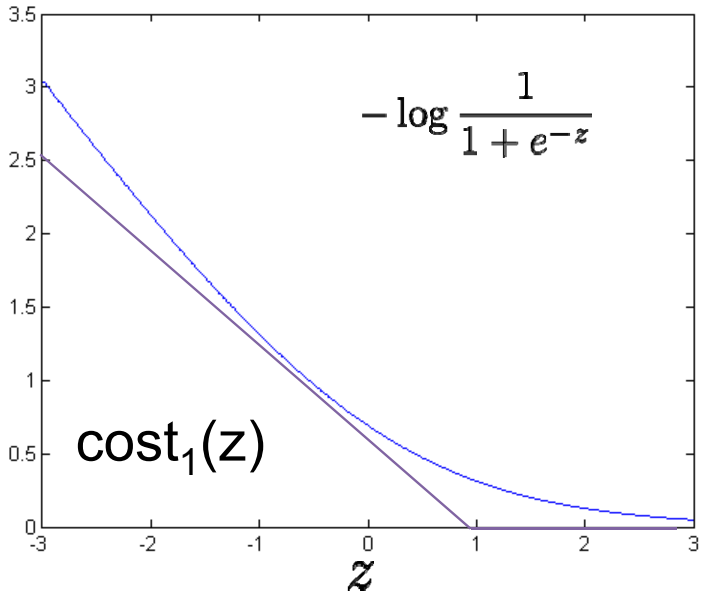
$$h_\theta(x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

SVM gets a similar effect with a different cost function

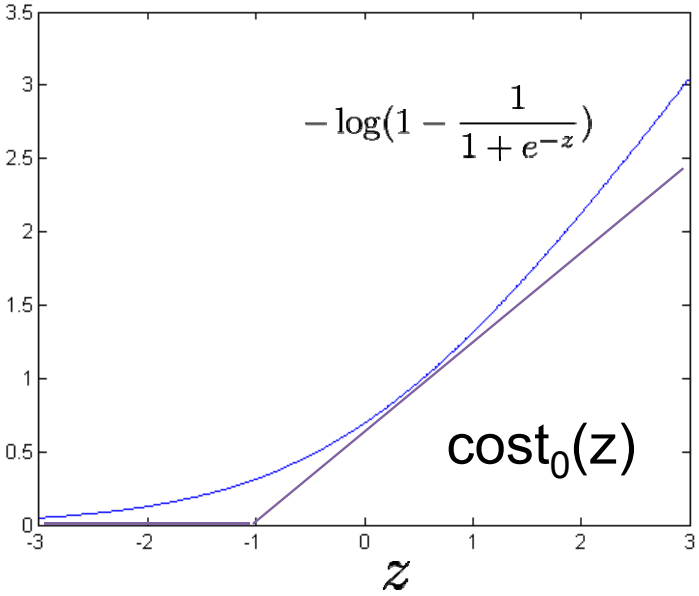
Cost of example: $-(y \log h_\theta(x) + (1-y)\log(1-h_\theta(x)))$

$$= -y \log \frac{1}{1+e^{-\theta^T x}} - (1-y)\log(1 - \frac{1}{1+e^{-\theta^T x}})$$

If $y = 1$ (want $\theta^T x \gg 0$):



If $y = 0$ (want $\theta^T x \ll 0$):



Support vector machine

Logistic regression:

$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\left(-\log h_\theta(x^{(i)})\right) + (1-y^{(i)})\left((-\log(1-h_\theta(x^{(i)}))\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Support vector machine:

$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)})\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

$$\min_{\boldsymbol{\theta}} C\sum_{i=1}^{m}\left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

The mathematics behind it

SVM Decision Boundary

$$\min_{\theta} C \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

If C is very large then we choose θ s.t. the cost term is 0:

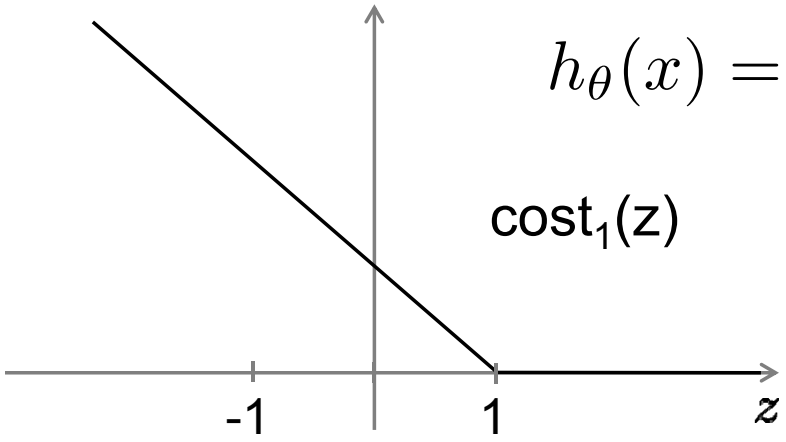
$$\min_{\theta} (C \times 0 + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2)$$

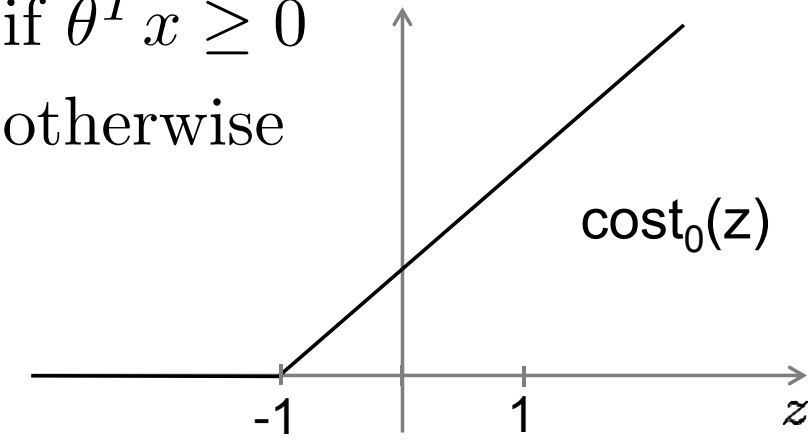
$$\begin{aligned} \text{s.t.} \quad & \theta^T x^{(i)} \geq 1 \quad && \text{if } y^{(i)} = 1 \\ & \theta^T x^{(i)} \leq -1 \quad && \text{if } y^{(i)} = 0 \end{aligned}$$

Support Vector Machine

$$\min_{\theta} C \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

$$h_\theta(x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

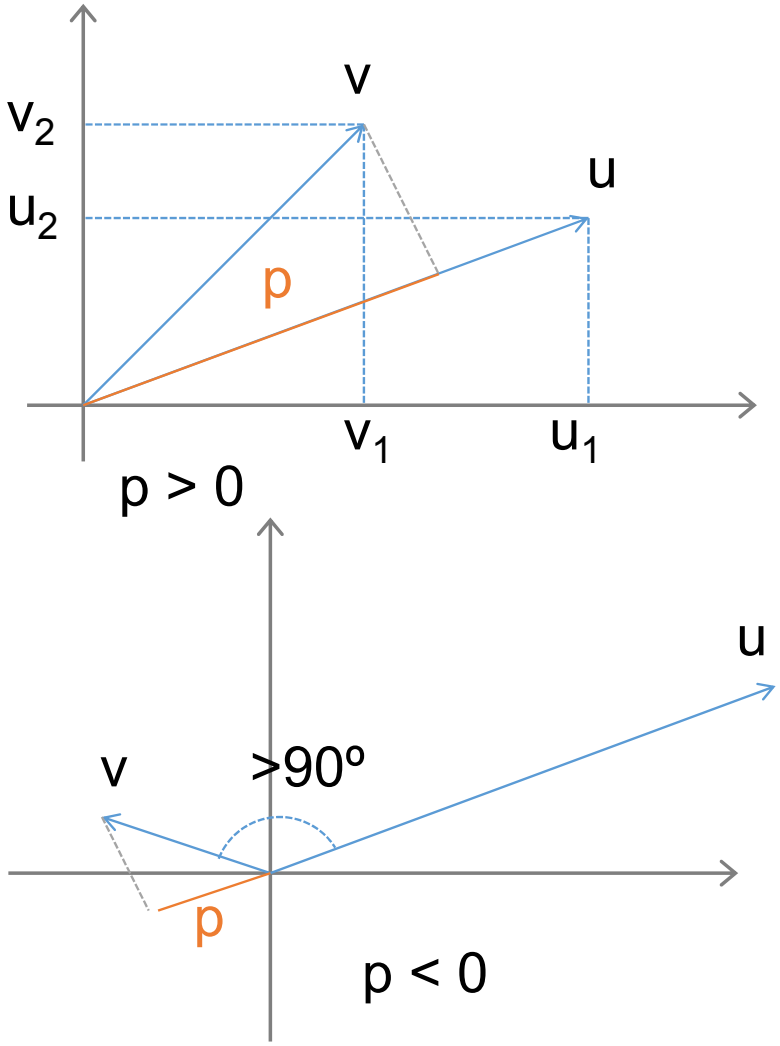




If $y = 1$, we want $\theta^T x \geq 1$ (not just ≥ 0)

If $y = 0$, we want $\theta^T x \leq -1$ (not just < 0)

Vector Inner Product



$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \qquad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$\|u\| = \sqrt{u_1^2 + u_2^2}$$

p: length of projection of v onto u

vector inner product: $u^T v$

$$u^T v = p \cdot \|u\| = u_1 v_1 + u_2 v_2$$

SVM Decision Boundary

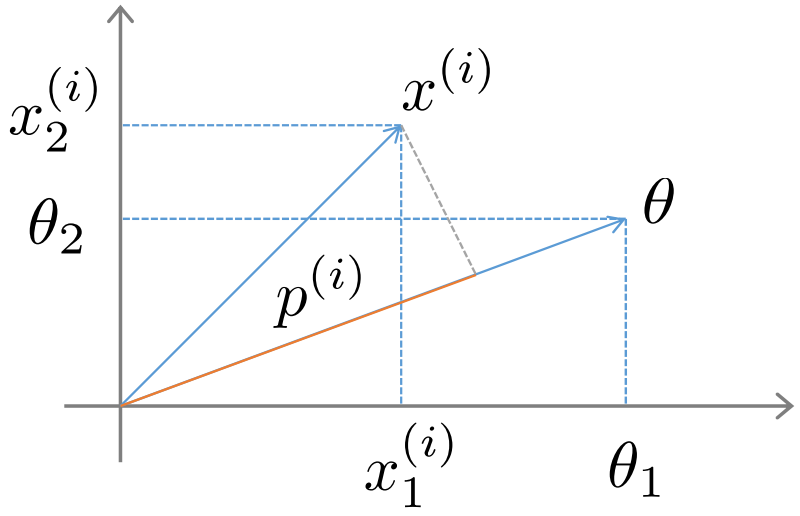
$$\min_{\theta} \frac{1}{2}\sum_{j=1}^{n}\theta_j^2 = \frac{1}{2}(\theta_1^2+\theta_2^2) = \frac{1}{2}\left(\sqrt{\theta_1^2+\theta_2^2}\right)^2 = \frac{1}{2}\|\theta\|^2$$

$$\text{s.t.} \quad \theta^T x^{(i)} \geq 1 \quad \text{if } y^{(i)} = 1$$

$$\theta^T x^{(i)} \leq -1 \quad \text{if } y^{(i)} = 0$$

simplification: $\theta_0 = 0$

n=2



$$\theta^T x^{(i)} = p^{(i)} \cdot \|\theta\|$$

$$= \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)}$$

SVM Decision Boundary

$$\min_{\theta} \frac{1}{2}\sum_{j=1}^{n}\theta_j^2 = \frac{1}{2}\|\theta\|^2$$

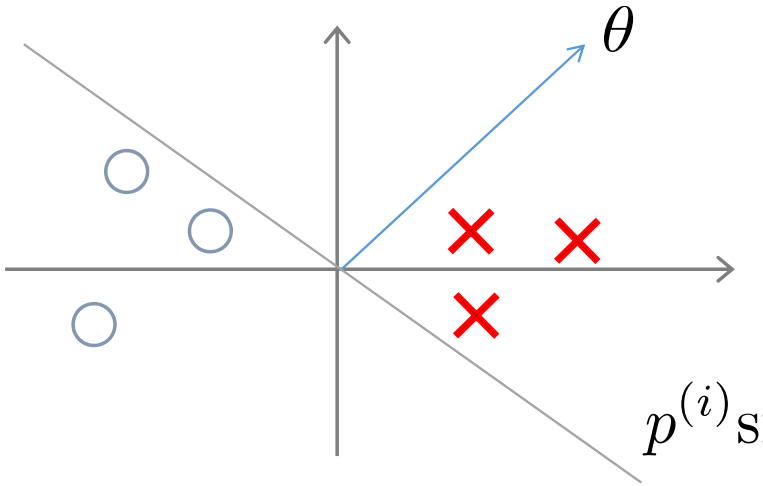
$$\text{s.t.} \quad p^{(i)} \cdot \|\theta\| \geq 1 \quad \text{if } y^{(i)} = 1$$

$$p^{(i)} \cdot \|\theta\| \leq -1 \quad \text{if } y^{(i)} = 0$$

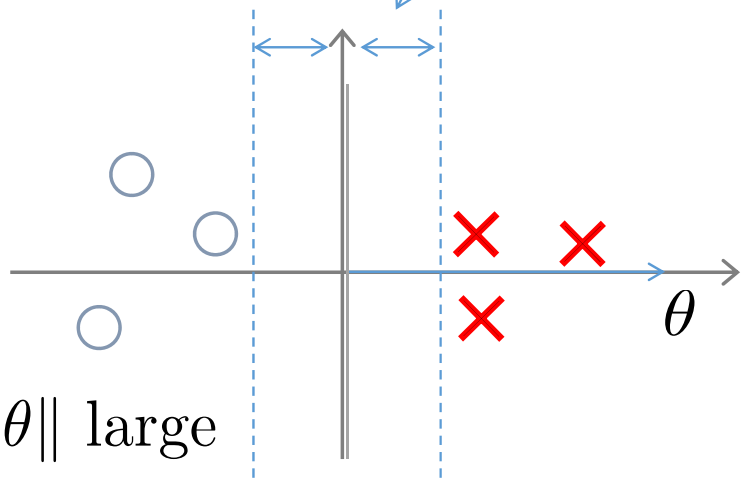
where $p^{(i)}$ is the projection of $x^{(i)}$ onto the vector θ.

Simplification: $\theta_0 = 0$

θ is orthogonal to the decision boundary:



large margin



$p^{(i)}$small $\implies$ $\|\theta\|$ large

Pros and cons

Pros:

- Very fast at classifying new data. No need to go through training data for new classifications.
- Can work for a mixture of categorical and numerical data
- SVMs are “Robust to high dimensionality”, meaning they can work well even with a large number of features.
- Highly accurate.

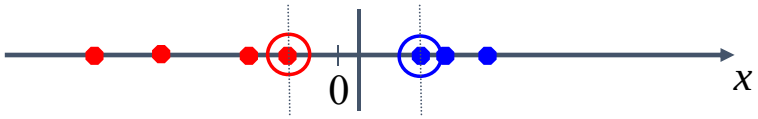
Cons:

- Black box technique. Unlike Bayesian Classifiers and Decision Trees, SVMs don’t offer easily digestible data “under the hood”. A SVM can be a highly effective classifier, but you may not be able to figure out why it makes those classifications.
- SVMs are not online. They will need to be updated every time new training data is to be incorporated.

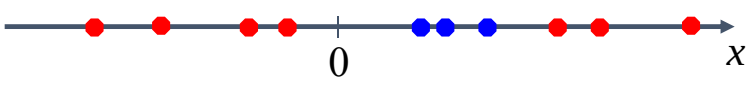
Need for non-linear transformations

Nonlinear SVMs

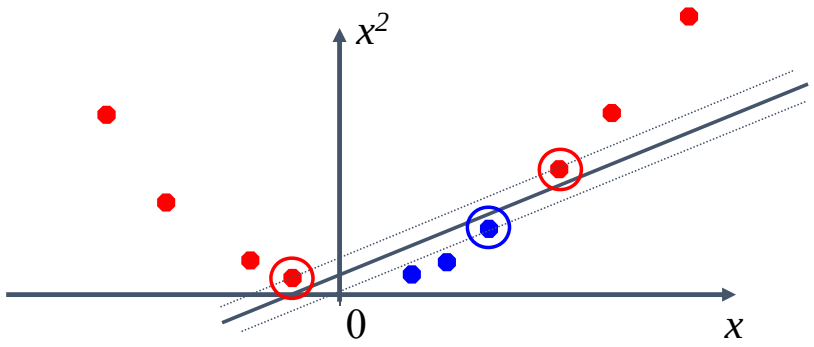
- Datasets that are linearly separable work out great:



- But what if the dataset is just too hard?

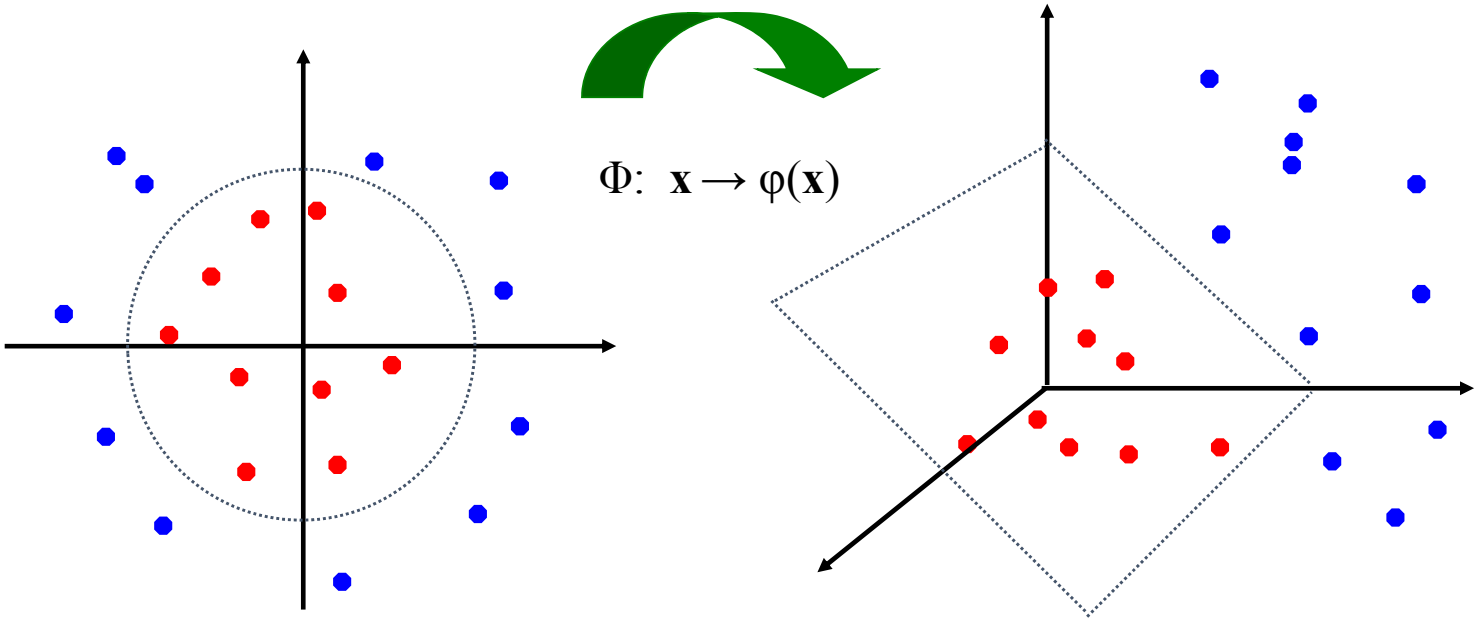


- We can map it to a higher-dimensional space:



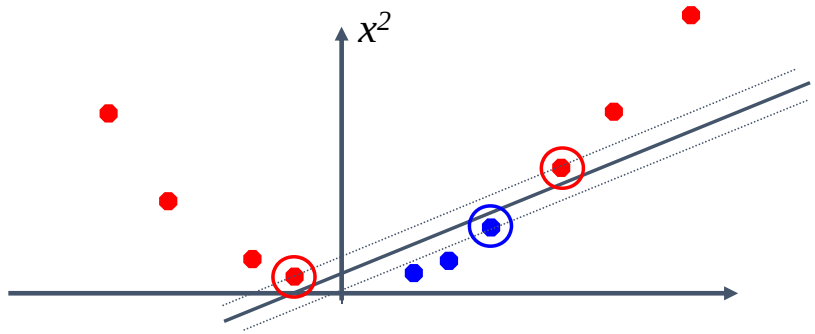
Nonlinear SVMs

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:



Nonlinear kernel: Example

- Consider the mapping $\varphi(x) = (x, x^2)$



$$\varphi(x) \cdot \varphi(y) = (x, x^2) \cdot (y, y^2) = xy + x^2 y^2$$

$$K(x, y) = xy + x^2 y^2$$

Connection with the algorithms

- Many linear parametric models can be recast into an equivalent **dual representation** in which the predictions are based on linear combinations of a **kernel function** evaluated at the training data points

- Kernel $k(x,x') = \Phi(x)^T \Phi(x')$
  - $\Phi(x)$ is a fixed nonlinear feature space mapping
  - Kernel is symmetric of its arguments
    i.e. $k(x,x') = k(x',x)$

Kernel Methods : intuitive idea

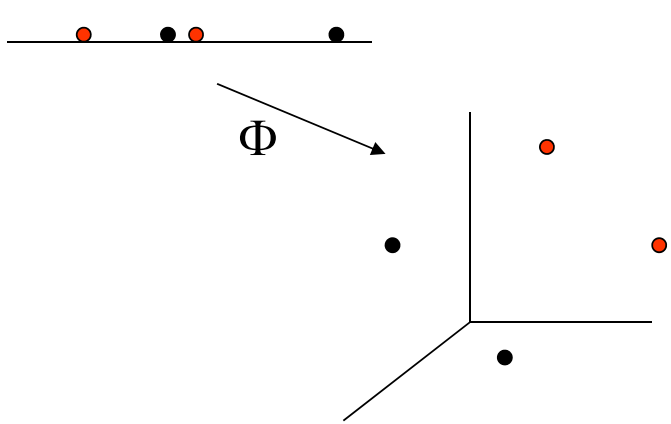
- Find a mapping f such that, in the new space, problem solving is easier (e.g. linear)
- The *kernel* represents the similarity between two objects (documents, terms, …), defined as the dot-product in this new vector space
- But the mapping is left implicit
- Easy generalization of a lot of dot-product (or distance) based pattern recognition algorithms

Feature Spaces

$$\Phi : x \to \Phi(x), \quad R^d \to F$$

non-linear mapping to F

1. high-D space L_2
2. infinite-D countable space :
3. function space (Hilbert space)



example: $(x, y) \to (x^2, y^2, \sqrt{2}xy)$

Kernel Trick

Note: In the dual representation we used the Gram matrix to express the solution.

Kernel Trick:

Replace : $x \rightarrow \Phi(x),$

kernel

$$G_{ij} =< x_i, x_j > \rightarrow G_{ij}^{\Phi} =< \Phi(x_i), \Phi(x_j) >= K(x_i, x_j)$$

If we use algorithms that only depend on the Gram-matrix, G, then we never have to know (compute) the actual features Φ

This is the crucial point of kernel methods

The Kernel Gram Matrix

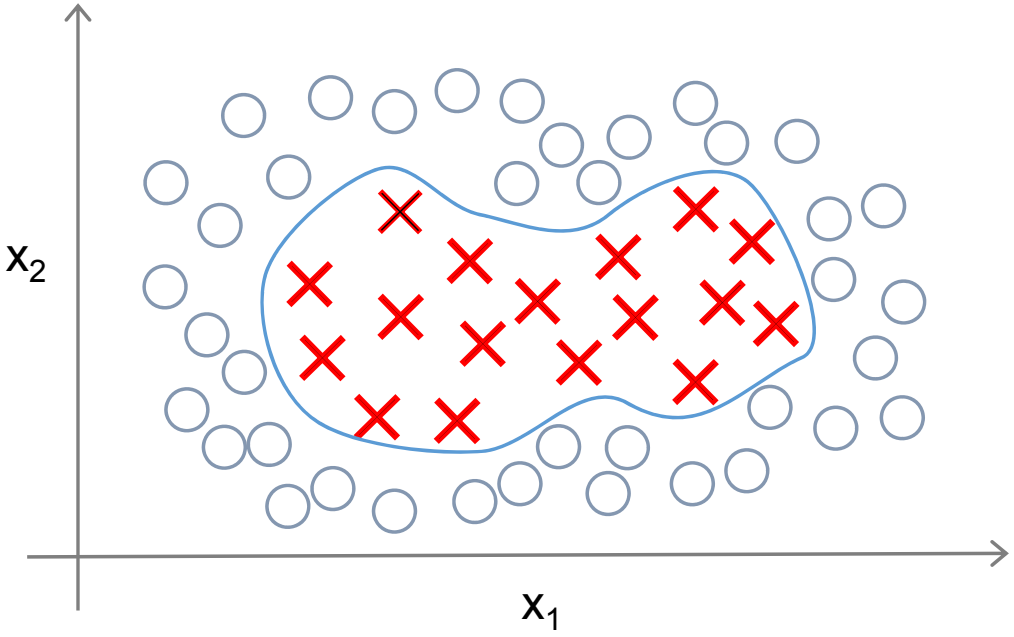
- With KM-based learning, the **sole** information used from the training data set is the Kernel Gram Matrix

$$K_{training} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \dots & k(\mathbf{x}_1, \mathbf{x}_m) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \dots & k(\mathbf{x}_2, \mathbf{x}_m) \\ \dots & \dots & \dots & \dots \\ k(\mathbf{x}_m, \mathbf{x}_1) & k(\mathbf{x}_m, \mathbf{x}_2) & \dots & k(\mathbf{x}_m, \mathbf{x}_m) \end{bmatrix}$$

- If the kernel is valid, K is symmetric definite-positive .

Intuition behind Kernel methods

Non-linear Decision Boundary



$$\text{Predict } y = 1 \text{ if}$$

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \cdots \geq 0$$

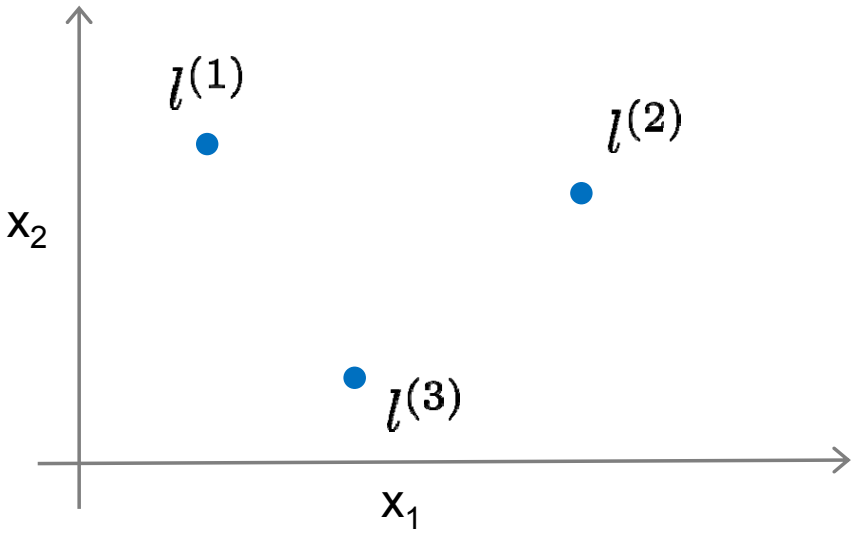
$$h_\theta(x) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \cdots \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \ldots$$

$$f_1 = x_1, f_2 = x_2, f_3 = x_1 x_2, f_4 = x_1^2, f_5 = x_2^2, \ldots$$

Is there a different / better choice of the features $f_1, f_2, f_3, \ldots$?

Kernel



Given x, compute new feature depending on proximity to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$

using for example a gaussian function:

$$f_1 = similarity(x, l^{(1)}) = exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

$$f_2 = similarity(x, l^{(2)}) = exp\left(-\frac{\|x - l^{(2)}\|^2}{2\sigma^2}\right)$$

$$f_3 = similarity(x, l^{(3)}) = exp\left(-\frac{\|x - l^{(3)}\|^2}{2\sigma^2}\right)$$

similarity or “kernel” function: $k(x, l^{(i)})$

Kernel : more formal definition

- A kernel k(x,y)
  - is a similarity measure
  - defined by an implicit mapping f,
  - from the original space to a vector space (feature space)
  - such that: $k(x,y)=f(x)\bullet f(y)$
- This similarity measure and the mapping include:
  - Simpler structure (linear representation of the data)
  - The class of functions the solution is taken from
  - Possibly infinite dimension (hypothesis space for learning)
  - … but still computational efficiency when computing $k(x,y)$

Kernels and Similarity

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(1)})^2}{2\sigma^2}\right)$$

If $x \approx l^{(1)}$:

$$f_1 \approx exp\left(-\frac{0^2}{2\sigma^2}\right) \approx 1$$

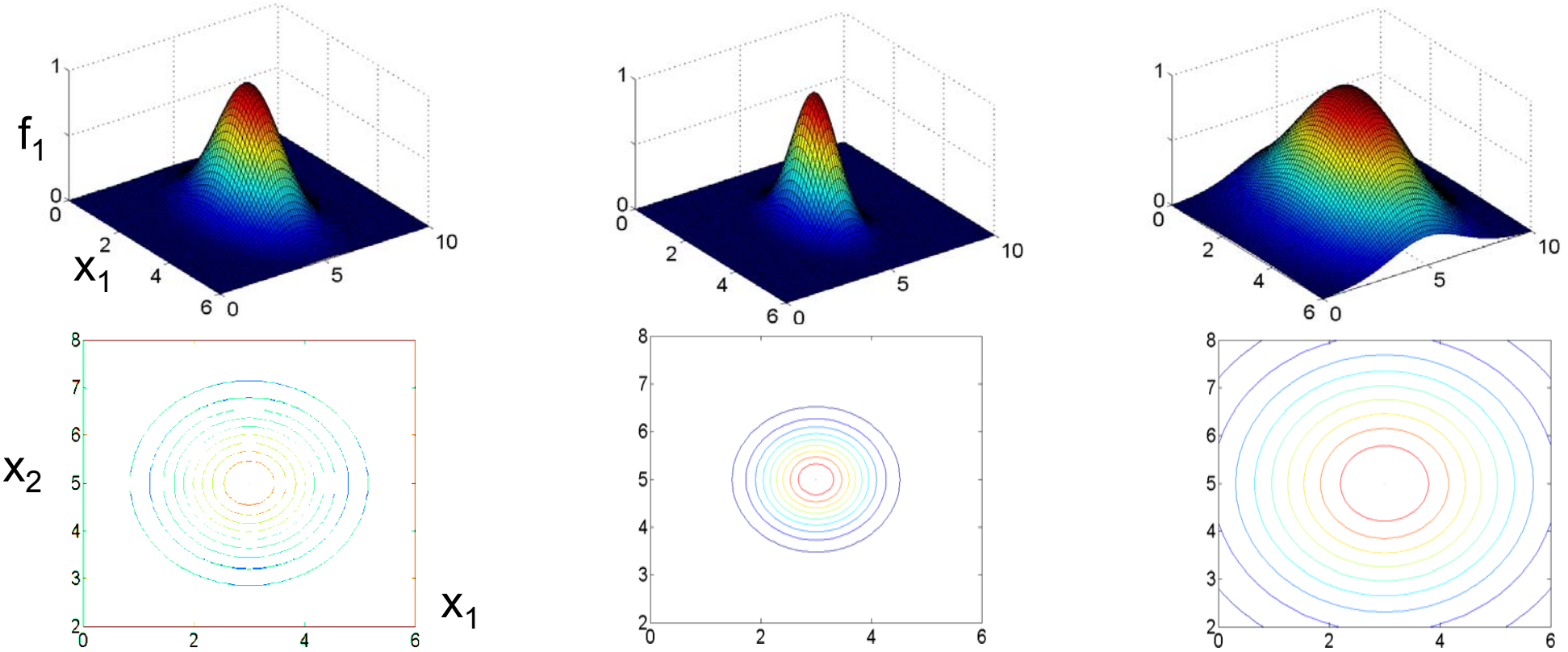
If x if far from $l^{(1)}$:

$$f_1 \approx exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \approx 0$$

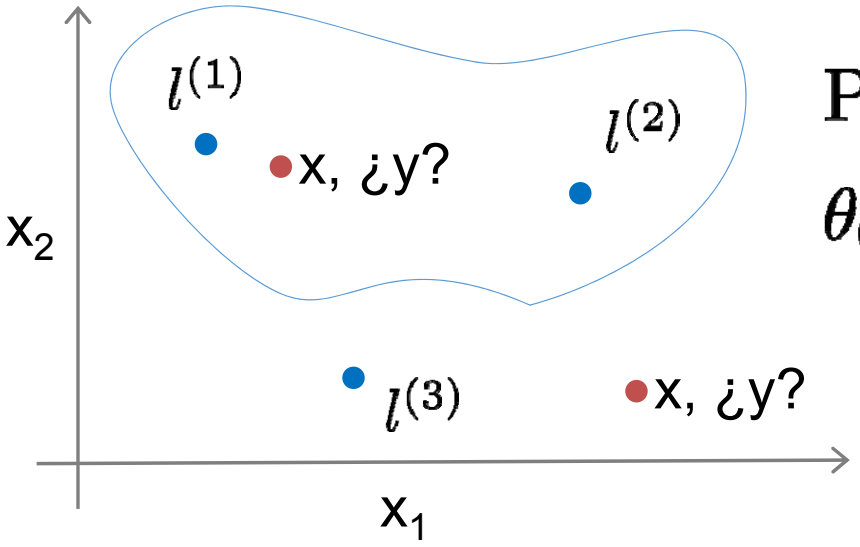
Example:

$$l^{(1)} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \quad f_1 = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

$\sigma^2 = 1$ $\sigma^2 = 0.5$ $\sigma^2 = 3$



Thus we get SVMs with non-linear decision boundaries



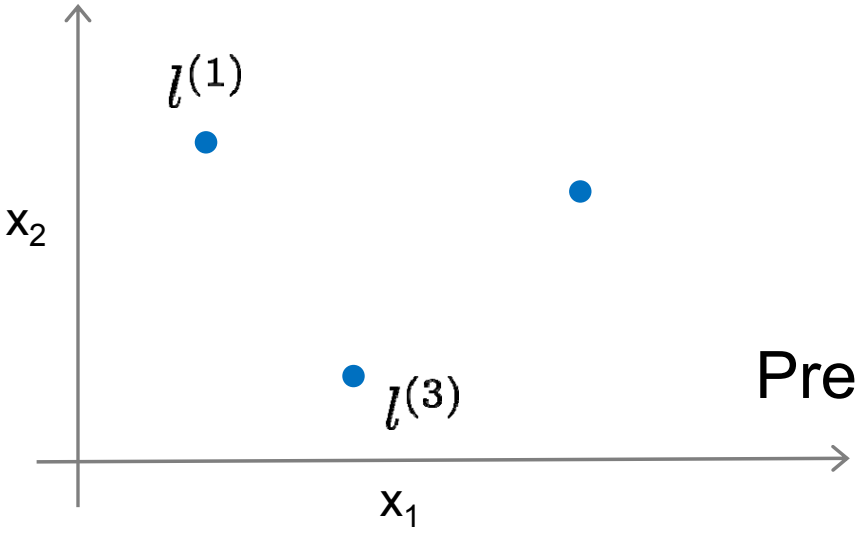
Predict "1" when

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$$

$$\theta_0 = -0.5, \theta_1 = 1, \theta_2 = 1, \theta_3 = 0$$

Predict 1 when x close to $l^{(1)}$ or $l^{(2)}$

Choosing the landmarks

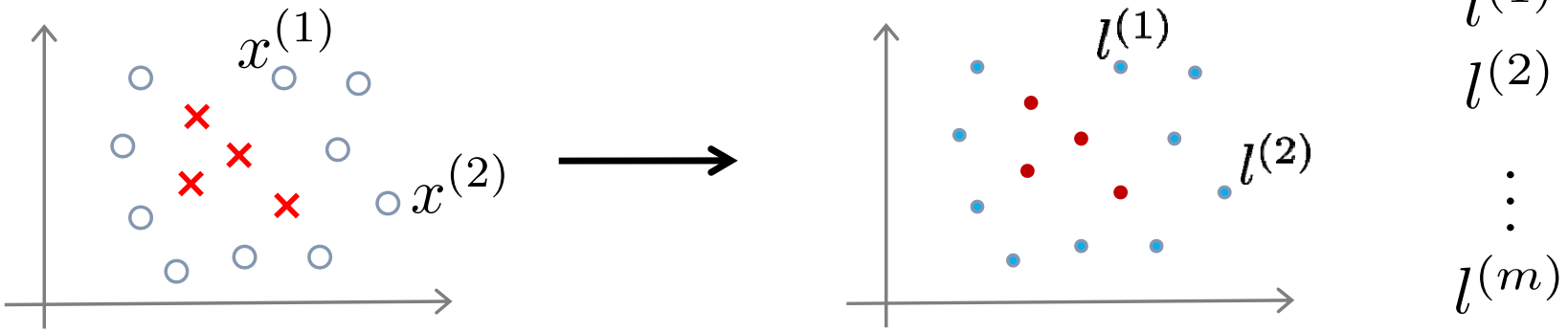


Given x :

$$f_i = \text{similarity}(x, l^{(i)})$$
$$= \exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right)$$

Predict $y = 1$ if $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

Where to get $l^{(1)}, l^{(2)}, l^{(3)}, \ldots$?



SVM with Kernels

Given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$,
choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots, l^{(m)} = x^{(m)}$.

Given example x :

$$f_1 = \text{similarity}(x, l^{(1)})$$
$$f_2 = \text{similarity}(x, l^{(2)})$$
$$\ldots$$

For training example $(x^{(i)}, y^{(i)})$

$$x^{(i)} \quad \begin{aligned} f_1^{(i)} &= sim(x^{(i)}, l^{(1)}) \\ f_2^{(i)} &= sim(x^{(i)}, l^{(2)}) \\ &\vdots \\ f_i^{(i)} &= sim(x^{(i)}, l^{(i)}) = exp\left(-\frac{0}{2\sigma^2}\right) = 1 \\ &\vdots \\ f_m^{(i)} &= sim(x^{(i)}, l^{(m)}) \end{aligned}$$

$$f^{(i)} = \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ f_2^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix}$$

$$f_0^{(i)} = 1$$

SVM with Kernels

Hypothesis: Given x, compute features $f \in \mathbb{R}^{m+1}$

Predict "y=1" if $\theta^T f \geq 0$

Training: *n=m*

$$\min_\theta C \sum_{i=1}^{m} y^{(i)} cost_1(\theta^T f^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

For efficiency reasons ('m' can be very big) most software packages minimize a modified version of this:

$$\sum_{j=1}^{n} \theta_j^2 = \theta^T \theta \quad \longrightarrow \quad \theta^T M \theta$$

and do not return θ but a *model* that can be used to make predictions

SVM parameters:

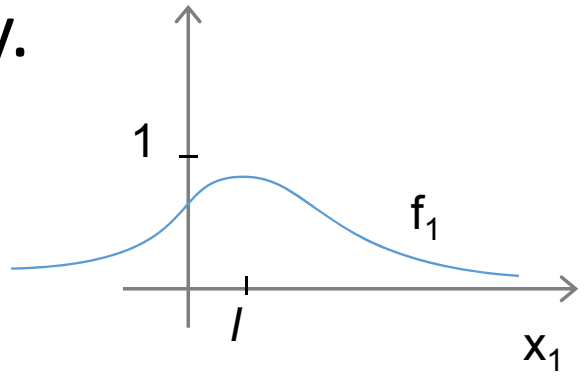
C ($= \frac{1}{\lambda}$). Large C (small λ): Lower bias, high variance.

Small C (large λ): Higher bias, low variance.

σ^2 Large σ^2: Features f_i vary more smoothly.

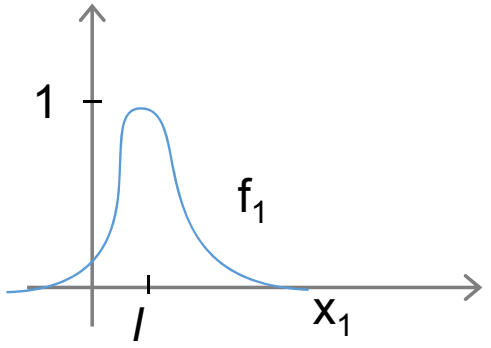
Higher bias, lower variance.

$$exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$$



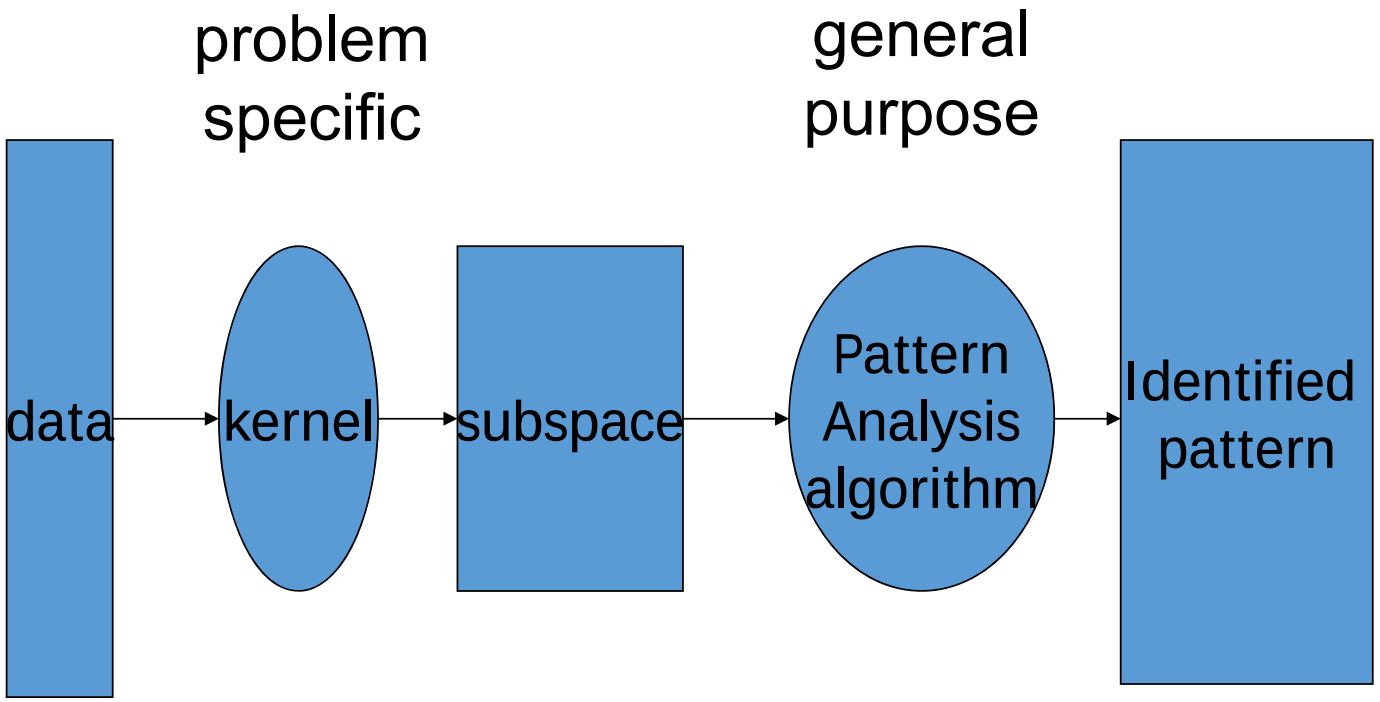
Small σ^2: Features f_i vary less smoothly.

Lower bias, higher variance.



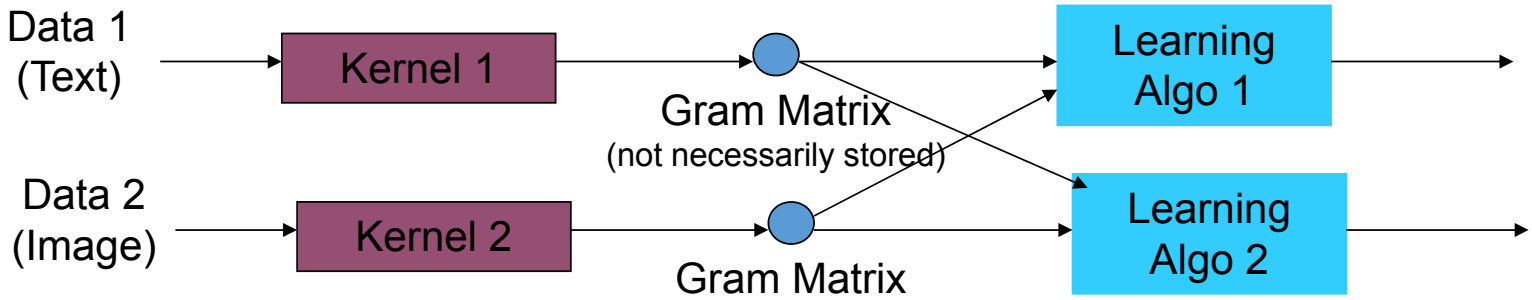
Overall structure

Kernel methods: plug and play



Modularity and re-usability

- Same kernel ,different learning algorithms
- Different kernels, same learning algorithms



SVMs for image classification

1. Pick an image **representation** (in our case, bag of features)
2. Pick a **kernel function** for that representation
3. Compute the **matrix of kernel values** between every pair of training examples
4. Feed the kernel matrix into your favorite **SVM solver** to obtain **support vectors** and **weights**
5. At test time: compute kernel values for your test example and each support vector, and combine them with the learned weights to get the value of the decision function

SVM in Python

Google “Implementing SVM Kernel SVM Python Scikit-Learn”

https://stackabuse.com/
implementing-svm-and-kernel-svm-with-pythons-scikit-learn/

SVM + different kernels on Iris data set

Code example 2: `kernel-iris-polynomial.py`

Code example 3: `kernel-iris-gaussian.py`

Code example 4: `kernel-iris-sigmoid.py`

SVM in Python

Google “In-Depth Support Vector Machines”

https://jakevdp.github.io/PythonDataScienceHandbook/
05.07-support-vector-machines.html

Code example 2:	`kernel-in-depth-0.py`	Plot circles data set
Code example 3:	`kernel-in-depth-1.py`	Plot linear SVC on data
Code example 4:	`kernel-in-depth-2.py`	Extra dimension
Code example 5:	`kernel-in-depth-3.py`	Plot rbf SVC

Soft margin SVM in Python

Google “In-Depth Support Vector Machines”

https://jakevdp.github.io/PythonDataScienceHandbook/
05.07-support-vector-machines.html

Code example 2: `kernel-in-depth-4.py` Plot overlapping blobs data set

Code example 3: `kernel-in-depth-5.py` Plot linear SVC different C

Using an SVM

Use SVM software package (e.g. liblinear, libsvm, …) to solve for parameters θ

Need to specify:

Choice of parameter C.

Choice of kernel (similarity function):

E.g. No kernel (“linear kernel”) (useful when ‘n’ is large and ‘m’ is small)

Predict “y = 1” if $\theta^T x \geq 0$

Gaussian kernel:

$f_i = \exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right)$, where $l^{(i)} = x^{(i)}$

Need to choose σ^2

Note: Do perform feature scaling (if needed) before using the Gaussian kernel

$$\|x - l\|^2 = (x_1 - l_1)^2 + (x_2 - l_2)^2 + \cdots + (x_n - l_n)^2$$

without feature scaling large attributes will subsume smaller ones

Other choices of kernel

Note: Not all similarity functions $\mathbf{similarity}(x, l)$ make valid kernels.
(Need to satisfy technical condition called "Mercer's Theorem" to make sure SVM packages' optimizations run correctly, and do not diverge).

Many off-the-shelf kernels available:

- Polynomial kernel:

$$k(x, l) = (x^T l + \text{constant})^{\text{degree}}$$

- More esoteric: String kernel, chi-square kernel, histogram intersection kernel, …

Issues in applying SVMs

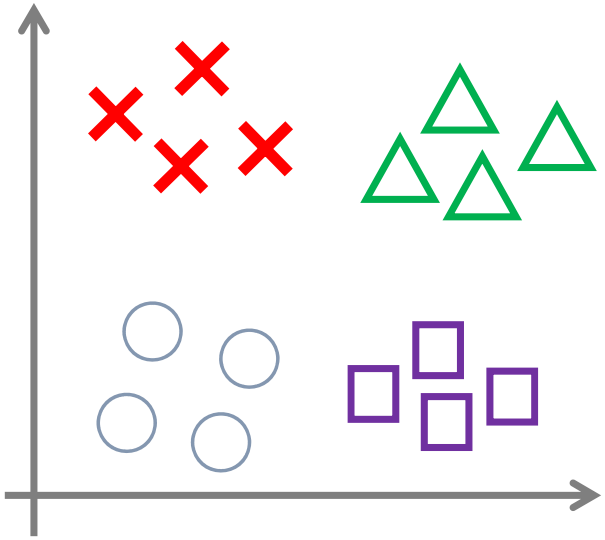
- Need to choose a kernel:
  - Standard inner product
  - Polynomial kernel – how to choose degree
  - Gaussian kernel – but how to choose width
  - Specific kernel for different datatypes
- Need to set parameter C:
  - Can use cross-validation
  - If data is normalised often standard value of 1 is fine

Multi-class

What about multi-class SVMs?

- Unfortunately, there is no “definitive” multi-class SVM formulation
- In practice, we have to obtain a multi-class SVM by combining multiple two-class SVMs
- One vs. others
  - Traning: learn an SVM for each class vs. the others
  - Testing: apply each SVM to test example and assign to it the class of the SVM that returns the highest decision value
- One vs. one
  - Training: learn an SVM for each pair of classes
  - Testing: each learned SVM “votes” for a class to assign to the test example

Multi-class classification



$$y \in \{1, 2, 3, \ldots, K\}$$

Many SVM packages already have built-in multi-class classification functionality

Otherwise, use one-vs.-all method. (Train K SVMs, one to distinguish $y = i$ from the rest, for $i = 1, 2, \ldots, K$), get $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$

Pick class i with largest $(\theta^{(i)})^T x$

Logistic regression vs. SVMs

$n =$ number of features ($x \in \mathbb{R}^{n+1}$), $m =$ number of training examples

If n is large relative to m (e.g., n =10.000, m = 1.000):
Use logistic regression, or SVM without a kernel (“linear kernel”)

If n is small, m is intermediate (e.g., n=1 .. 1000, m=10 .. 10.000):
Use SVM with Gaussian kernel

If n is small, m is large (e.g., n=1 .. 1000, m=50.000+):
Create/add more features, then use logistic regression or SVM without a kernel (Gaussian kernel too slow with m so large)

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