Machine Learning, Machine Learning (extended)

8 – Supervised Learning: Discriminative Classification

Kashif Rajpoot

k.m.rajpoot@cs.bham.ac.uk

School of Computer Science

University of Birmingham

Outline

- Supervised learning
- Discriminative classification
    - Decision boundary
    - The margin
- Maximizing the margin
- Making predictions
- Support vectors
- Hard margin
- Soft margin
- Non-linear decision boundary
- Kernel trick

Supervised learning

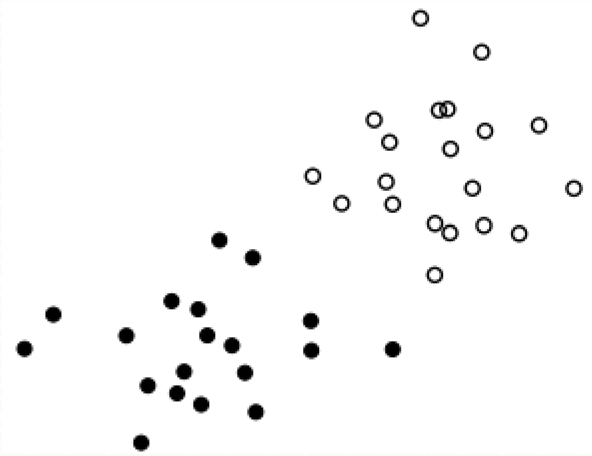
- Regression
  - Minimised loss (e.g. least squares)
  - Maximum likelihood
- Classification
  - Generative (e.g. Bayesian)
  - Instance-based (e.g. k-NN)
  - Discriminative (e.g. SVM)

Classification

- A set of N objects with attributes (usually vector) $\boldsymbol{x}_n$
- Each object has an associated target label t_n
- Binary classification

  $t_n \in \{0,1\}$ or $t_n \in \{-1,1\}$
- Multi-class classification

  $t_n \in \{1,2,\dots,C\}$



- Classifier learns from $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N$ and $t_1, t_2, \dots, t_N$ so that it can later classify $\boldsymbol{x}_{new}$

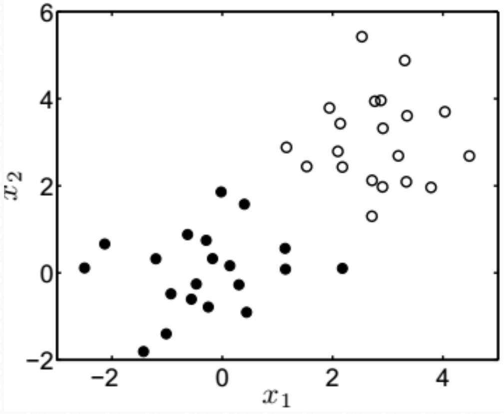
Generative vs discriminative classification

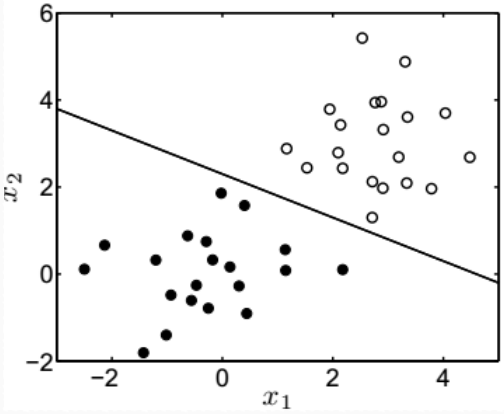
- Generative classifiers generate a model for each class, based on training samples available
  - Data in each class can be seen as *generated* by some model
  - For new test samples, they assign these samples to the class that suits best (e.g. by probability measure)

- In contrast, discriminative classifiers attempt to explicitly define the decision boundary that separates the classes
  - Intuitively, these methods are for binary class problems but can be extended to multi-class problems

Support vector machines

- Let's consider a 2-d example where a model needs to learn classification
- Let's consider model as a linear decision boundary (i.e. straight line) that separates the two classes
- SVM is a binary classifier that learns a linear decision boundary from attributes $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ and target labels $t_1, t_2, \ldots, t_N$, with $t_n \in \{-1,1\}$
- In n-d, SVM is a discriminant classifier that determines a *linear hyperplane*
- SVM is a very popular classifier in bioinformatics, medical imaging, digit classification, and various other areas





Line: refresher

- What's the equation of a straight line?
  - $y = mx + c \Rightarrow \boldsymbol{w}^T\boldsymbol{x} + b = 0$?
    - $\boldsymbol{w}$? b?
  - $y = -\frac{1}{4}x + 3 \Rightarrow \boldsymbol{w}^T\boldsymbol{x} + b = 0$
    - $\boldsymbol{w}$? b?

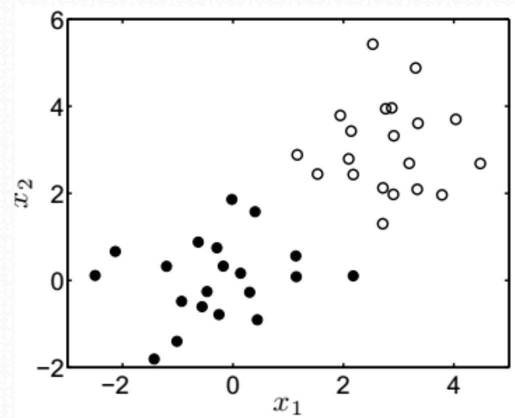
- For what points: $\boldsymbol{w}^T\boldsymbol{x} + b > 0$?
- For what points: $\boldsymbol{w}^T\boldsymbol{x} + b < 0$?

- For what points: $\boldsymbol{w}^T\boldsymbol{x} + b = 1$?
- For what points: $\boldsymbol{w}^T\boldsymbol{x} + b = -1$?

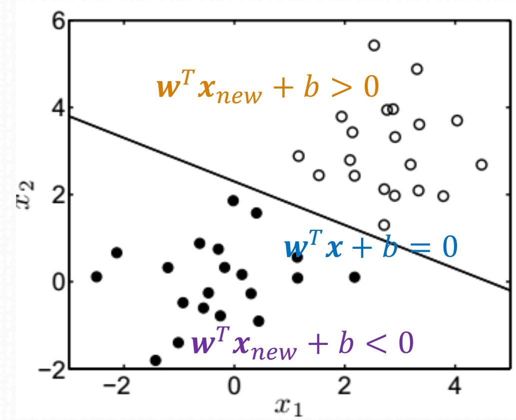
- Relation of $\boldsymbol{w}$ to the straight line $\boldsymbol{w}^T\boldsymbol{x} + b = 0$?
  - For example: consider $y = 2x$

Decision boundary

- Linear decision boundary can be represented as a straight line
  - $\boldsymbol{w}^T\boldsymbol{x} + b = 0$
- To classify a new test sample $\boldsymbol{x}_{new}$:
  - $t_{new} = 1$: if $\boldsymbol{w}^T\boldsymbol{x}_{new} + b > 0$
  - $t_{new} = -1$: if $\boldsymbol{w}^T\boldsymbol{x}_{new} + b < 0$

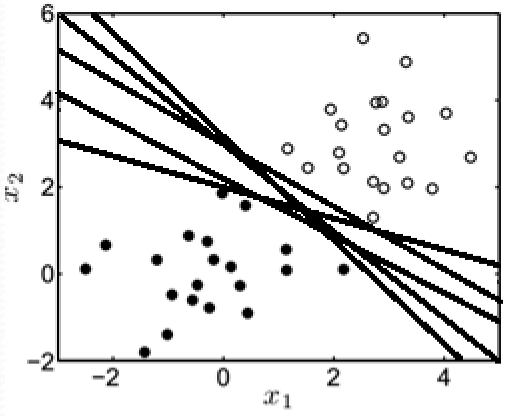


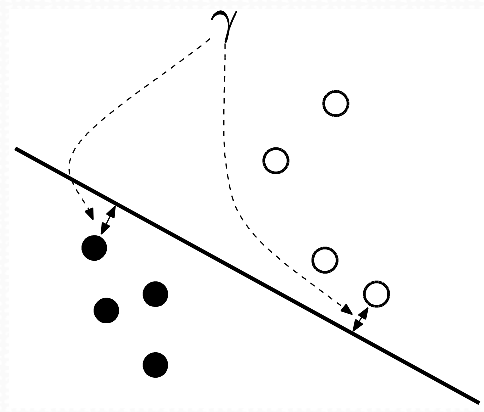
- The decision function (prediction) becomes:
  - $t_{new} = sign(\boldsymbol{w}^T\boldsymbol{x}_{new} + b)$
- Decision boundary is determine by $\boldsymbol{w}$ and b
  - How to choose $\boldsymbol{w}$ and b?



The margin

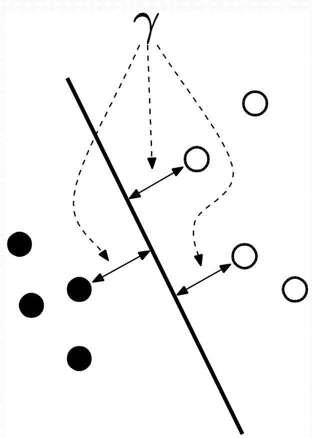
- Given linearly separable two class data, there are *infinite* number of straight lines that can separate it
  - Which should we choose?
- According to *learning theory*, a decision boundary that maximizes the margin of the boundary to the training set is the one that will minimize the generalization error
  - Margin: perpendicular distance (γ) between the boundary and closest training points of each class
- SVM finds the decision boundary that maximizes the margin
- How to choose $\boldsymbol{w}$ and b?
  - Optimize the margin γ



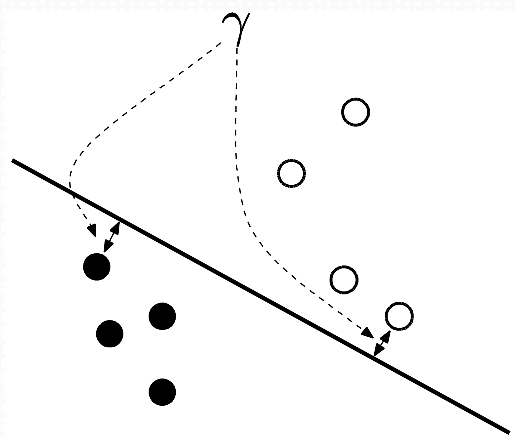


Maximizing the margin

- Maximize the *perpendicular distance* from the decision boundary to the closest points on each side
  - Maximum margin decision boundary better reflects the data characteristics than non-optimal boundary
  - Maximum margin decision boundary classifier generalizes well on unseen test data



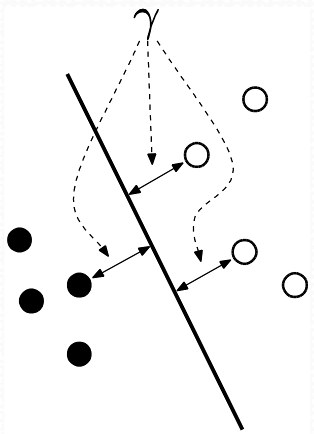
(a) The decision boundary that maximises the margin



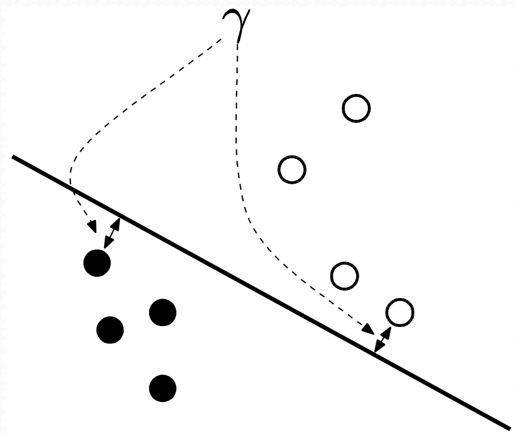
(b) A non-optimal decision boundary

Maximizing the margin

- From all possible linear decision boundaries, the one that maximizes the margin on the training set will minimize the generalization error
  - subject to have seen “enough” training samples and assuming that data isn’t “noisy”



(a) The decision boundary that maximises the margin

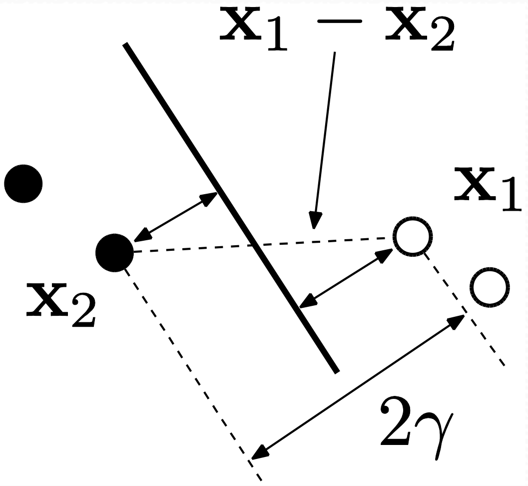


(b) A non-optimal decision boundary

Maximizing the margin

- Let's consider two closest points to the boundary: $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$

- Double margin (2γ) is equal to the component of vector joining $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ in the direction perpendicular to the boundary
  - $\boldsymbol{x}_1 - \boldsymbol{x}_2$ is the vector joining $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$
  - $\boldsymbol{w}/||\boldsymbol{w}||$ is the direction perpendicular to the boundary
  - Thus $2\gamma = \frac{1}{||\boldsymbol{w}||}\boldsymbol{w}^T(\boldsymbol{x}_1 - \boldsymbol{x}_2)$

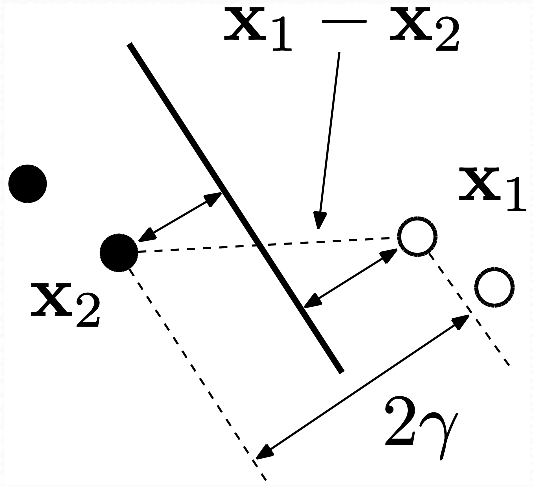


Maximizing the margin

- Double margin can be estimated as:

$$2\gamma = \frac{1}{||\boldsymbol{w}||}\boldsymbol{w}^T(\boldsymbol{x}_1 - \boldsymbol{x}_2)$$

- Decision function $t_{new} = sign(\boldsymbol{w}^T\boldsymbol{x}_{new} + b)$ is invariant to scaling its argument by a positive constant λ
  - $sign(\boldsymbol{w}^T\boldsymbol{x}_{new} + b) = sign(\lambda\boldsymbol{w}^T\boldsymbol{x}_{new} + \lambda b)$



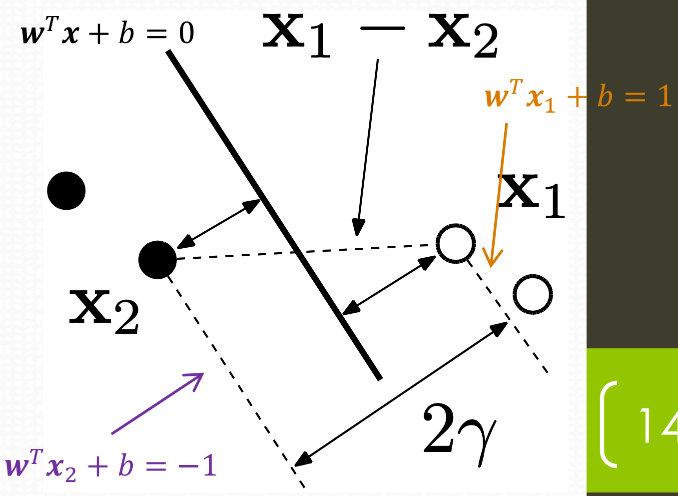
- Let's set the scale such that:
  - $\boldsymbol{w}^T\boldsymbol{x}_1 + b = 1$
  - $\boldsymbol{w}^T\boldsymbol{x}_2 + b = -1$

Maximizing the margin

- Considering:
  - $\boldsymbol{w}^T \boldsymbol{x}_1 + b = 1$ (line parallel to decision boundary)
  - $\boldsymbol{w}^T \boldsymbol{x}_2 + b = -1$ (line parallel to decision boundary)

- By subtracting the above two equations:
  - $(\boldsymbol{w}^T \boldsymbol{x}_1 + b) - (\boldsymbol{w}^T \boldsymbol{x}_2 + b) = 1 - (-1)$
  - $\boldsymbol{w}^T (\boldsymbol{x}_1 - \boldsymbol{x}_2) = 2$

- Thus:
  - $2\gamma = \frac{1}{\|\boldsymbol{w}\|} \boldsymbol{w}^T (\boldsymbol{x}_1 - \boldsymbol{x}_2) = \frac{1}{\|\boldsymbol{w}\|} 2$
  - $\gamma = \frac{1}{\|\boldsymbol{w}\|}$



Maximizing the margin

- SVM maximizes $2\gamma = \frac{2}{\|\boldsymbol{w}\|}$
  - Equivalent to minimize $\frac{\|\boldsymbol{w}\|}{2}$
  - Equivalent to minimize(due to mathematical simplicity) $\frac{1}{2}\|\boldsymbol{w}\|^2 = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$
- There are constraints (to prevent training samples falling in margin band):
  - For $\boldsymbol{x}_n$ with $t_n = 1$: $\boldsymbol{w}^T\boldsymbol{x}_n + b \geq 1$
  - For $\boldsymbol{x}_n$ with $t_n = -1$: $\boldsymbol{w}^T\boldsymbol{x}_n + b \leq -1$
- This can be expressed as:
  - $t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1$
- This is why using $t_n \in \{-1,1\}$ is beneficial over using $t_n \in \{0,1\}$

Maximizing the margin

- SVM optimization problem is:

$$\underset{\boldsymbol{w}}{argmin} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$$

subject to constraint $t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1$

- By the use of Lagrange multipliers (α_n), the constraints can be expressed in the minimization function (beyond our module scope):

$$\underset{\boldsymbol{w},\alpha}{argmin}\, \mathcal{L} = \underset{\boldsymbol{w},\alpha}{argmin} \left[\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} - \sum_{n=1}^{N} \alpha_n \left(t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) - 1\right)\right]$$

subject to $\alpha_n \geq 0$

Maximizing the margin

- To find the minimum of minimization function $\mathcal{L}$, take the 1st derivative with respect to $\boldsymbol{w}$ and b and set to $\mathbf{0}$:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{n=1}^{N} \alpha_n \, t_n \boldsymbol{x}_n = 0$$

$$\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n \, t_n \boldsymbol{x}_n$$

and

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{n=1}^{N} \alpha_n \, t_n = 0$$

$$\sum_{n=1}^{N} \alpha_n \, t_n = 0$$

- α_n?

Let's recall..

$f(\mathbf{w})$	$\frac{\partial f}{\partial \mathbf{w}}$
$\mathbf{w}^\mathsf{T}\mathbf{x}$	$\mathbf{x}$
$\mathbf{x}^\mathsf{T}\mathbf{w}$	$\mathbf{x}$
$\mathbf{w}^\mathsf{T}\mathbf{w}$	$2\mathbf{w}$
$\mathbf{w}^\mathsf{T}\mathbf{C}\mathbf{w}$	$2\mathbf{C}\mathbf{w}$

Maximizing the margin

- By substituting $\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n t_n \boldsymbol{x}_n$ in the SVM minimization function:

$$\underset{\boldsymbol{w},\alpha}{argmin} \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w} - \sum_{n=1}^{N} \alpha_n \left(t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) - 1\right)$$

we obtain:

$$\underset{\alpha}{argmin} \frac{1}{2} \left(\sum_{m=1}^{N} \alpha_m t_m \boldsymbol{x}_m^T \right) \left(\sum_{n=1}^{N} \alpha_n t_n \boldsymbol{x}_n \right) - \sum_{n=1}^{N} \alpha_n \left(t_n \left(\sum_{m=1}^{N} \alpha_m t_m \boldsymbol{x}_m^T \boldsymbol{x}_n + b \right) - 1 \right)$$

$$\underset{\alpha}{argmin} \frac{1}{2} \sum_{m=1}^{N} \sum_{n=1}^{N} \alpha_m \alpha_n t_m t_n \boldsymbol{x}_m^T \boldsymbol{x}_n - \sum_{m=1}^{N} \sum_{n=1}^{N} \alpha_m \alpha_n t_m t_n \boldsymbol{x}_m^T \boldsymbol{x}_n - \sum_{n=1}^{N} \alpha_n t_n b + \sum_{n=1}^{N} \alpha_n$$

$$\boxed{\sum_{n=1}^{N} \alpha_n t_n = 0}$$

$$\underset{\alpha}{argmin} \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m=1}^{N} \sum_{n=1}^{N} \alpha_m \alpha_n t_m t_n \boldsymbol{x}_m^T \boldsymbol{x}_n$$

Maximizing the margin

- SVM optimization function becomes:

$$\underset{\alpha}{argmin} \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_m \alpha_n \, t_m t_n \boldsymbol{x}_m^T \boldsymbol{x}_n$$

subject to

$$\alpha_n \geq 0$$

and

$$\sum_{n=1}^{N} \alpha_n t_n = 0$$

- Where are $\boldsymbol{w}$ and b in optimization?
- This is a standard quadratic programming optimization problem (beyond our module scope) which can be solved numerically in MATLAB (*quadprog*)
    - There is no analytical solution

Making predictions

- Let's recall that target label t_{new} for a new test sample $\boldsymbol{x}_{new}$ can be predicted as:

$$t_{new} = sign(\boldsymbol{w}^T \boldsymbol{x}_{new} + b)$$

- Do we know $\boldsymbol{w}$ and b?

- Let's recall $\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n t_n \, \boldsymbol{x}_n$, so we get:

$$t_{new} = sign\left(\sum_{n=1}^{N} \alpha_n t_n \, \boldsymbol{x}_n^T \boldsymbol{x}_{new} + b\right)$$

- Do we know b?

Making predictions

- To find b, consider the closest point $\boldsymbol{x}_n$ to a new test sample $\boldsymbol{x}_{new}$, for which we already know:

$$\boldsymbol{w}^T\boldsymbol{x}_n + b = \pm 1 = t_n \qquad \text{or} \qquad t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) = 1$$

- Let's recall $\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n t_n \boldsymbol{x}_n$, so we get:

$$t_n\left(\sum_{m=1}^{N} \alpha_m t_m \boldsymbol{x}_m^T \boldsymbol{x}_n + b\right) = 1$$

$$\sum_{m=1}^{N} \alpha_m t_m \boldsymbol{x}_m^T \boldsymbol{x}_n + b = \frac{1}{t_n}$$

$$b = t_n - \sum_{m=1}^{N} \alpha_m t_m \boldsymbol{x}_m^T \boldsymbol{x}_n$$

$$\boxed{\frac{1}{t_n} = t_n}$$

- Thus, we can predict t_{new}:

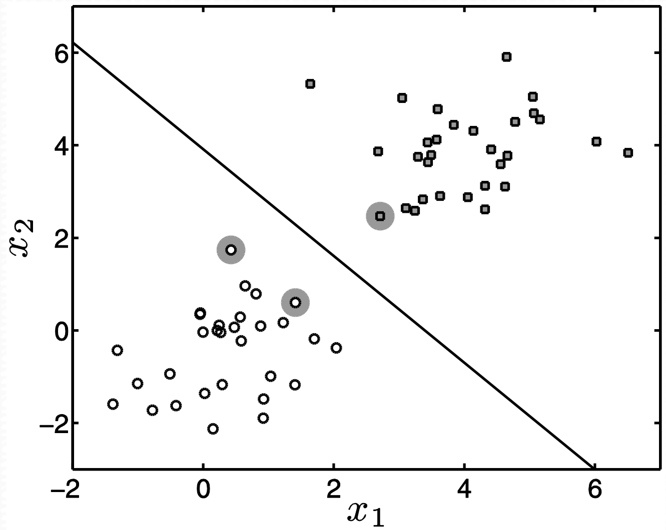
$$t_{new} = sign\left(\sum_{n=1}^{N} \alpha_n t_n \boldsymbol{x}_n^T \boldsymbol{x}_{new} + b\right)$$

Support vectors

- Support vector?

- Support vectors are the training samples closest to the maximum margin decision boundary
  - These vectors “support” the decision boundary

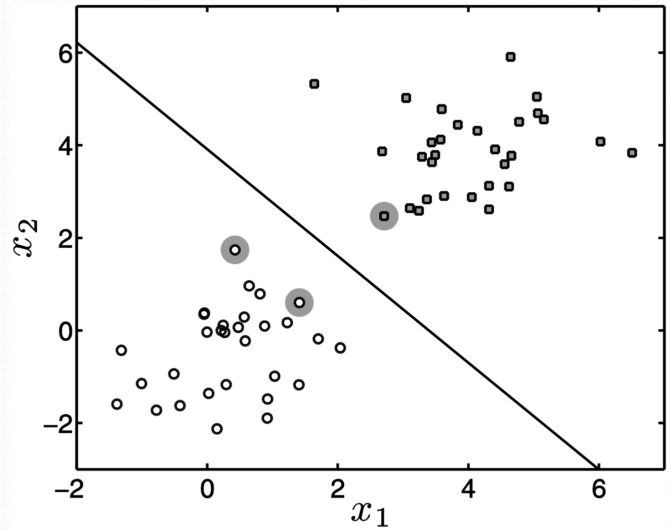
- Maximizing the margin determines the boundary
  - Margin is defined by support vectors
  - Can we discard non-support vectors?



Support vectors

- Can we discard non-support vectors?
- At the optimum, non-support vectors will have zero α_n

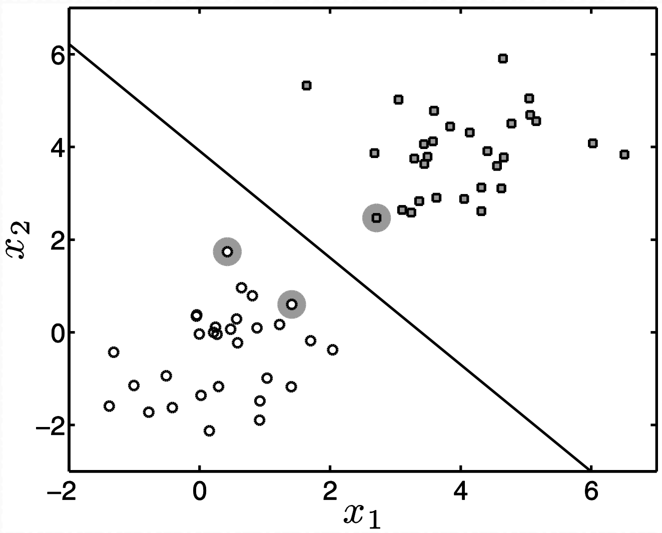
  $t_{new} = sign(\sum_{n=1}^{N} \alpha_n t_n \boldsymbol{x}_n^T \boldsymbol{x}_{new} + b)$

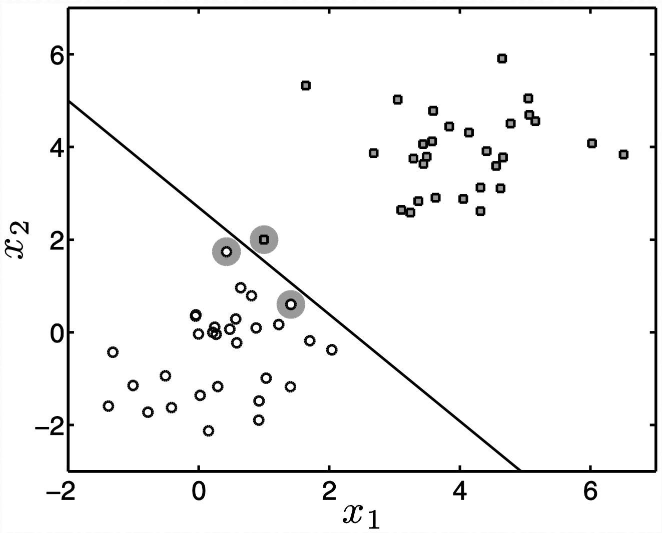


- We get a sparse solution
  - Decision boundary is a function of a small subset (i.e. support vectors)
- How does this compare to kNN classification?
  - kNN finds distance to all objects and finds k closest ones

Support vectors

- At times, a sparse solution can result in problems
- Why does this happen?
  - Hard margin: decision boundary is completely determined by training samples: $t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1$
  - All training samples need to reside on correct side of decision boundary
- Soft margin: permit some points to lie within margin band or even on the wrong side of boundary





Soft margin

- Permit some training points to lie within margin band or even on the wrong side of boundary
  - Relax the constraint $t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1$ to

$$t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1 - \xi_n, \text{ subject to } \xi_n \geq 0$$

- The optimization problem becomes:

$$\underset{\boldsymbol{w}}{argmin} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{n=1}^{N}\xi_n$$

subject to $t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1 - \xi_n$ and $\xi_n \geq 0$

- Parameter C controls to what extent the algorithm will permit the training samples to reside within margin band or on the wrong side of the boundary

Soft margin

- With use of Lagrange multipliers and similar math work as before, the optimization problem becomes:

$$\underset{\alpha}{argmax} \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_m \alpha_n \, t_m t_n \boldsymbol{x}_m^T \boldsymbol{x}_n$$

subject to

$$0 \le \alpha_n \le C$$

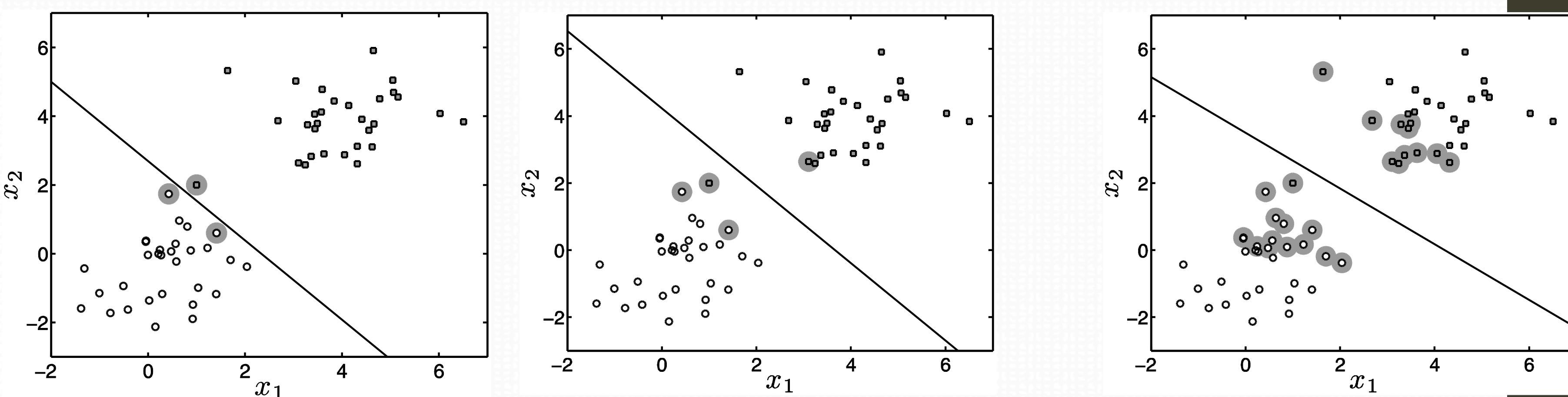
and

$$\sum_{n=1}^{N} \alpha_n t_n = 0$$

- Note that the only difference, compared to hard margin classifier, is an upper bound C on α_n

Soft margin

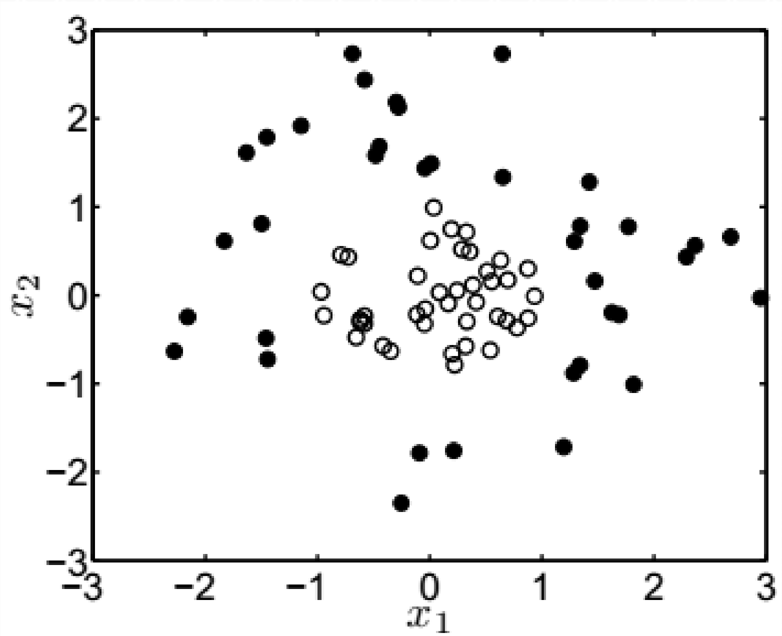
- For the stray support vector: $\alpha_n = 5.45$
- Setting C can bring a change in decision boundary
  - Some other α_n will have to become non-zero to bring change in the decision boundary
- With decreasing C, maximum potential influence of each training point is reduced
  - More training points become involved in the decision function
- How to choose C?
  - Cross-validation



(a) $C = 1$ (b) $C = 0.01$

Non-linear decision boundary

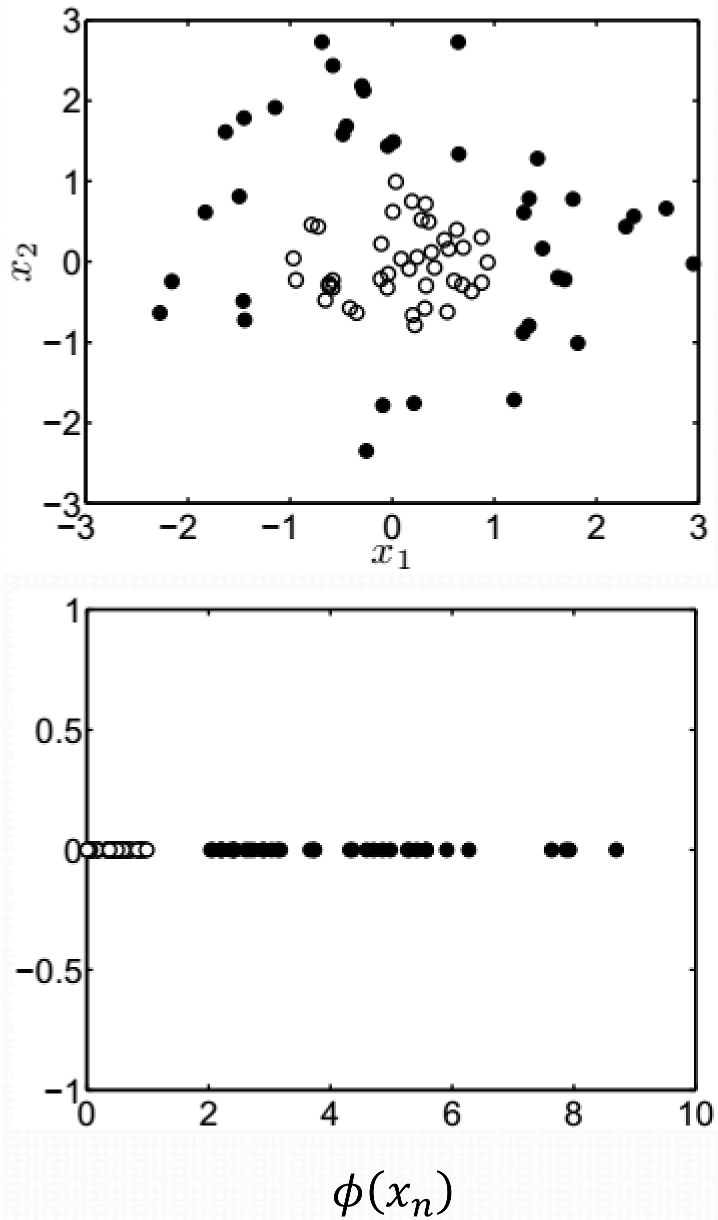
- SVM can determine linear decision boundary
  - What if data is not linearly separable?
  - Can soft margin help?



Non-linear decision boundary

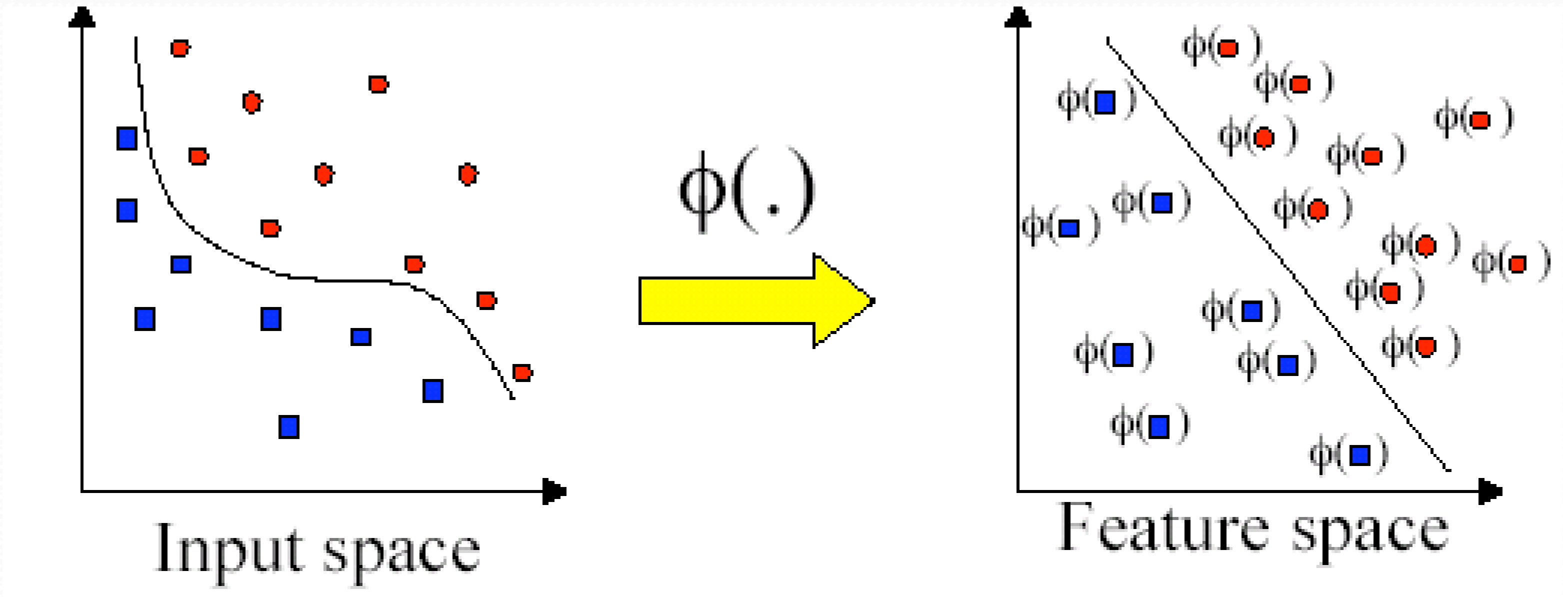
- How about transforming data in to a new space where it can be separable?
  - $\boldsymbol{x} \rightarrow \phi(\boldsymbol{x})$

- For this example, consider:
  $\phi(\boldsymbol{x}_n) = x_{n1}^2 + x_{n2}^2$



Non-linear decision boundary

- Transform (or project) the data in to a space where it's linearly separable



Non-linear decision boundary

- SVM optimization function:

$$\underset{\alpha}{argmax} \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_m \alpha_n t_m t_n \boldsymbol{x}_m^T \boldsymbol{x}_n$$

- Use $\phi(\boldsymbol{x}_n)$ instead of $\boldsymbol{x}_n$ in optimization:

$$\underset{\alpha}{argmax} \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_m \alpha_n t_m t_n \phi(\boldsymbol{x}_m)^T \phi(\boldsymbol{x}_n)$$

and in prediction:

$$t_{new} = sign\left(\sum_{n=1}^{N} \alpha_n t_n \phi(\boldsymbol{x}_n)^T \phi(\boldsymbol{x}_{new}) + b \right)$$

- Note that the data $\boldsymbol{x}_m$, $\boldsymbol{x}_n$, and $\boldsymbol{x}_{new}$ always appear within dot product
- After the transformation, the dot product is calculated in the new space

Kernel trick

- Let's consider:
  $\phi(\boldsymbol{x}_m)^T\phi(\boldsymbol{x}_n) = (x_{m1}^2 + x_{m2}^2)(x_{n1}^2 + x_{n2}^2) = k(\boldsymbol{x}_m, \boldsymbol{x}_n)$
- Dot product in the transformed space can be considered as a function of the original space
- *Kernel function*: a function that is equivalent to the dot product of vectors in the transformed $\phi(\ldots)$ space
  - $\phi(\boldsymbol{x}_m)^T\phi(\boldsymbol{x}_n) = k(\boldsymbol{x}_m, \boldsymbol{x}_n)$
- *Kernel trick*: very neat trick which doesn't even require the explicit data transformation

Kernel function

- There are a number of off-the-shelf kernels that have been shown to work well
  - Think of kernel as a similarity metric
- Linear kernel
  - $k(\boldsymbol{x}_m, \boldsymbol{x}_n) = \boldsymbol{x}_m{}^T \boldsymbol{x}_n$
- Gaussian kernel
  - $k(\boldsymbol{x}_m, \boldsymbol{x}_n) = exp\{-\beta(\boldsymbol{x}_m - \boldsymbol{x}_n)^T(\boldsymbol{x}_m - \boldsymbol{x}_n)\} = exp\{-\beta\|\boldsymbol{x}_m - \boldsymbol{x}_n\|^2\}$
- Polynomial kernel
  - $k(\boldsymbol{x}_m, \boldsymbol{x}_n) = (\boldsymbol{x}_m{}^T \boldsymbol{x}_n + c)^\beta$
- $k(\boldsymbol{x}_m, \boldsymbol{x}_n)$ corresponds to $\phi(\boldsymbol{x}_m)^T \phi(\boldsymbol{x}_n)$ for some transformation $\phi(\boldsymbol{x}_n)$
  - Don't even need to know what is $\phi(\boldsymbol{x}_n)$

Non-linear decision boundary

- Use $k(\boldsymbol{x}_m, \boldsymbol{x}_n)$ instead of $\phi(\boldsymbol{x}_m)^T \phi(\boldsymbol{x}_n)$ in optimization function:

$$\underset{\alpha}{argmax} \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_m \alpha_n \, t_m t_n k(\boldsymbol{x}_m, \boldsymbol{x}_n)$$

and in prediction function:

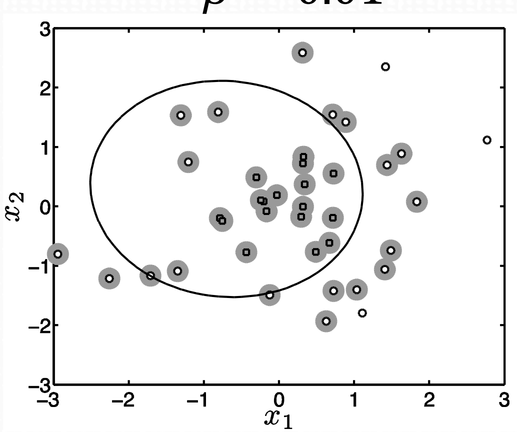
$$t_{new} = sign\left(\sum_{n=1}^{N} \alpha_n t_n \, k(\boldsymbol{x}_n, \boldsymbol{x}_{new}) + b \right)$$

- SVM is still finding linear boundaries…
  - …but in some other space

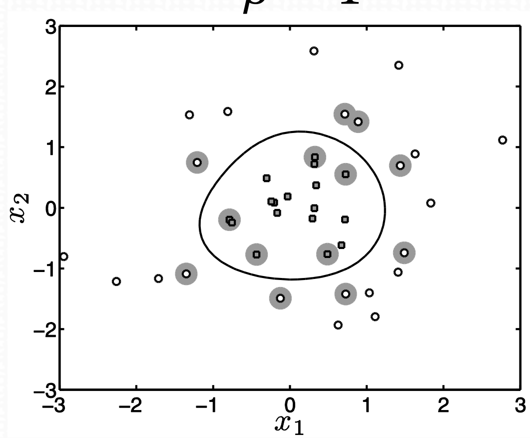
Non-linear decision boundary

- Non-linear data classification with SVM using Gaussian kernel
  - $C = 10$
- β controls the *model complexity*
  - Very small $\beta \Rightarrow$ too simple (under-fitting)
  - Very large $\beta \Rightarrow$ too complex (over-fitting)
- Non-sparse model for too small or too large β

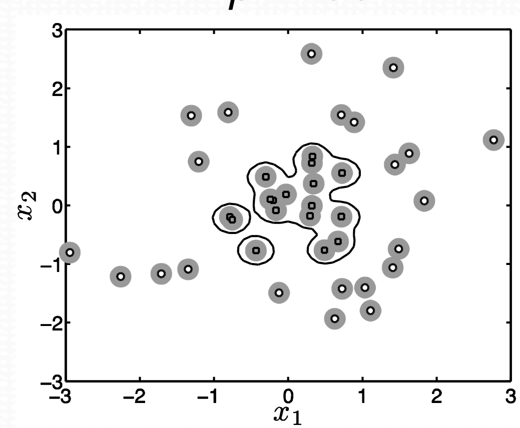
$\beta = 0.01$



$\beta = 1$

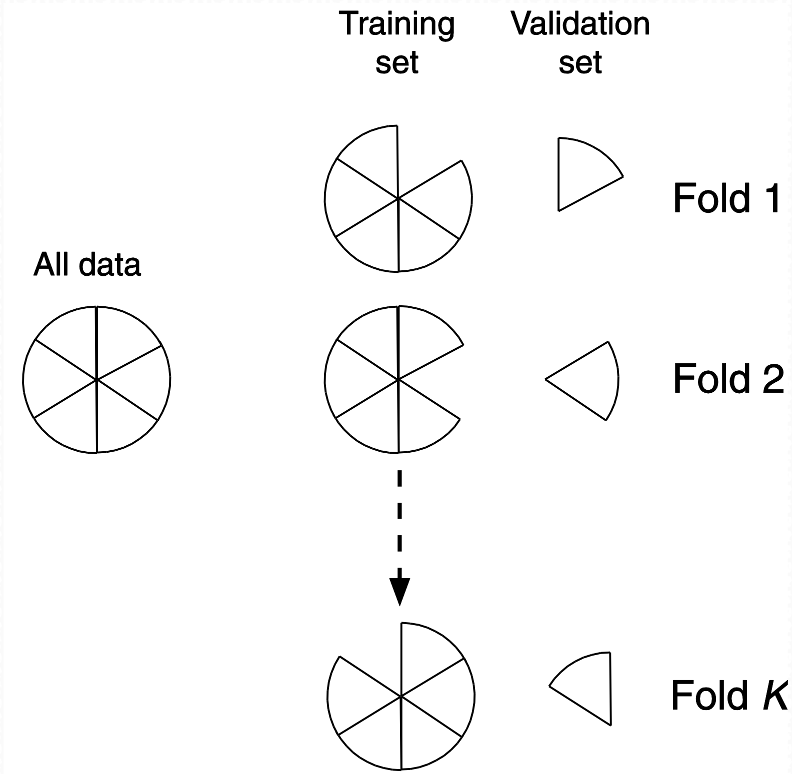


$\beta = 50$



Parameter selection

- How to choose C and β?
  - Parameter choice is data dependent
- Cross-validation
  - Search over C and β
- Extra computational burden



Kernelizing other algorithms

- Other algorithms can be kernelized
  - As long as they have data appearing only in inner products in model learning and prediction
- Simple algorithms can learn complex decision boundaries
  - We have seen its usefulness in k-means clustering
- kNN requires distance between each training sample $\boldsymbol{x}_n$ and test sample $\boldsymbol{x}_{new}$
  - The distance can be written as: $(\boldsymbol{x}_n - \boldsymbol{x}_{new})^T(\boldsymbol{x}_n - \boldsymbol{x}_{new})$
  - Can we kernelize kNN classifier?

SVM multi-class classification

- How to use a binary classifier (e.g. SVM) for multi-class classification?
- Consider that we have C number of classes
  - C not to be confused with C parameter for soft margin (they're different!!)
- There are two common strategies
  - One-vs-all
  - One-vs-one

SVM multi-class classification

- One-vs-all
  - This is the most frequently used option
  - Train C distinct binary classifiers, each classifier learning one-vs-rest (one: $+1$, rest: -1)
  - For classification:

$$t_{new} = arg \max_{c \in 1 \ldots C} f_c(x_{new})$$

  where f_c is the classifier function that predicts confidence score for label c

  - This approach creates class imbalance problem
    - i.e. one class has many more samples than the other one

SVM multi-class classification

- One-vs-one
  - Train $C(C-1)/2$ distinct binary classifiers, each classifier learning one-vs-one ($+1$ vs -1)
  - For classification, all classifiers are used and t_{new} is assigned according to maximum voting

  - Addresses class imbalance problem in data
  - Some test samples may receive same number of votes for multiple classes

Summary

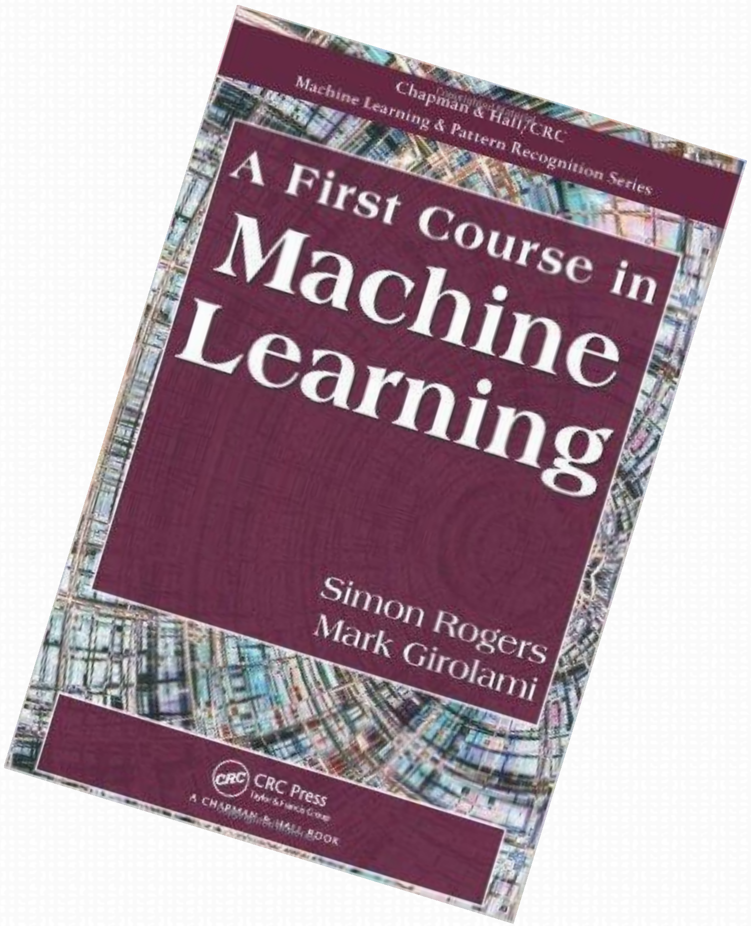
- Discriminative classification
- SVM: non-probabilistic linear binary classifier
  - Margin maximization
- Support vectors
- Hard margin vs soft margin
- Non-linear boundary learning with kernel trick
- Multi-class classification with a binary classifier

Exercise (ungraded)

- Try MATLAB code - svmhard.m (from FCML book website)
  - Requires quadprog from Optimization Toolbox

- Try MATLAB code - svmsoft.m (from FCML book website)
  - Requires quadprog from Optimization Toolbox

- Try MATLAB code - svmgauss.m (from FCML book website)
  - Requires quadprog from Optimization Toolbox



Author’s material (Simon Rogers)

Ata Kaban

That's All

Thank You