Maximum Margin Classifier / Kernel Machine / Support Vector Machine

Tae-Kyun Kim
Senior Lecturer
https://labicvl.github.io/

Some references on *Kernel*

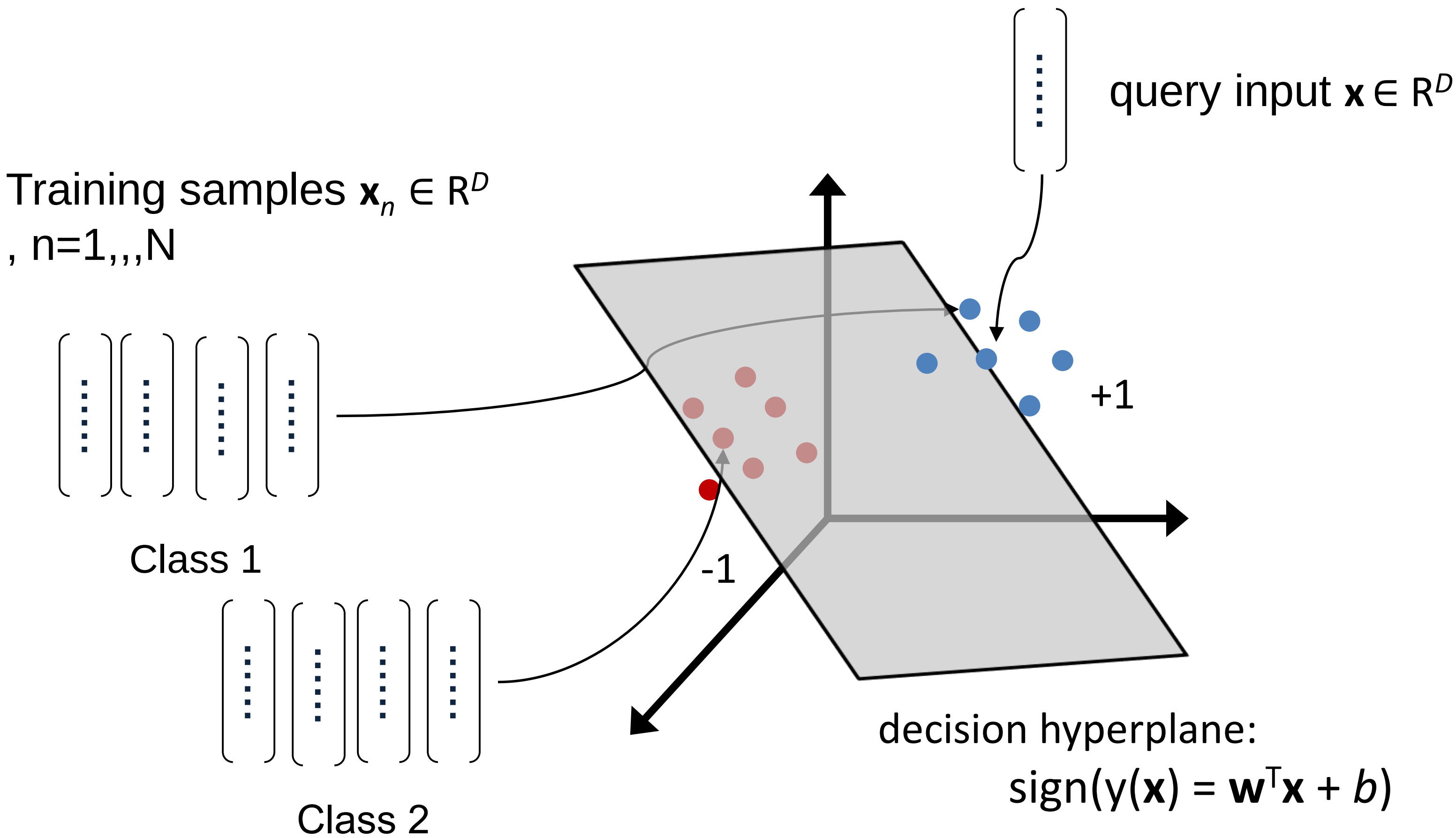
ICML 2018

Optimal Tuning for Divide-and-conquer Kernel Ridge Regression with Massive Data, Ganggang Xu · Zuofeng Shang · Guang Cheng
BOCK : Bayesian Optimization with Cylindrical Kernels, ChangYong Oh · Efstratios Gavves · Max Welling
Differentiable Compositional Kernel Learning for Gaussian Processes, Shengyang Sun · Guodong Zhang · Chaoqi Wang · Wenyuan Zeng · Jiaman Li · Roger Grosse
The Weighted Kendall and High-order Kernels for Permutations, Yunlong Jiao · Jean-Philippe Vert
Kernelized Synaptic Weight Matrices, Lorenz Müller · Julien Martel · Giacomo Indiveri
AutoPrognosis: Automated Clinical Prognostic Modeling via Bayesian Optimization with Structured Kernel Learning, Ahmed M. Alaa Ibrahim · M van der Schaar
To Understand Deep Learning We Need to Understand Kernel Learning, Mikhail Belkin · Siyuan Ma · Soumik Mandal
Kernel Recursive ABC: Point Estimation with Intractable Likelihood, Takafumi Kajihara · Motonobu Kanagawa · Keisuke Yamazaki · Kenji Fukumizu
Differentially Private Database Release via Kernel Mean Embeddings, Matej Balog · Ilya Tolstikhin · Bernhard Schölkopf
Learning in Reproducing Kernel Kreĭn Spaces, Dino Oglic · Thomas Gaertner
Trainable Calibration Measures for Neural Networks from Kernel Mean Embeddings, Aviral Kumar · Sunita Sarawagi · Ujjwal Jain
Adaptive Sampled Softmax with Kernel Based Sampling, Guy Blanc · Steffen Rendle

NIPS 2018

Persistence Fisher Kernel: A Riemannian Manifold Kernel for Persistence Diagrams, Tam Le · Makoto Yamada
Neural Tangent Kernel: Convergence and Generalization in Neural Networks, Arthur Jacot-Guillarmod · Clement Hongler · Franck Gabriel
Statistical and Computational Trade-Offs in Kernel K-Means, Daniele Calandriello · Lorenzo Rosasco
RetGK: Graph Kernels based on Return Probabilities of Random Walks, Zhen Zhang · Mianzhi Wang · Yijian Xiang · Yan Huang · Arye Nehorai
Learning Bounds for Greedy Approximation with Explicit Feature Maps from Multiple Kernels, Shahin Shahrampour · Vahid Tarokh
Quadrature-based features for kernel approximation, Marina Munkhoeva · Yermek Kapushev · Evgeny Burnaev · Ivan Oseledets
Causal Inference via Kernel Deviance Measures, Jovana Mitrovic · Dino Sejdinovic · Yee Whye Teh
KONG: Kernels for ordered-neighborhood graphs, Moez Draief · Konstantin Kutzkov · Kevin Scaman · Milan Vojnovic
Continuous-time Value Function Approximation in Reproducing Kernel Hilbert Spaces, Motoya Ohnishi · Masahiro Yukawa · Mikael Johansson · Masashi Sugiyama
Semi-supervised Deep Kernel Learning: Regression with Unlabeled Data by Minimizing Predictive Variance, Neal Jean · Sang Michael Xie · Stefano Ermon
Streaming Kernel PCA with $\tilde{O}(\sqrt{n})$ ***Random Features,*** Md Enayat Ullah · Poorya Mianjy · Teodor Vanislavov Marinov · Raman Arora

Discriminative linear classifier



Notations

x : vector for classification

D : dimension of input vector **x**

N : number of training data vectors **x**

M : number of classes

$\mathcal{C}_m$: m-th class

y(**x**) : SVM output (before discretization)

ϕ(**x**) : nonlinear (kernel) mapping

D': dimension of feature space after kernel mapping

$k(\mathbf{x}_1, \mathbf{x}_2)$: kernel function

t_n : binary target variable of $\boldsymbol{x}_n$

N_{sv} : number of support vectors

ξ_n *: slack variables*

C : trade-off constant for misclassified samples

Linear discriminant function

– A discriminant function maps an input vector **x** into one of *M* classes, denoted $\mathcal{C}_m$.

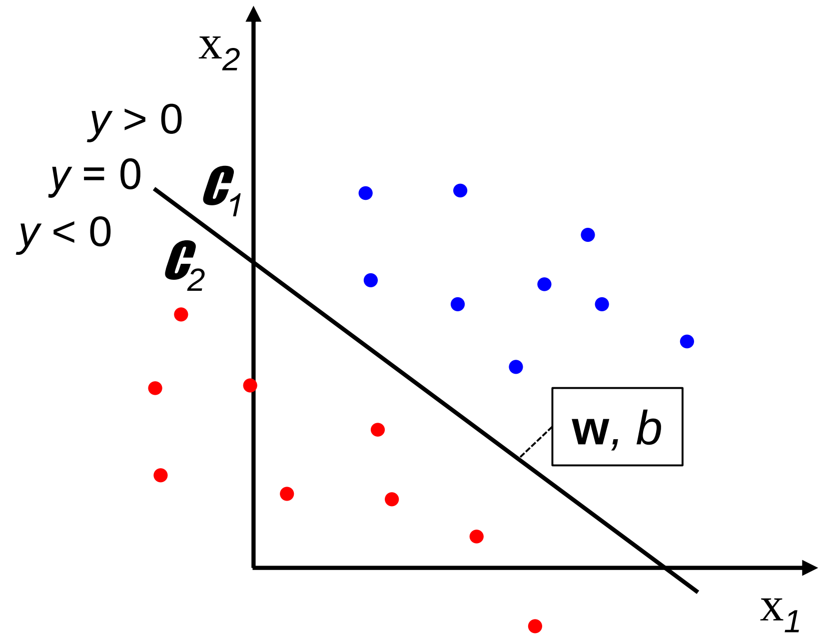
  For simplicity, consider two classes.

– *Linear* discriminant function takes the form of

  $y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$

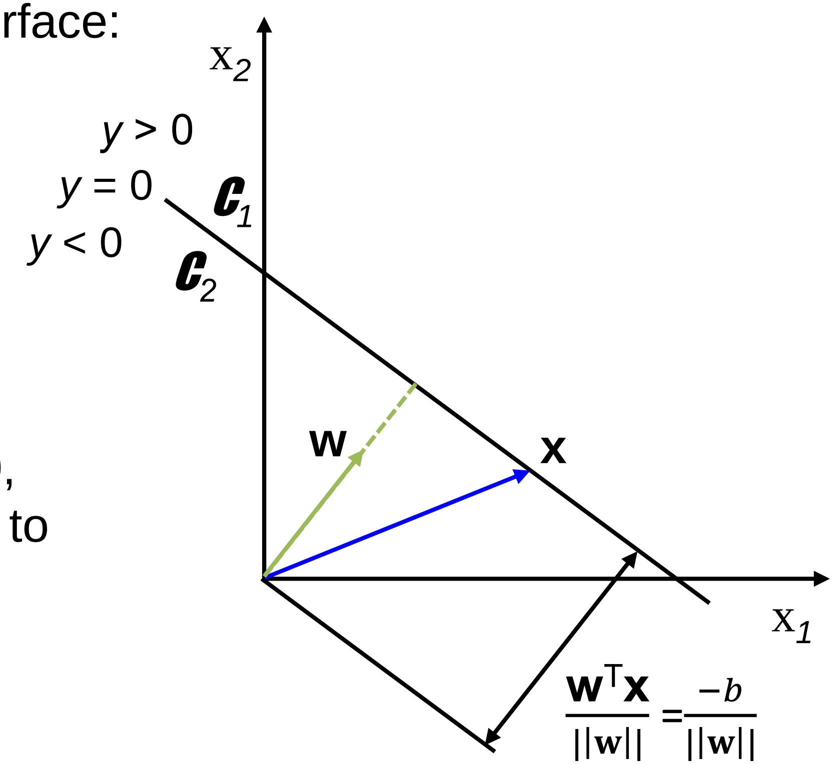
  where **w** is a *weight vector* and *b* a *bias*.

– A vector **x** is assigned to $\mathcal{C}_1$ if $y(\mathbf{x}) \geq 0$, and $\mathcal{C}_2$ otherwise.



Linear discriminant function

- The decision boundary is defined by $y(\mathbf{x}) = 0$, which is a $(K\text{-}1)$-dimensional hyperplane in the K-dimensional input space.
- Consider $\mathbf{x}_A$ and $\mathbf{x}_B$ on the decision surface:
  - → $y(\mathbf{x}_A) = y(\mathbf{x}_B) = 0$
  - → $\mathbf{w}^T(\mathbf{x}_A - \mathbf{x}_B) = 0$



- The vector $\mathbf{w}$ is orthogonal to the decision surface. $\mathbf{w}$ determines the direction of the decision surface.
- For $\mathbf{x}$ on the decision surface, $y(\mathbf{x}) = 0$, so the normal distance from the origin to the decision surface is

$$\frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||} = \frac{-b}{||\mathbf{w}||}$$

- Thus, b determines the location of the decision surface.

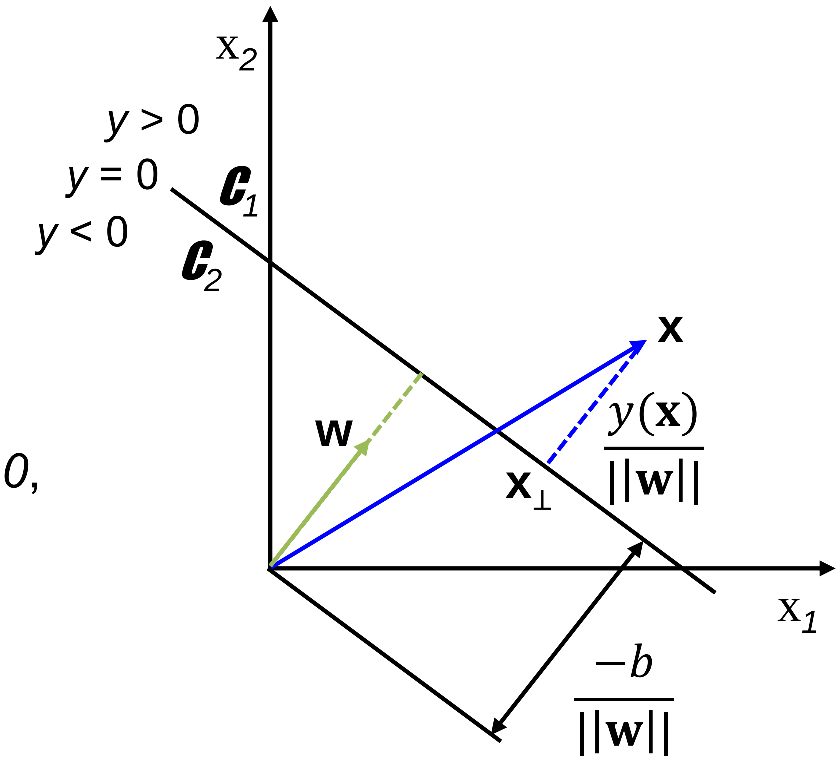
Linear discriminant function

– For an arbitrary point **x**, its orthogonal projection onto the decision surface $\mathbf{x}_\perp$ is such that

$$\mathbf{x} = \mathbf{x}_\perp + r\frac{\mathbf{w}}{||\mathbf{w}||}.$$

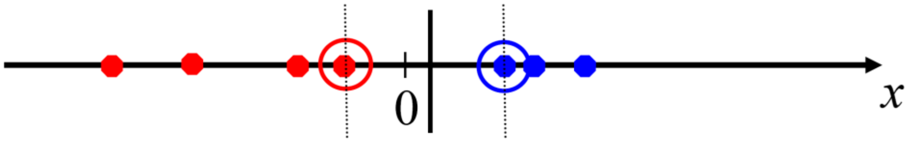
– Multiply both sides by $\mathbf{w}^T$ and add b, and use $y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$ and $y(\mathbf{x}_\perp) = \mathbf{w}^T\mathbf{x}_\perp + b = 0$, we have

$$r = \frac{y(\mathbf{x})}{||\mathbf{w}||}.$$

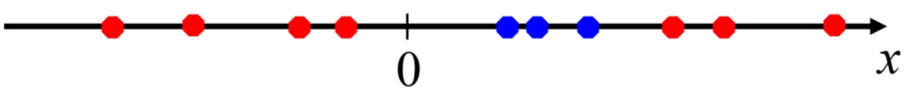


Kernel trick

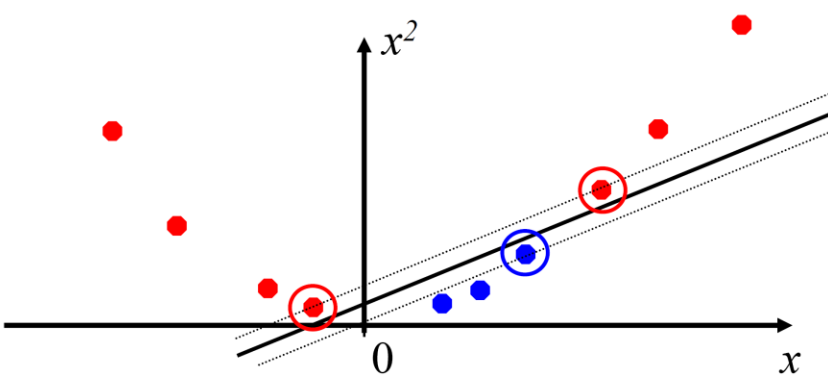
– Linear discriminant functions separate data linearly:



– Dataset is often not linearly separable:

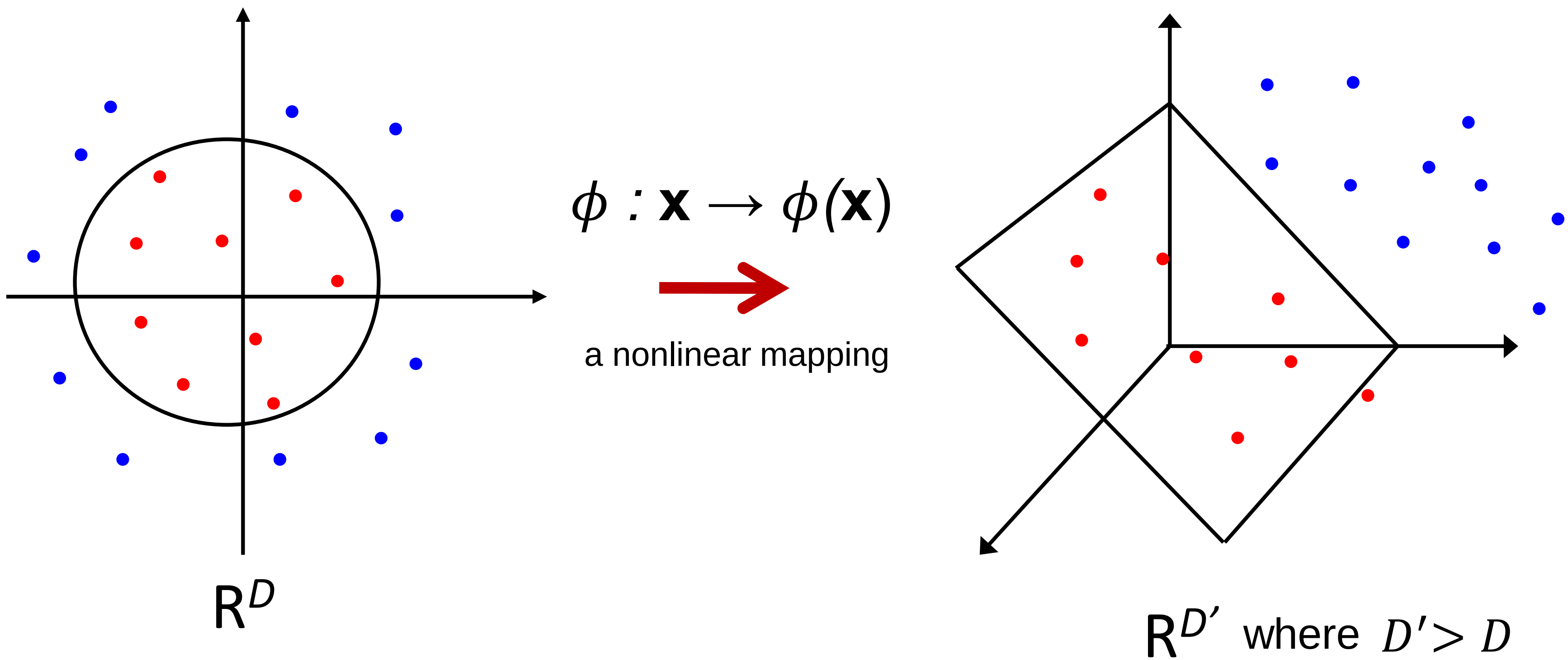


– We can map it to a higher-dimensional space, then separate it linearly:



Kernel trick

- The input space is transformed into a higher-dimensional feature space by a nonlinear mapping $\phi : \mathbf{x} \rightarrow \phi(\mathbf{x})$, where a linear decision boundary can separate all data points.
- This second feature space may have a high or even infinite dimension.



Kernel trick

– One highlight of Kernel Trick is that machine learning problems can be formulated entirely in terms of scalar products in the second feature space, by introducing the kernel function

$$k(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1)^{\mathsf{T}}\phi(\mathbf{x}_2)$$

– The kernel is a symmetric function s.t. $k(\mathbf{x}_1, \mathbf{x}_2) = k(\mathbf{x}_2, \mathbf{x}_1)$

– NOTE:

- It is not required to compute $\phi(\mathbf{x})$ explicitly.

– Examples of the kernel functions are:

- Linear kernel: $k(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1^T\mathbf{x}_2$
- Polynomial kernel: $k(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1^T\mathbf{x}_2 + c)^M$, where $c > 0$
- Gaussian (or RBF) kernel: $k(\mathbf{x}_1, \mathbf{x}_2) = \exp(-||\mathbf{x}_1 - \mathbf{x}_2||^2/2\sigma^2)$

Maximum margin classifier / Sparse kernel machine / Support vector machine

- The classifier finds a hyperplane which separates two-class data with maximal margin.
- The margin is defined as the distance of the closest training point to the separating or decision hyperplane.
- Thus it is also called maximum margin classifier.

- It can be shown that the support vectors are those feature vectors lying nearest to the decision hyperplane.

- It has sparse solutions i.e. the predictions for new inputs depend only on the kernel function evaluated at a subset of the training data points, which are called support vectors.
- Thus it is also called sparse kernel machine.

Maximum margin classifier

– We have N training data vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$ and their target values $t_1, \ldots, t_N$ where $t_n \in \{-1, 1\}$.

– For the two-class problem it takes the form of

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b$$

where $\phi(\mathbf{x})$ denotes the feature mapping, $\mathbf{w}$ is the weight vector and b the bias.

– Data points $\mathbf{x}$ are classified by the sign of $y(\mathbf{x})$.

– Assume that the training data is linearly separable in the feature space. Thus, optimal $\mathbf{w}$ and b satisfy

$$t_n y(\mathbf{x}_n) > 0$$

for all training data points.

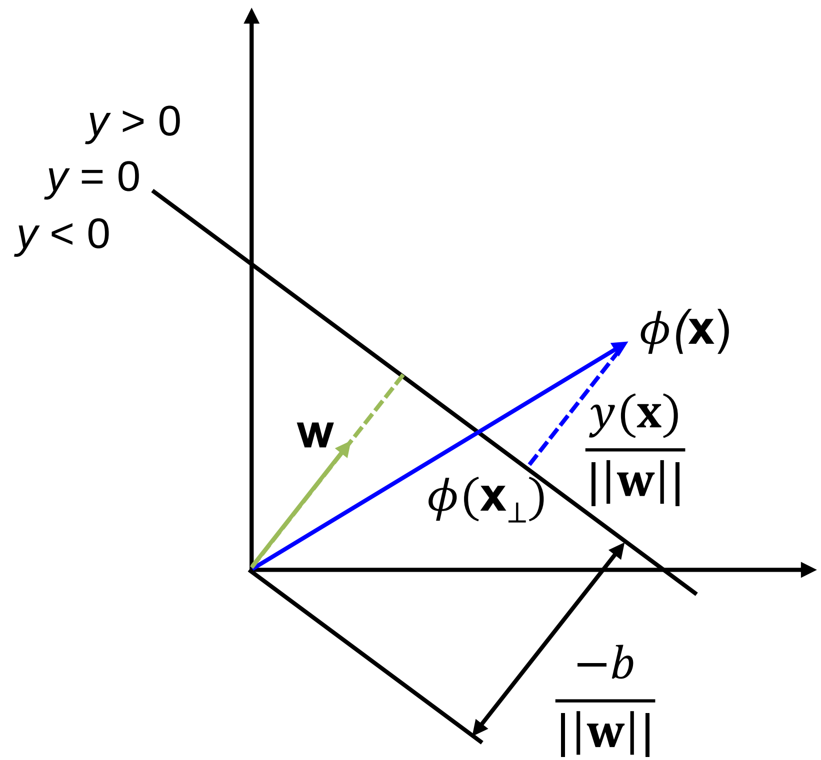
Maximum margin classifier

- The perpendicular distance of a point **x** from a hyperplane $y(\mathbf{x})$ is

$$|y(\mathbf{x})|/||\mathbf{w}||$$

- As we assume $t_n y(\mathbf{x}_n) > 0$ for all n, the distance of a point $\mathbf{x}_n$ to the decision surface is

$$\frac{t_n y(\mathbf{x}_n)}{||\mathbf{w}||} = \frac{t_n(\mathbf{w}^T\phi(\mathbf{x}_n) + b)}{||\mathbf{w}||}$$



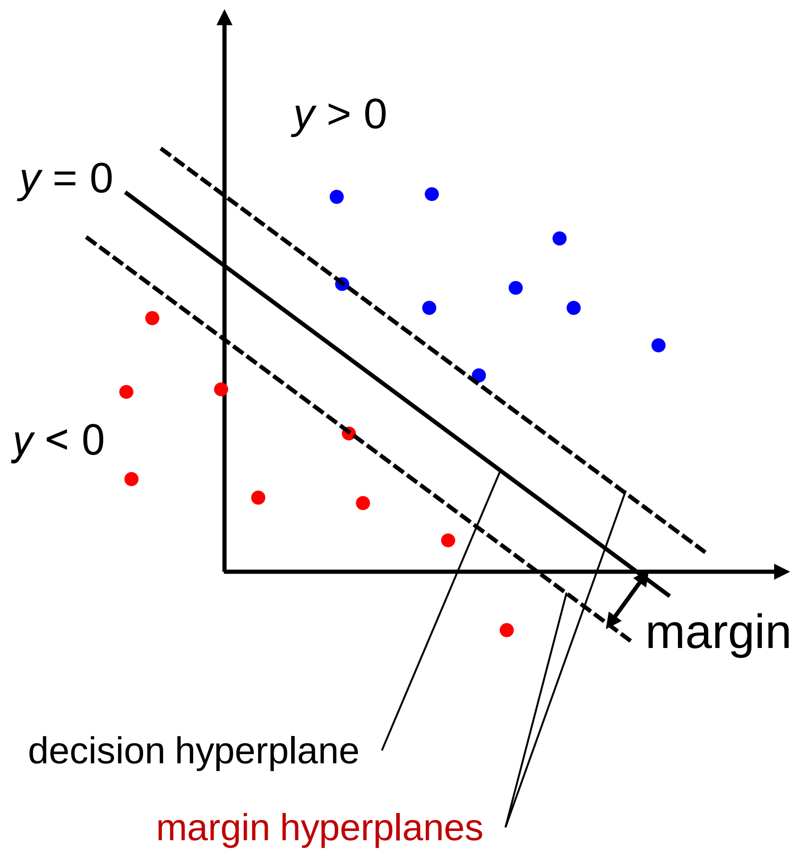
Maximum margin classifier

- The margin is the minimum perpendicular distance:

$$\frac{1}{||\mathbf{w}||}\min_{n}[t_n(\mathbf{w}^T\phi(\mathbf{x}_n)+b)]$$

- We find **w** and *b* that maximise the margin i.e.

$$\arg\max_{\mathbf{w},b}\left\{\frac{1}{||\mathbf{w}||}\min_{n}[t_n(\mathbf{w}^T\phi(\mathbf{x}_n)+b)]\right\}$$



https://goo.gl/m6Mnov for questions

Canonical representation of decision hyperplane

- Rescaling **w** → s**w** and b → sb does not change the distance from any point $\mathbf{x}_n$ to the decision hyperplane.

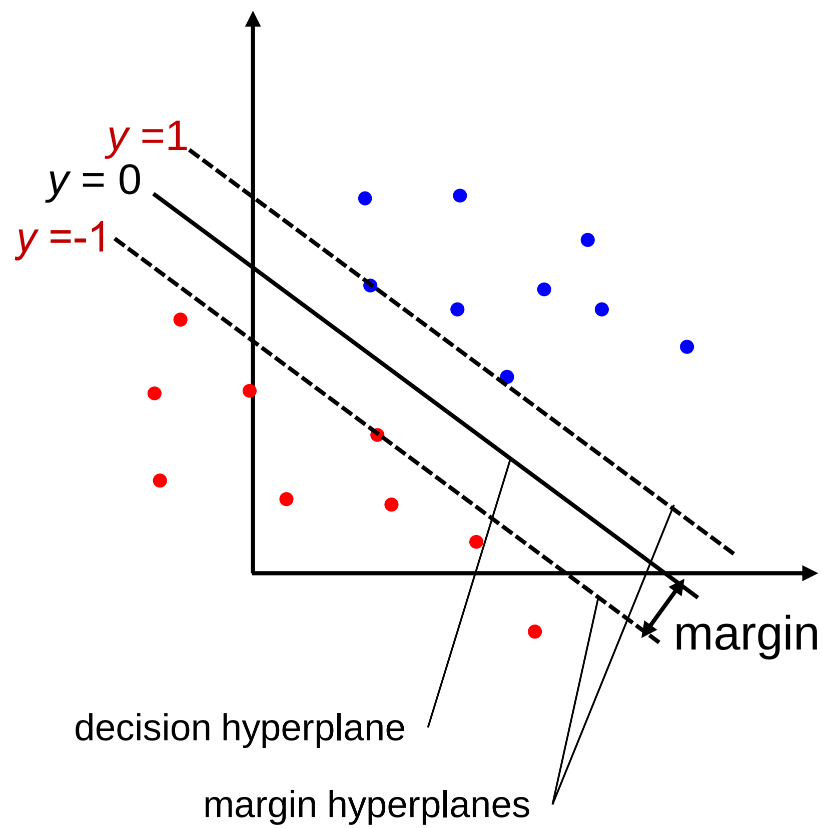
- We can therefore set

$$t_n(\mathbf{w}^T\phi(\mathbf{x}_n)+b)=1$$

for the point that is closest to the hyperplane.

- All data points satisfy

$$t_n(\mathbf{w}^T\phi(\mathbf{x}_n)+b)\geq 1, \quad n=1,...,N$$



Optimisation

– The original problem $$\arg\max_{\mathbf{w},b}\left\{\frac{1}{||\mathbf{w}||}\min_n[t_n(\mathbf{w}^T\phi(\mathbf{x}_n)+b)]\right\}$$

becomes to maximise $||w||^{-1}$.

– Equivalently, we have $$\arg\min_{\mathbf{w},b}\frac{1}{2}||\mathbf{w}||^2$$

subject to the constraints $$t_n(\mathbf{w}^T\phi(\mathbf{x}_n)+b)\geq 1,\quad n=1,...,N$$

– The problem is solved by Lagrange multipliers $a_n \geq 0$:

$$L(\mathbf{w},b,\mathbf{a})=\frac{1}{2}||\mathbf{w}||^2-\sum_{n=1}^{N}a_n\{t_n(\mathbf{w}^T\phi(\mathbf{x}_n)+b)-1\}$$

– Note the minus sign in front of the Lagrange multiplier term, as we are minimising the function.

Optimisation

– Karush-Kuhn-Tucker (KKT) conditions of the generic Lagrange function with inequality constraints are

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x}). \xrightarrow{\text{KKT conditions}} \begin{aligned} \lambda &\geq 0, \\ g(\mathbf{x}) &\geq 0, \\ \lambda g(\mathbf{x}) &= 0. \end{aligned}$$

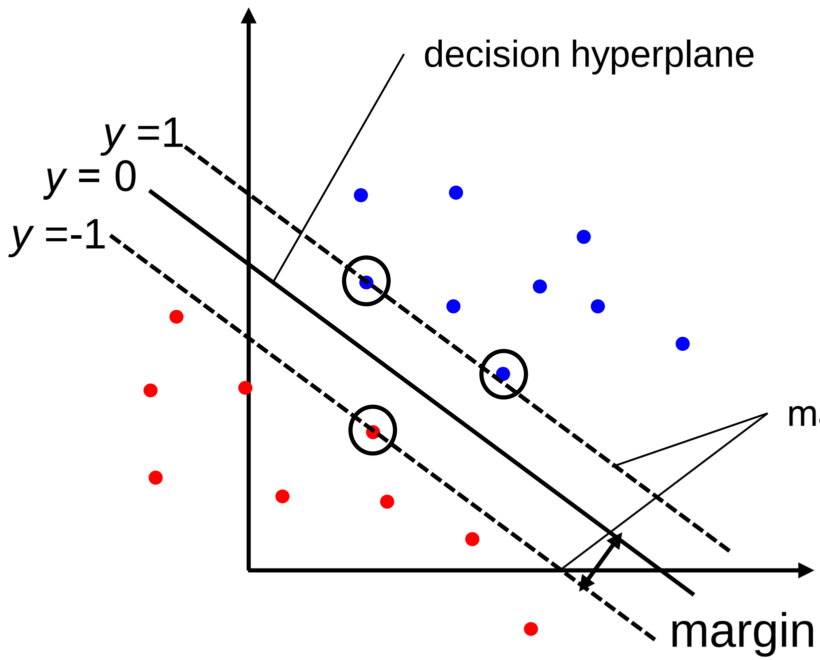
– The KKT conditions for SVM are

$$a_n \geq 0$$
$$t_n y(\mathbf{x}_n) - 1 \geq 0$$
$$a_n \{t_n y(\mathbf{x}_n) - 1\} = 0.$$

$\rightarrow$

- For every data point, a_n=0 or $t_n y(\mathbf{x}_n)$ =1.
- Any data point for which a_n=0 plays no role in making predictions.
- The remaining data points are called support vectors, and they satisfy $t_n y(\mathbf{x}_n)$ =1, i.e. lying on the margin hyperplanes.

Optimisation



Support vectors (in circle) lie on the margin hyperplanes.

Optimisation

– Setting the derivative of $L(\mathbf{w}, b, \mathrm{a})$ w.r.t. **w** to zero, we obtain

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n)$$

– The decision function $y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b$

takes the form of $$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n \boxed{k(\mathbf{x}, \mathbf{x}_n)} + b$$

where $a_n \geq 0$, $\mathbf{x}_n$ are the training data, t_n are the label, b is the bias, and

$$k(\mathbf{x}, \mathbf{x}_n) = \phi(\mathbf{x})^T \phi(\mathbf{x}_n)$$

Further reading:
Christopher J.C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition (http://research.microsoft.com/pubs/67119/svmtutorial.pdf)

Optimisation

– In the above,
  - a_n are typically zero for most *n*.
  - Equivalently, the sum can be taken only over a selected few of $\mathbf{x}_n$.
  - These vectors are known as support vectors.
  - It was shown that the support vectors are those vectors lying on the margin hyperplanes.

– Computational complexity for classifying a new data point **x** (linear vs nonlinear SVM)
  - Whereas the nonlinear SVM requires O(DxN$_{sv}$), the linear SVM takes O(D) where D is the data vector dimension and N$_{sv}$ is the number of support vectors.

Nonlinear: $$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b$$ where $k(\mathbf{x}_1, \mathbf{x}_2) = \exp(-||\mathbf{x}_1 - \mathbf{x}_2||^2 / 2\sigma^2)$

Linear: $$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n \mathbf{x}^T \mathbf{x}_n + b = \mathbf{x}^T \sum_{n=1}^{N} a_n t_n \mathbf{x}_n + b$$

pre-computed

https://goo.gl/m6Mnov for questions

Overlapping class distributions (soft margin)

– It penalizes samples by $\xi_n \geq 0$, n=1,...,N s.t.

$\xi_n = 0$ for data points that are on or inside the correct margin boundary

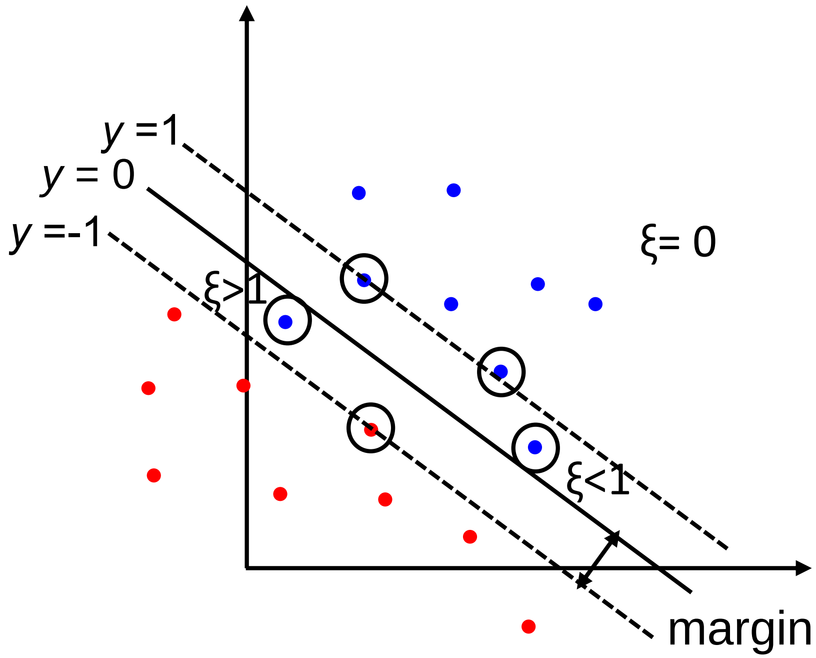
$\xi_n = |t_n - y(\mathbf{x}_n)|$ for other points.

– We therefore minimise

$$C \sum_{n=1}^{N} \xi_n + \frac{1}{2}||\mathbf{w}||^2$$

where $C > 0$ is a trade-off constant and $\xi_n \geq 0$, subject to

$$t_n y(\mathbf{x}_n) \geq 1 - \xi_n, \quad n = 1, ..., N.$$

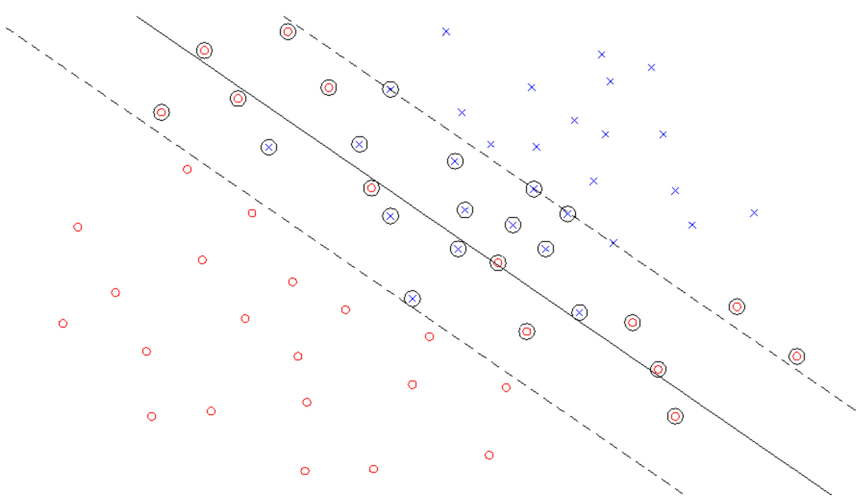


Support vectors are shown in circle

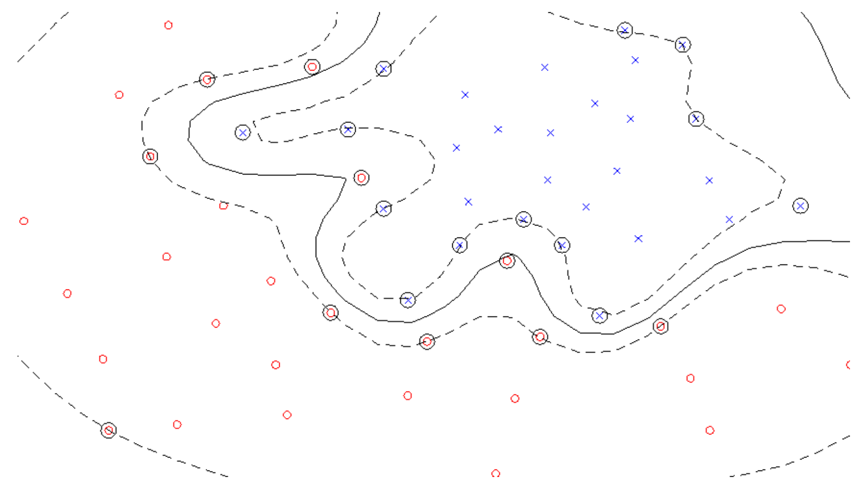
Overfitting

– The kernel *k* (type and parameter) and other hyper-parameters (C) are problem dependent and need to be determined by a user.
– Simple model has better generalization to unseen data, complex model can be overfitted to training data.
– However, simple model may exhibit a limited separation (underfitting).

Linear kernel (underfit in this example)

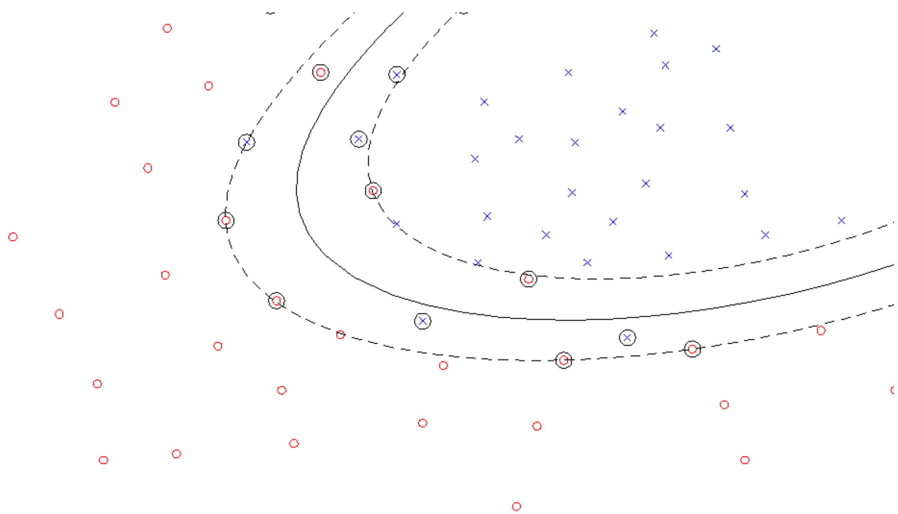


RBF kernel, σ=0.2 (overfit)

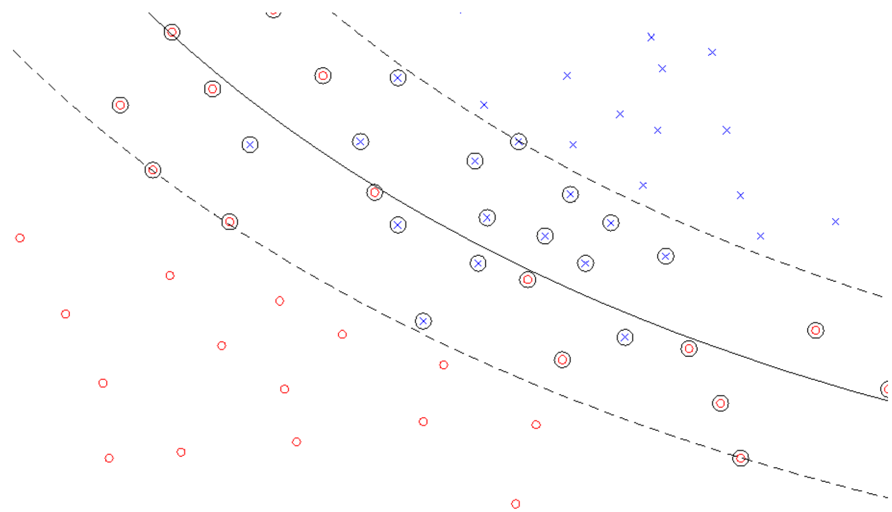


Overfitting

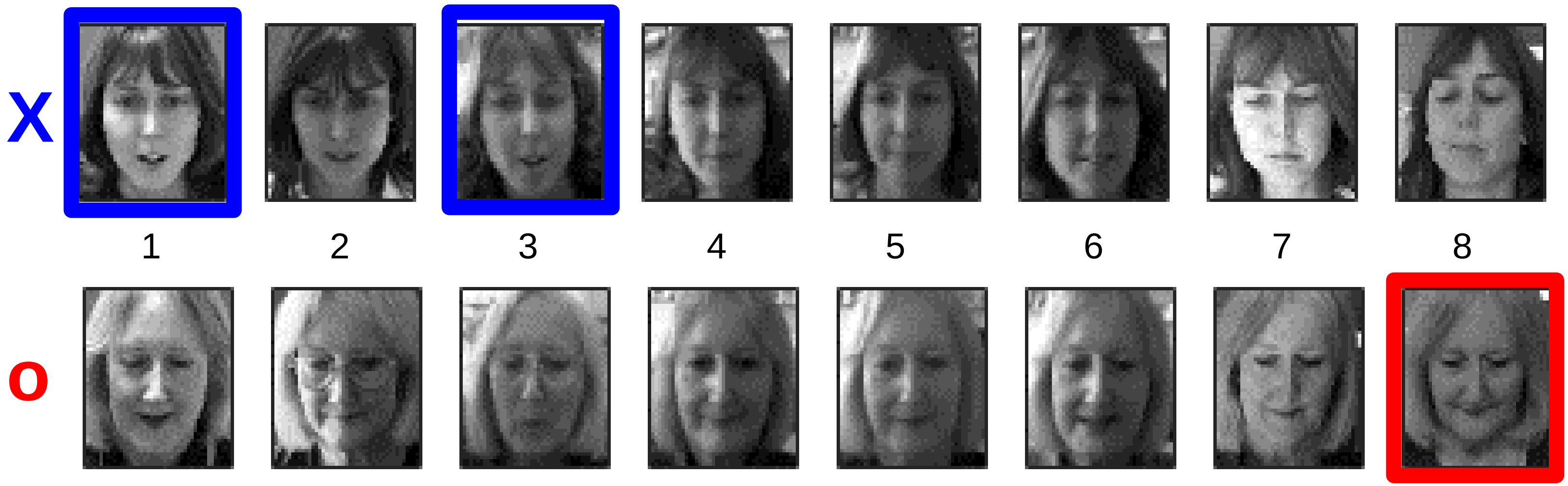
RBF kernel, σ =1.0



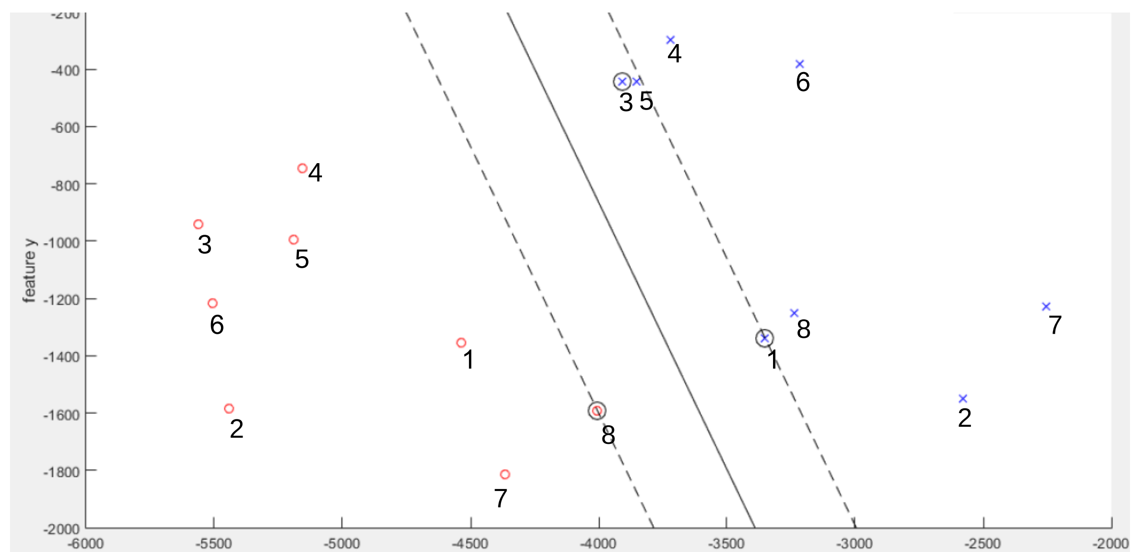
RBF kernel, σ =5.0
(underfit)



Application to face recognition



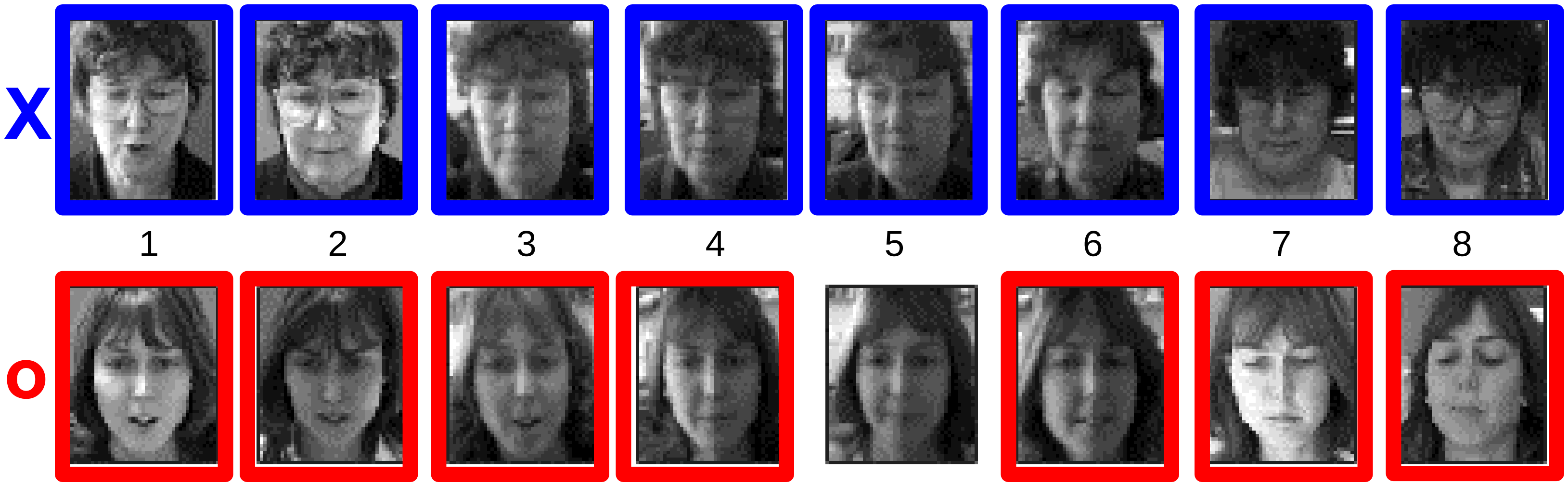
Kernel type: linear
Kernel argument: n/a
C: 100
Number of support vectors: 3
Margin: 348.1073
Training error: 0.00%



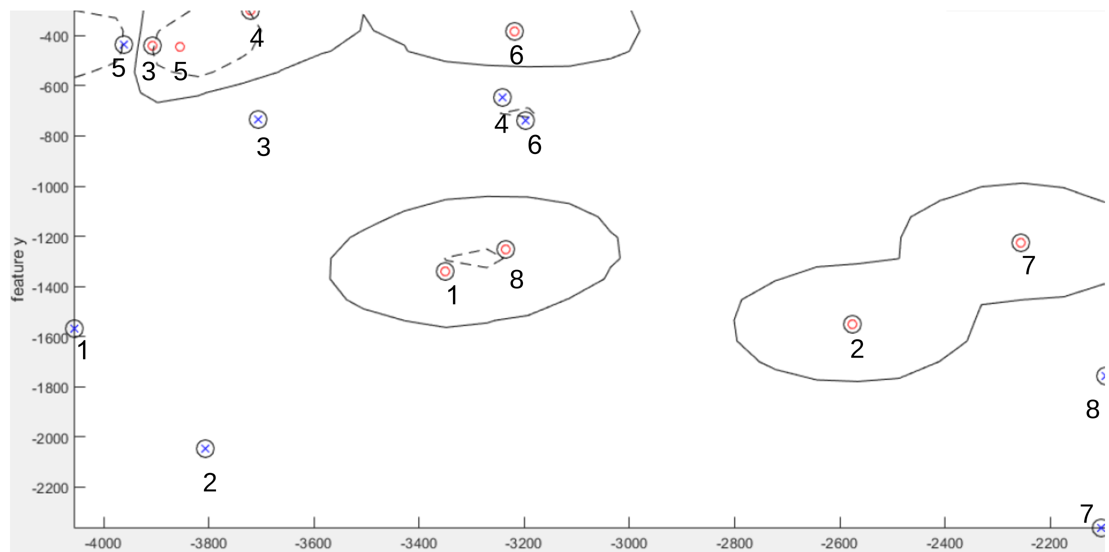
Face images and the decision boundary in 2-dimensional eigensubspace

https://goo.gl/m6Mnov for questions

Application to face recognition

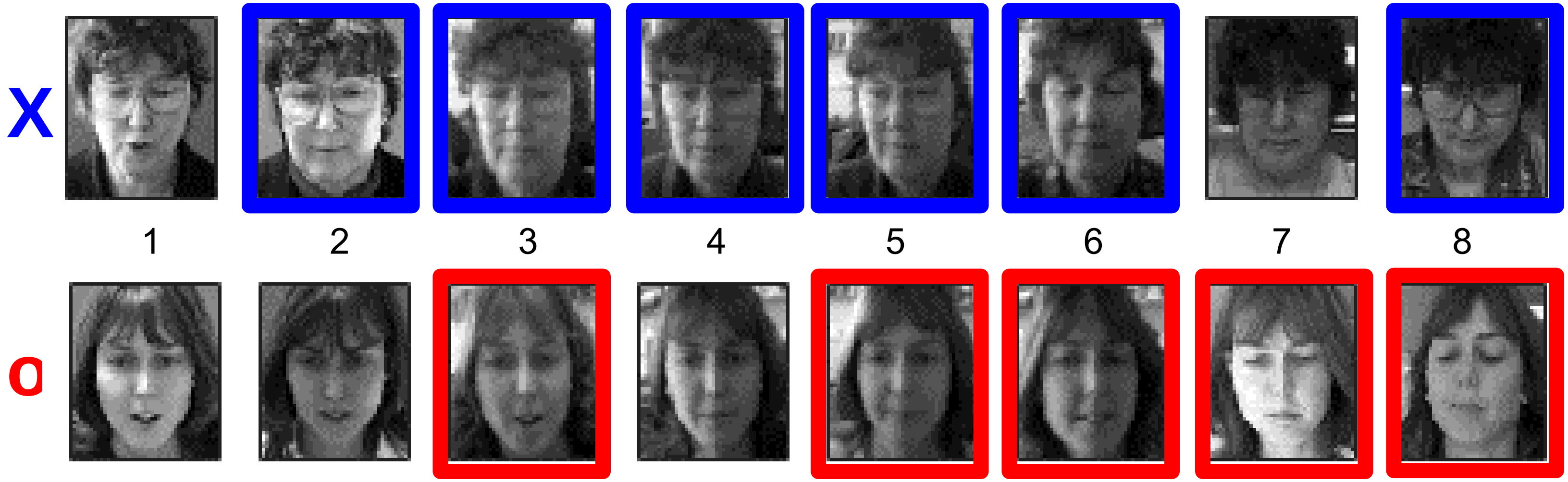


Kernel type: RBF
Kernel argument: 100
C: 100
Number of support vectors: 15
Margin: 0.1980
Training error: 0.00%

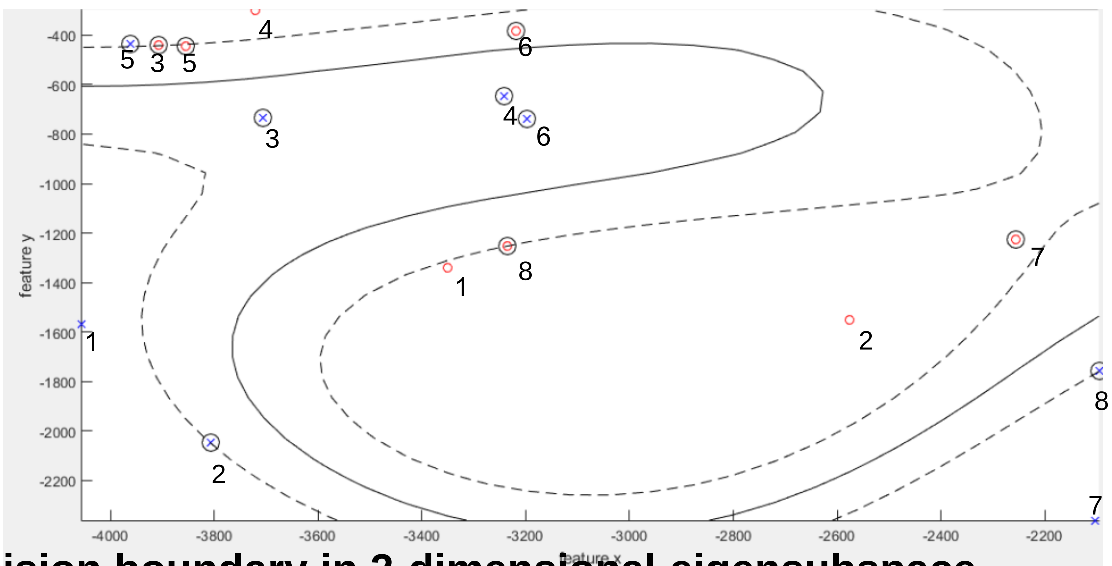


Face images and the decision boundary in 2-dimensional eigensubspace

Application to face recognition

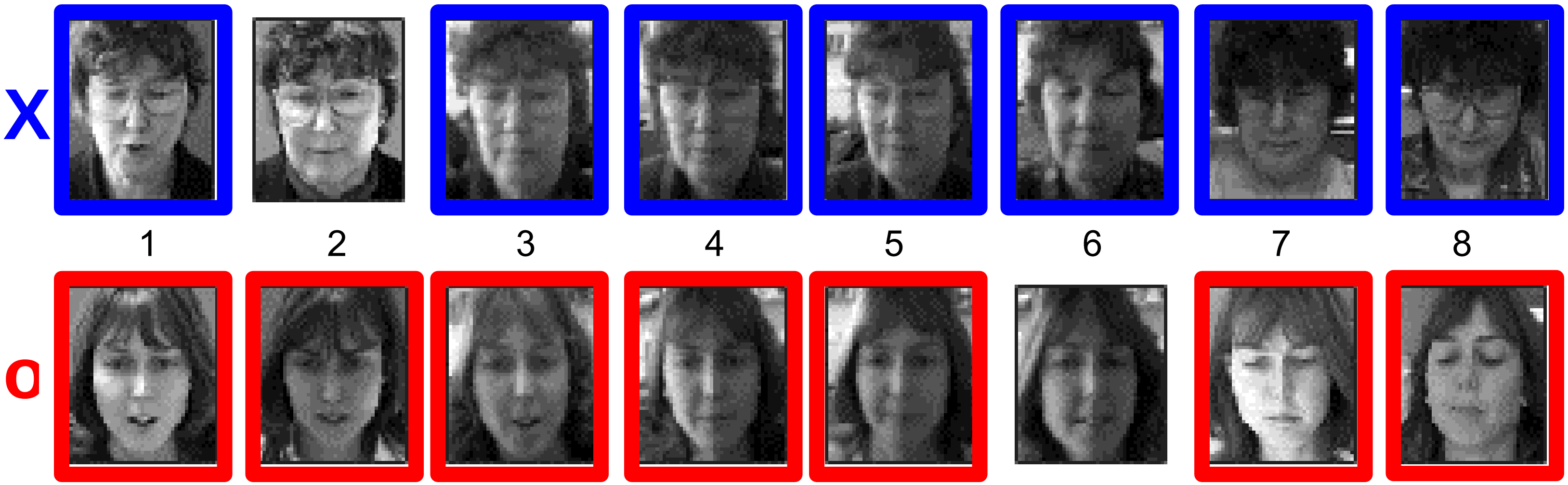


Kernel type: RBF
Kernel argument: 1000
C: 100
Number of support vectors: 11
Margin: 0.0514
Training error: 6.25%

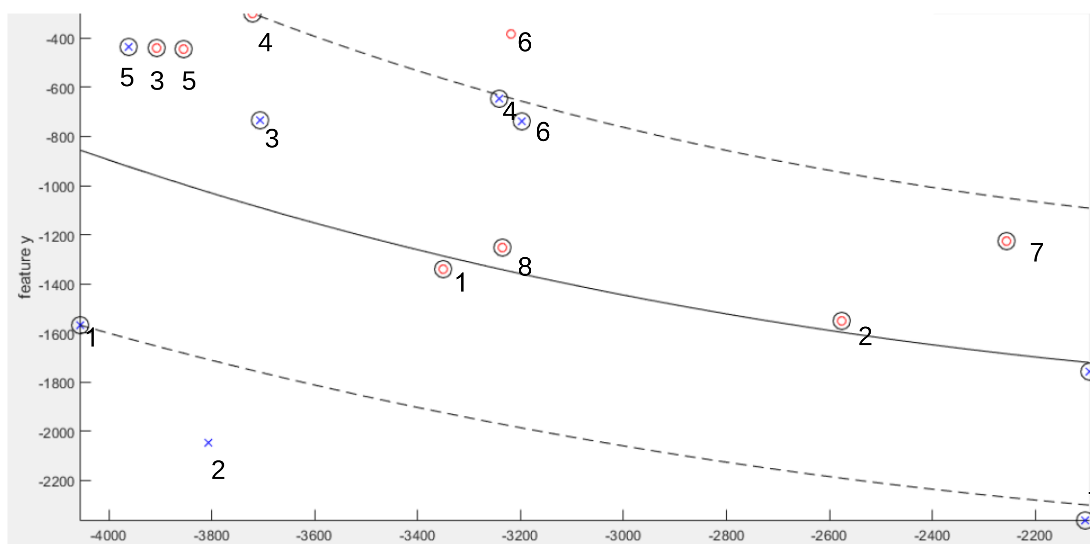


Face images and the decision boundary in 2-dimensional eigensubspace

Application to face recognition



Kernel type: RBF
Kernel argument: 5000
C: 100
Number of support vectors: 14
Margin: 0.1008
Training error: 31.25%



Face images and the decision boundary in 2-dimensional eigensubspace

https://goo.gl/m6Mnov for questions

Multi-class SVM

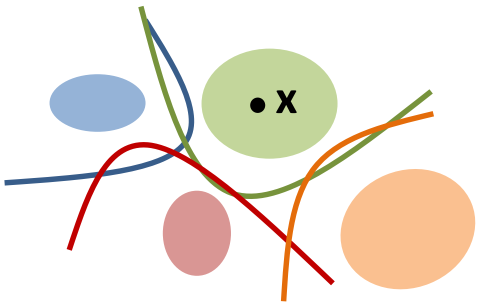
No definitive multi-class SVM formulation.
In practice, we have to obtain a multi-class SVM by combining multiple two-class SVMs.

One-versus-the-rest

- Training:
  - Given an *M*-class problem, we train *M* separate SVMs. Each distinguishes images of one category from images of all the other *M-1* categories.
  - The *m*-th SVM $y_m(\mathbf{x})$ is trained by $\mathcal{C}_m$ as the positive class and the remaining *M-1* classes as the negative class.
- Testing (max fusion) :
  - Given a query image **x**, we apply each SVM to the query and assign to it the class of the SVM that returns the highest SVM output value

$$y(\mathbf{x}) = \max_m y_m(\mathbf{x})$$

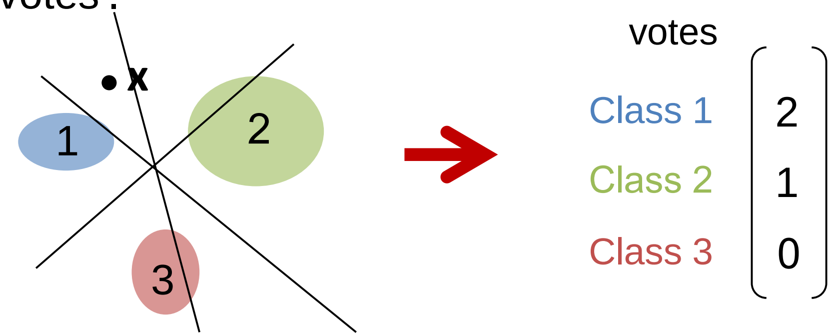
- Issues:
  - The output values $y_m(\mathbf{x})$ for different classifiers have no appropriate scales.
  - The training data sets are imbalanced.



Multi-class SVM

One-versus-one

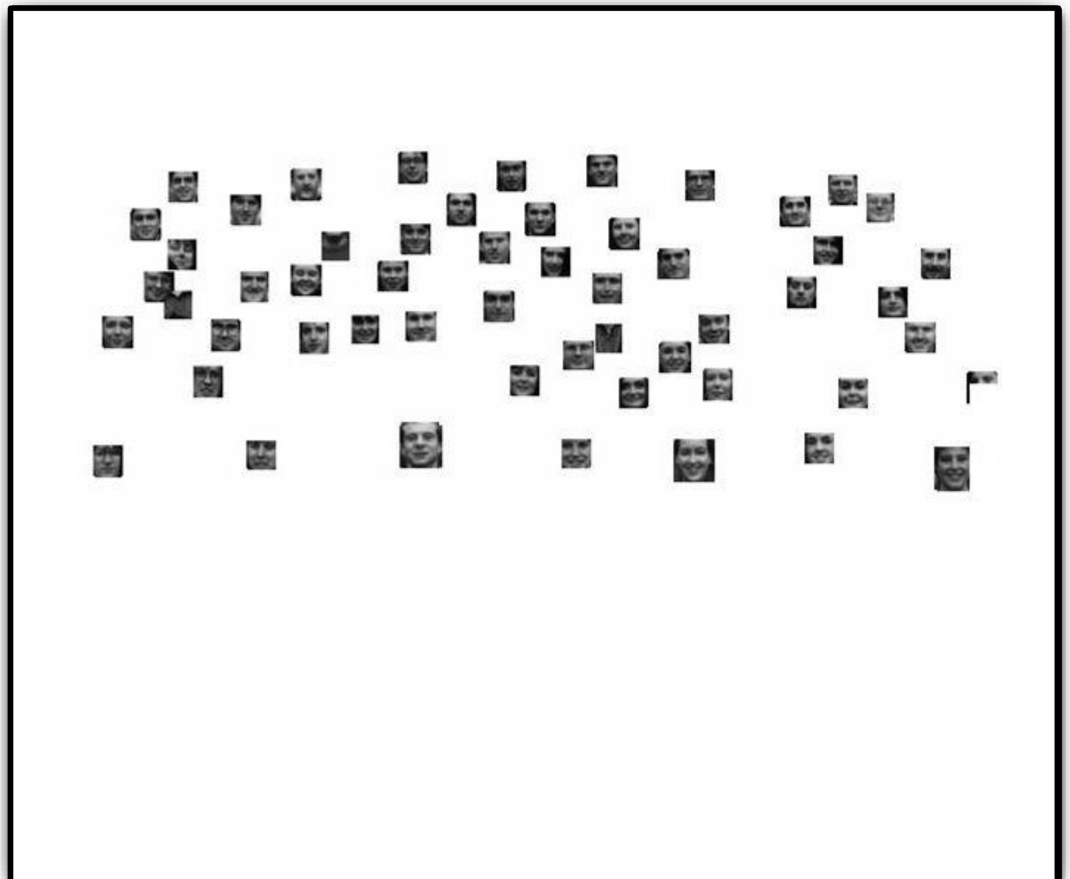
- Training:
    - Given an *M*-class problem, we train *M(M-1)/2* separate SVMs.
    - We take all possible pairs of classes.
    - An SVM is learnt for each pair of classes, i.e. $\mathcal{C}_i$ as the positive class and $\mathcal{C}_j$,j≠i as the negative class.
- Testing (majority voting fusion):
    - Each learned SVM votes for a class to assign to a query image.
    - Classification of the query image is by assigning the class has the highest number of 'votes'.

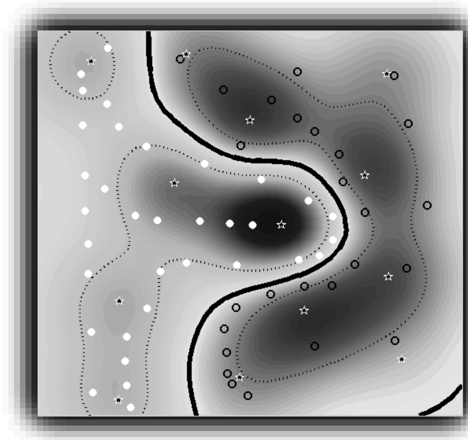


- Issues:
    - It requires a large number of SVMs. Training and testing time depends on the complexity of individual SVMs.
    - A fuzzy case happens if multiple classes receive an equal number of votes.

Application to face detection by a cascade of classifiers

- Scanning window method: we scan every scale and every pixel location in the image.
- It applies a cascade of SVM classifiers, from the simpler to the more complex, to speed up.





https://goo.gl/m6Mnov for questions

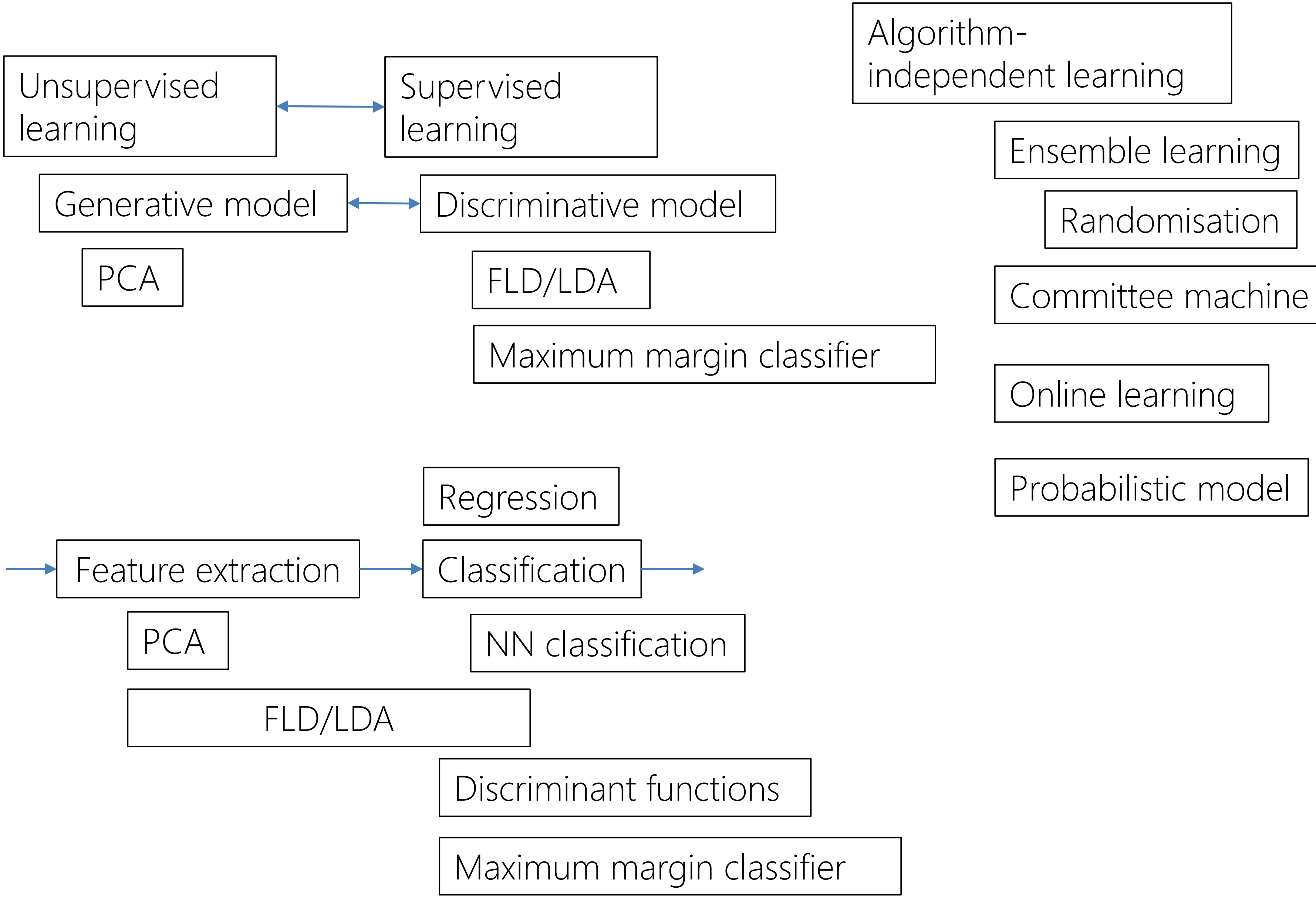
SVM pros and cons

Pros

- Many publicly available SVM packages: http://www.kernel-machines.org/software
- Kernel-based framework is very powerful, flexible
- SVMs work very well in practice, even with small training sample sizes

Cons

- No direct multi-class SVM, must combine two-class SVMs
- Computation and memory in training
  - During training time, must compute matrix of kernel values for every pair of examples
  - Learning can take a very long time for large-scale problems
- Computational cost of nonlinear SVM classification is very high: $O(KxN_{sv})$


Unsupervised learning
Supervised learning
Generative model
Discriminative model
PCA
FLD/LDA
Maximum margin classifier
Algorithm-independent learning
Ensemble learning
Randomisation
Committee machine
Online learning
Probabilistic model
Regression
Feature extraction
Classification
PCA
NN classification
FLD/LDA
Discriminant functions
Maximum margin classifier

Subspace learning

- PCA
- FLD/LDA
- Projection/pooling
- Orthonormal basis vectors
- Dimension reduction
- Data visualisation
- Reconstruction

Optimisation ↔ Randomisation

- Gradient-based
- Lagrange multipliers
- Eigenvalue problem
- Generalized eigenvalue problem
- Gaussian distributions

Underfitting → Generalization → Overfitting

- Training loss
- Testing (real) loss

Linear model ↔ Nonlinear model

- Eigenface
- Fisherface
- Kernel trick
- Feature space