ML Concepts Covered in 678

- Advanced MLP concepts: Higher Order, Batch, Classification Based, etc.
- Recurrent Neural Networks
- Support Vector Machines
- Relaxation Neural Networks
  - Hopfield Networks, Boltzmann Machines
- Deep Learning – Deep Neural Networks
- HMM (Hidden Markov Model) learning and Speech Recognition, EM algorithm
- Rule Based Learning – CN2, etc.
- Bias Variance Decomposition, Advanced Ensembles
- Semi-Supervised Learning

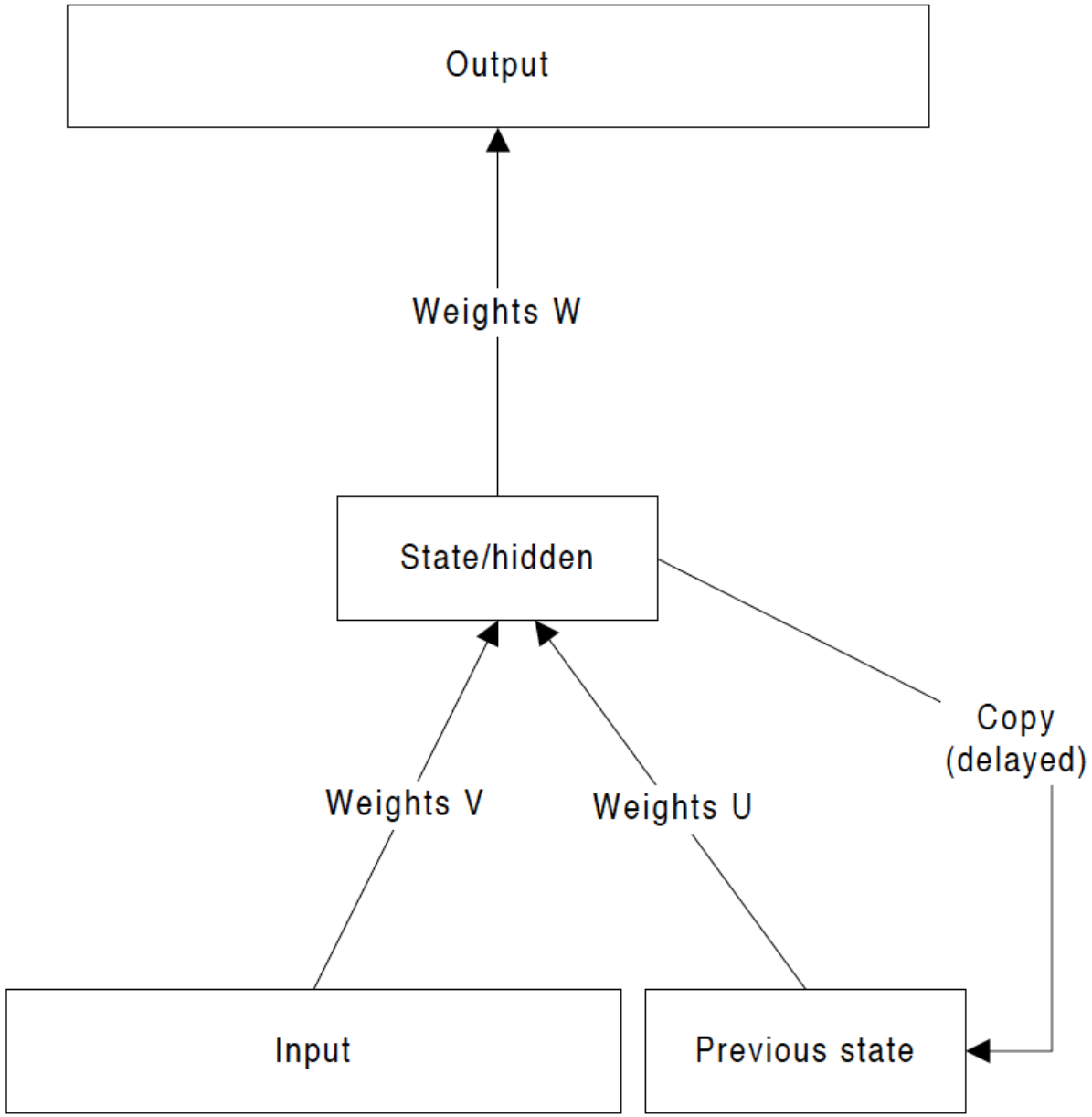
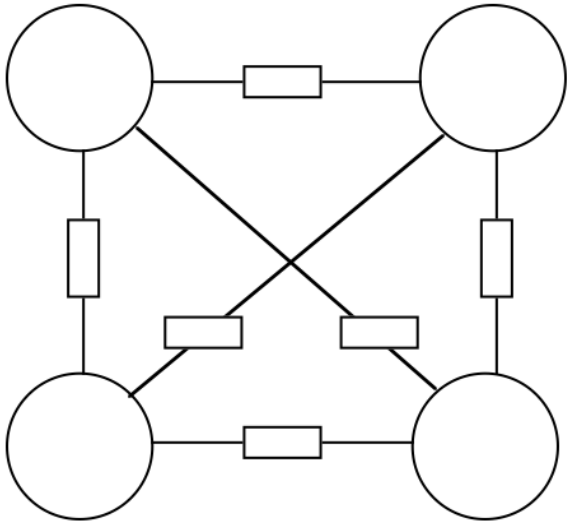


Figure 4: A simple recurrent network.

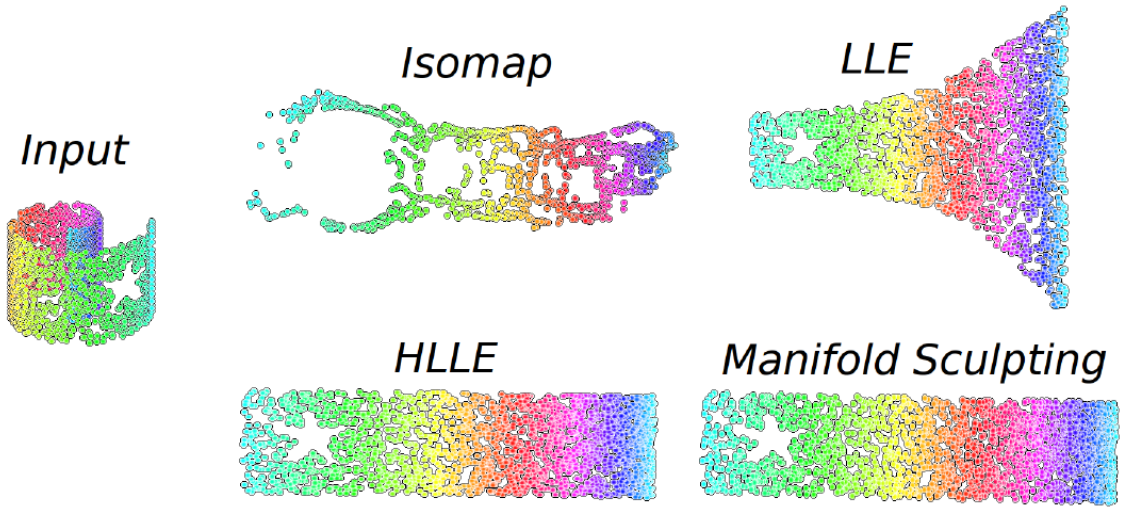
Relaxation Networks

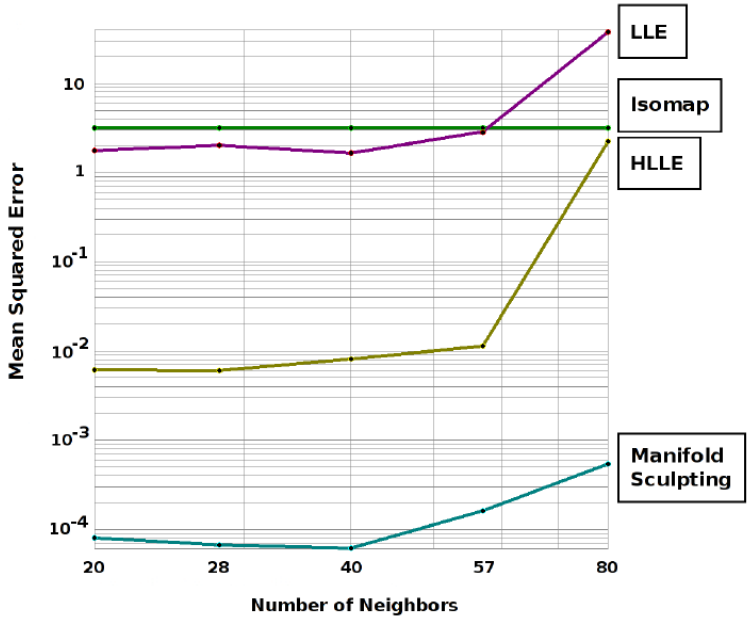


Other ML/678 Areas

- ADIB (Automatic Discovery of Inductive Bias)
- Structured Prediction, Multi-output Dependence Learning
- Manifold Learning/Non-Linear Dimensionality Reduction
- Record Linkage/Family History Directions
- Meta-Learning
- Feature Selection
- Computational Learning Theory
- Transfer Learning
- Transduction
- Other Unsupervised Learning Models
- Statistical Machine Learning Class

Manifold Sculpting





Support Vector Machines

- Elegant combination of statistical learning theory and machine learning – Vapnik
- Good empirical results
- Non-trivial implementation
- Can be slow and memory intensive
- Binary classifier
- Much current work

SVM Overview

- Non-linear mapping from input space into a higher dimensional feature space
- Linear decision surface (hyper-plane) sufficient in the high dimensional feature space (just like MLP)
- Avoid complexity of high dimensional feature space with kernel functions which allow computations to take place in the input space, while giving much of the power of being in the feature space
- Get improved generalization by placing hyper-plane at the maximum margin

Only need the support vectors since they define the decision surface, the other points can be ignored.

SVM learning finds the support vectors which optimize the maximum margin decision surface

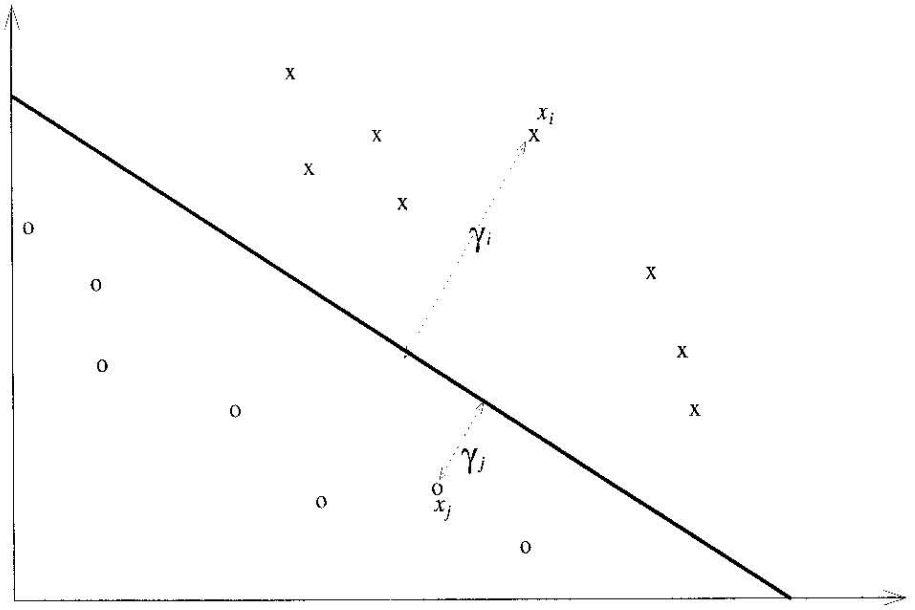


Figure 2.2: The geometric margin of two points

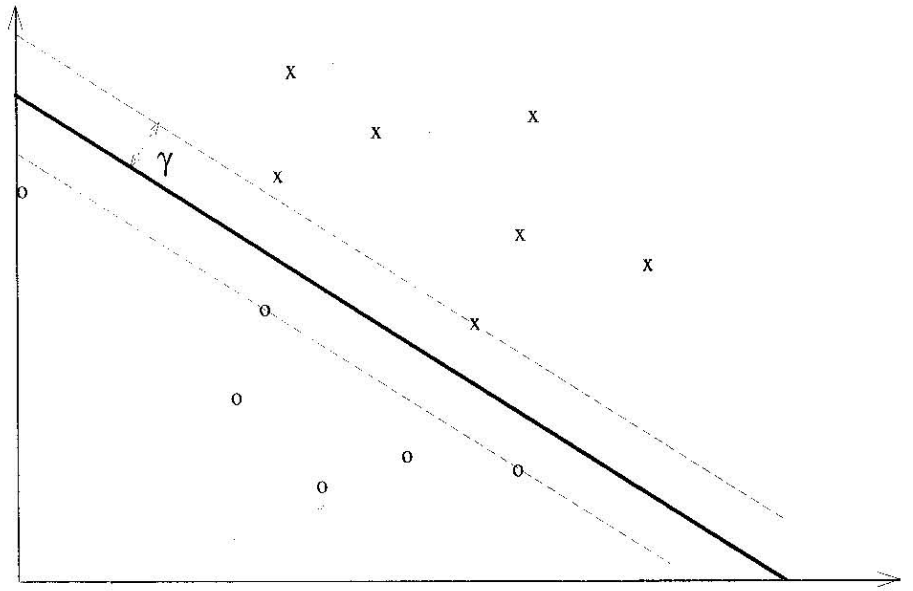


Figure 2.3: The margin of a training set

Feature Space and Kernel Functions

- Since most problems require a non-linear decision surface, we do a non-linear map $\Phi(\mathbf{x}) = (\Phi_1(\mathbf{x}),\Phi_2(\mathbf{x}), \ldots,\Phi_N(\mathbf{x}))$ from input space to feature space
- Feature space can be of very high (even infinite) dimensionality
- By choosing a proper kernel function/feature space, the high dimensionality can be avoided in computation but effectively used for the decision surface to solve complex problems - "Kernel Trick"

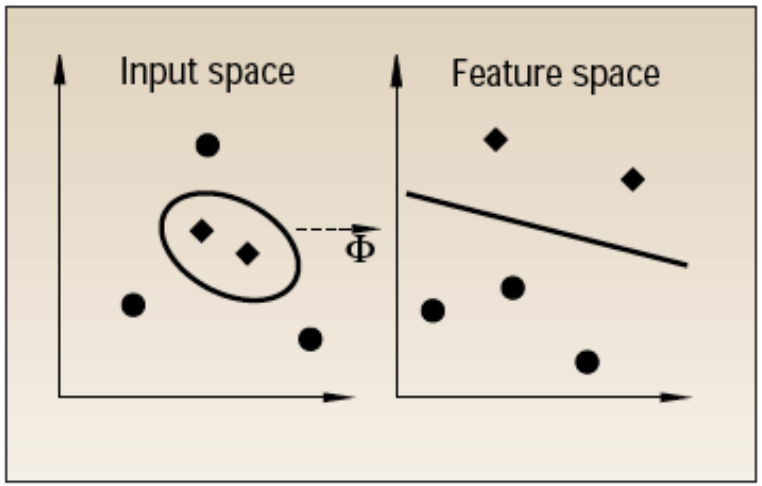


Figure 2. The idea of SV machines: map the training data nonlinearly into a higher-dimensional feature space via Φ, and construct a separating hyperplane with maximum margin there. This yields a nonlinear decision boundary in input space. By the use of a kernel function, it is possible to compute the separating hyperplane without explicitly carrying out the map into the feature space.

$$\Phi: \mathcal{R}^N \to F, \tag{4}$$

and perform the above linear algorithm in F. As I've noted, this only requires the evaluation of dot products.

$$k(\mathbf{x},\mathbf{y}) := (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})). \tag{5}$$

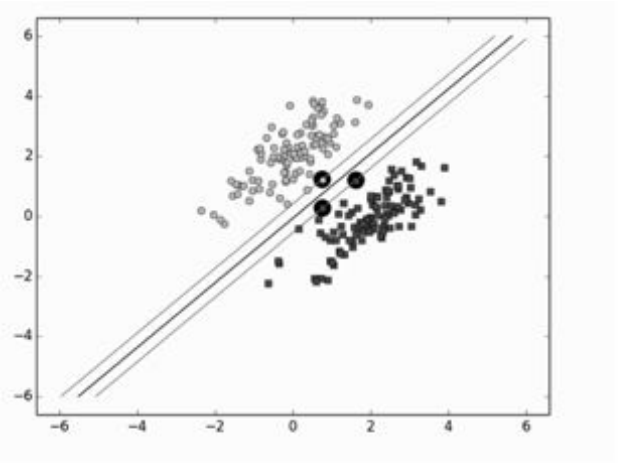
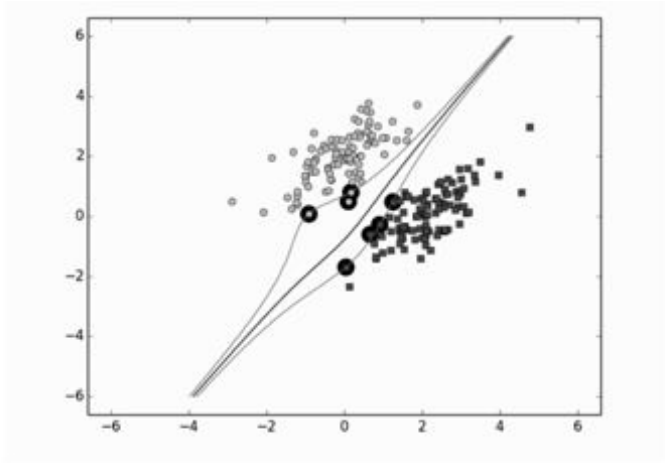
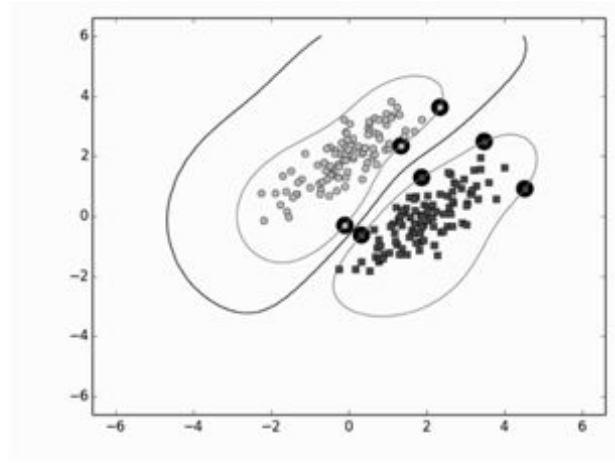
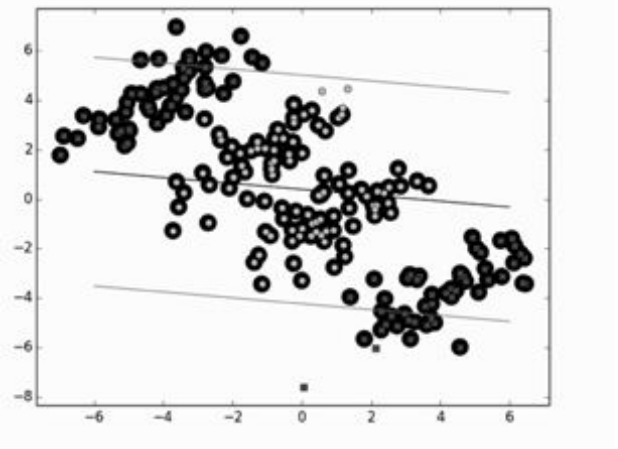
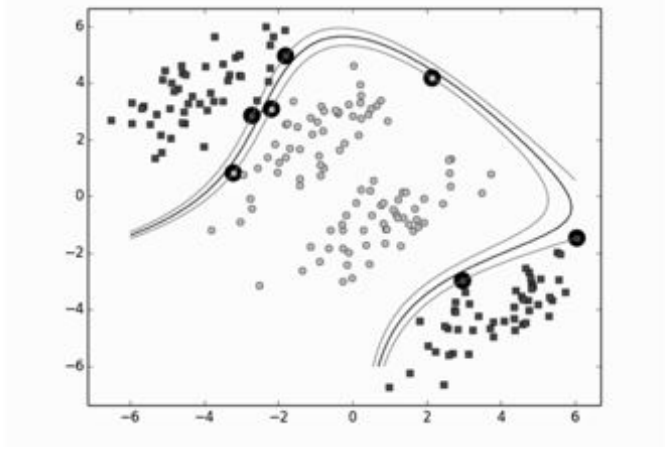
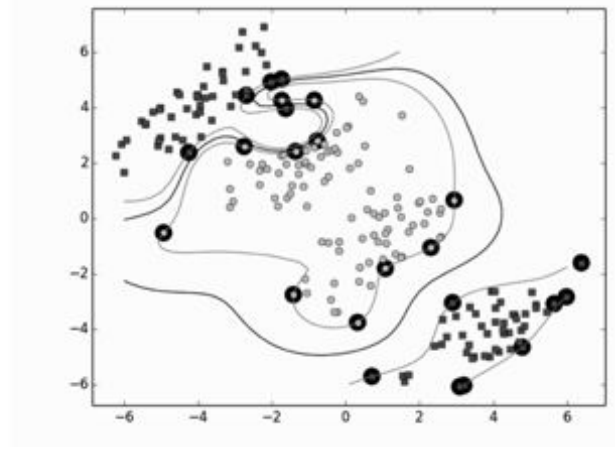
Clearly, if F is high-dimensional, the right-hand side of Equation 5 will be very expensive to compute. In some cases, however, there is a simple *kernel* k that can be evaluated efficiently. For instance, the polynomial kernel

$$k(\mathbf{x},\mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d \tag{6}$$

can be shown to correspond to a map Φ into the space spanned by all products of exactly d dimensions of $\mathcal{R}^N$. For $d=2$ and $\mathbf{x}, \mathbf{y} \in \mathcal{R}^2$, for example, we have

$$\begin{aligned}(\mathbf{x} \cdot \mathbf{y})^2 &= \left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \right)^2 \\ &= \left(\begin{pmatrix} x_1^2 \\ \sqrt{2} x_1 x_2 \\ x_2^2 \end{pmatrix} \cdot \begin{pmatrix} y_1^2 \\ \sqrt{2} y_1 y_2 \\ y_2^2 \end{pmatrix} \right) \\ &= (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})),\end{aligned} \tag{7}$$

The SVM learning about a linearly separable dataset (top row) and a dataset that needs two straight lines to separate in 2D (bottom row) with left the linear kernel, middle the polynomial kernel of degree 3, and right the RBF kernel. Remember that right two models are separating with a Hyperplane in the expanded space.

Standard SVM Approach

1. Select a 2 class training set, a kernel function, and C value (soft margin parameter)
2. Pass these to a Quadratic optimization package which will return an α for each training pattern based on the following (non-bias version). This optimization keeps weights and error both small.

$$\begin{aligned} &\textit{maximise} && W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} \alpha_i - \tfrac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j), \\ &\textit{subject to} && \sum_{i=1}^{\ell} y_i \alpha_i = 0, \\ & && C \geq \alpha_i \geq 0, \; i = 1, \ldots, \ell. \end{aligned}$$

3. Training instances with non-zero α are the support vectors for the maximum margin SVM classifier.
4. Execute by using the support vectors $$f(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b.$$

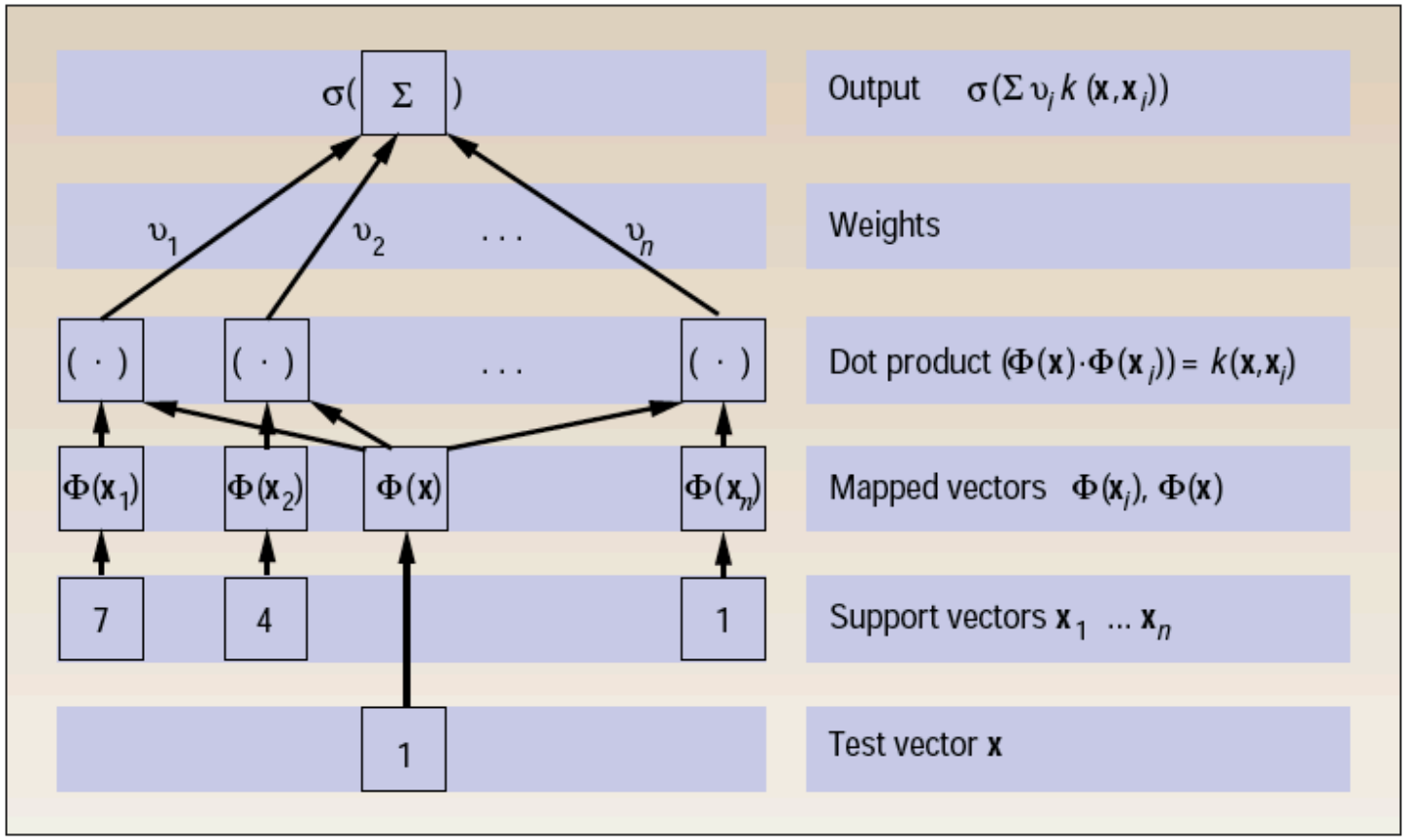


Figure 4. Architecture of SV methods. The input $\mathbf{x}$ and the support vectors $\mathbf{x}_i$ (in this example: digits) are nonlinearly mapped (by Φ) into a feature space F, where dot products are computed. By the use of the kernel k, these two layers are in practice computed in one single step. The results are linearly combined by weights ν_i, found by solving a quadratic program (in pattern recognition, $\nu_i = y_i\alpha_i$; in regression estimation, $\nu_i = \alpha^*_i - \alpha_i$) [2] or an eigenvalue problem (in kernel PCA[3]). The linear combination is fed into the function σ (in pattern recognition, $\sigma(x) = \text{sign}(x + b)$; in regression stimation, $\sigma(x) = x + b$; in kernel PCA, $\sigma(x) = x$.

Basic Kernel Execution

Primal: $$f(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi_i(\mathbf{x}) + b,$$

Dual: $$f(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i y_i \langle \boldsymbol{\phi}(\mathbf{x}_i) \cdot \boldsymbol{\phi}(\mathbf{x}) \rangle + b.$$

Definition 3.4 A *kernel* is a function K, such that for all $\mathbf{x}, \mathbf{z} \in X$

$$K(\mathbf{x}, \mathbf{z}) = \langle \boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{z}) \rangle,$$

where $\boldsymbol{\phi}$ is a mapping from X to an (inner product) feature space F.

Kernel version: $$f(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b.$$

Support vectors and weights α_i are obtained by a quadratic optimization solution

Dual vs. Primal Form

- Magnitude of α_i is an indicator of effect of pattern on weights (*embedding strength*)
- Note that patterns on borders have large α_i while easy patterns never effect the weights
- Could have trained with just the subset of patterns with $\alpha_i > 0$ (support vectors) and ignored the others
- Can train in dual. How about execution? Either way (dual can be efficient if support vectors are few and/or the feature space would be very large)

Standard (Primal) Perceptron Algorithm

- Target minus Output not used. Just add (or subtract) a portion (multiplied by learning rate) of the current pattern to the weight vector
- If weight vector starts at 0 then learning rate can just be 1
- *R* could also be 1 for this discussion

Given a linearly separable training set S and learning rate $\eta \in \mathbb{R}^+$
$\mathbf{w}_0 \leftarrow \mathbf{0}; b_0 \leftarrow 0; k \leftarrow 0$
$R \leftarrow \max_{1 \leq i \leq \ell} \|\mathbf{x}_i\|$
repeat
  for $i = 1$ to ℓ
    if $y_i(\langle \mathbf{w}_k \cdot \mathbf{x}_i \rangle + b_k) \leq 0$ then
      $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \eta y_i \mathbf{x}_i$
      $b_{k+1} \leftarrow b_k + \eta y_i R^2$
      $k \leftarrow k + 1$
    end if
  end for
until no mistakes made within the *for* loop
return $(\mathbf{w}_k, b_k)$ where k is the number of mistakes

Table 2.1: **The Perceptron Algorithm (primal form)**

Dual and Primal Equivalence

- Note that the final weight vector is a linear combination of the training patterns $$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i y_i \mathbf{x}_i,$$
- The basic decision function (primal and dual) is

$$\begin{aligned} h(\mathbf{x}) &= \operatorname{sgn}\left(\langle \mathbf{w} \cdot \mathbf{x} \rangle + b\right) \\ &= \operatorname{sgn}\left(\left\langle \sum_{j=1}^{\ell} \alpha_j y_j \mathbf{x}_j \cdot \mathbf{x} \right\rangle + b\right) \\ &= \operatorname{sgn}\left(\sum_{j=1}^{\ell} \alpha_j y_j \langle \mathbf{x}_j \cdot \mathbf{x} \rangle + b\right), \end{aligned}$$

- How do we obtain the coefficients α_i

Basic Kernel Execution

Primal: $$f(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi_i(\mathbf{x}) + b,$$

Dual: $$f(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i y_i \langle \boldsymbol{\phi}(\mathbf{x}_i) \cdot \boldsymbol{\phi}(\mathbf{x}) \rangle + b.$$

Definition 3.4 A *kernel* is a function K, such that for all $\mathbf{x}, \mathbf{z} \in X$

$$K(\mathbf{x}, \mathbf{z}) = \langle \boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{z}) \rangle,$$

where $\boldsymbol{\phi}$ is a mapping from X to an (inner product) feature space F.

Kernel version: $$f(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b.$$

Support vectors and weights α_i are obtained by a quadratic optimization solution

Choosing a Kernel

- Can start from a desired feature space and try to construct kernel
- More often one starts from a reasonable kernel and may not analyze the feature space
- Some kernels are better fit for certain problems, domain knowledge can be helpful
- Common kernels:
  - Polynomial $K(x,z) = (a\langle x \cdot z \rangle + c)^d$
  - Gaussian $K(x,z) = e^{-\gamma \|x-z\|^2}$
  - Sigmoidal $K(x,z) = \tanh(a\langle x \cdot z \rangle + c)$
  - Application specific

SVM Notes

- Excellent empirical and theoretical potential
- Some people have just used the maximum margin with a linear classifier
- Multi-class problems not handled naturally. Basic model classifies into just two classes. Can do one model for each class (class *i* is 1 and all else 0) and then decide between conflicting models using confidence, etc.
- How to choose kernel – main learning parameter other than margin penalty C. Kernel choice will include other parameters to be defined (degree of polynomials, variance of Gaussians, etc.)
- Speed and Size: both training and testing, how to handle very large training sets (millions of patterns and/or support vectors) not yet solved

Standard (Primal) Perceptron Algorithm

- Target minus Output not used. Just add (or subtract) a portion (multiplied by learning rate) of the current pattern to the weight vector
- If weight vector starts at 0 then learning rate can just be 1
- *R* could also be 1 for this discussion

Given a linearly separable training set S and learning rate $\eta \in \mathbb{R}^+$
$\mathbf{w}_0 \leftarrow \mathbf{0}; b_0 \leftarrow 0; k \leftarrow 0$
$R \leftarrow \max_{1 \leq i \leq \ell} \|\mathbf{x}_i\|$
repeat
  for $i = 1$ to ℓ
    if $y_i(\langle \mathbf{w}_k \cdot \mathbf{x}_i \rangle + b_k) \leq 0$ then
      $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \eta y_i \mathbf{x}_i$
      $b_{k+1} \leftarrow b_k + \eta y_i R^2$
      $k \leftarrow k + 1$
    end if
  end for
until no mistakes made within the *for* loop
return $(\mathbf{w}_k, b_k)$ where k is the number of mistakes

Table 2.1: **The Perceptron Algorithm (primal form)**

Dual and Primal Equivalence

- Note that the final weight vector is a linear combination of the training patterns $$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i y_i \mathbf{x}_i,$$
- The basic decision function (primal and dual) is

$$\begin{aligned} h(\mathbf{x}) &= \operatorname{sgn}\left(\langle \mathbf{w} \cdot \mathbf{x} \rangle + b\right) \\ &= \operatorname{sgn}\left(\left\langle \sum_{j=1}^{\ell} \alpha_j y_j \mathbf{x}_j \cdot \mathbf{x} \right\rangle + b\right) \\ &= \operatorname{sgn}\left(\sum_{j=1}^{\ell} \alpha_j y_j \langle \mathbf{x}_j \cdot \mathbf{x} \rangle + b\right), \end{aligned}$$

- How do we obtain the coefficients α_i

Dual Perceptron Training Algorithm

- Assume initial 0 weight vector

Given training set S
$\boldsymbol{\alpha} \leftarrow \mathbf{0}$; $b \leftarrow 0$
$R \leftarrow \max_{1 \le i \le \ell} \|\mathbf{x}_i\|$
repeat
  for $i = 1$ to ℓ
    if $y_i \left(\sum_{j=1}^{\ell} \alpha_j y_j \langle \mathbf{x}_j \cdot \mathbf{x}_i \rangle + b \right) \le 0$ then
      $\alpha_i \leftarrow \alpha_i + 1$
      $b \leftarrow b + y_i R^2$
    end if
  end for
until no mistakes made within the *for* loop
return $(\boldsymbol{\alpha}, b)$ to define function $h(\mathbf{x})$ of equation (2.1)

Table 2.2: **The Perceptron Algorithm (dual form)**

Dual vs. Primal Form

- Gram Matrix: all $(x_i \cdot x_j)$ pairs – Done once and stored (can be large)
- α_i: One for each pattern in the training set. Incremented each time it is misclassified, which would lead to a weight change in primal form
- Magnitude of α_i is an indicator of effect of pattern on weights (*embedding strength*)
- Note that patterns on borders have large α_i while easy patterns never effect the weights
- Could have trained with just the subset of patterns with $\alpha_i > 0$ (support vectors) and ignored the others
- Can train in dual. How about execution? Either way (dual can be efficient if support vectors are few)

Polynomial Kernels

$$\langle \mathbf{x} \cdot \mathbf{z} \rangle^2 = \left(\sum_{i=1}^{n} x_i z_i \right)^2 = \left(\sum_{i=1}^{n} x_i z_i \right) \left(\sum_{j=1}^{n} x_j z_j \right)$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j z_i z_j = \sum_{(i,j)=(1,1)}^{(n,n)} \left(x_i x_j \right) \left(z_i z_j \right),$$

which is equivalent to an inner product between the feature vectors

$$\phi(\mathbf{x}) = \left(x_i x_j \right)_{(i,j)=(1,1)}^{(n,n)}.$$

In this case the features are all the monomials of degree 2

- For greater dimensionality can do

$$K(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x} \cdot \mathbf{z} \rangle^d \text{ and } K(\mathbf{x}, \mathbf{z}) = \left(\langle \mathbf{x} \cdot \mathbf{z} \rangle + c \right)^d.$$

Kernel Trick Realities

- Polynomial Kernel - all monomials of degree 2
  - $x_1x_3y_1y_2 + x_1x_3y_1y_3 + ...$ (all 2nd order terms)
  - $K(x,z) = \langle \Phi_1(\mathbf{x}) \cdot \Phi_2(\mathbf{x}) \rangle = (x_1x_3)(y_1y_2) + (x_1x_3)(y_1y_3) + ...$
  - Lot of stuff represented with just one $\langle \mathbf{x} \cdot \mathbf{z} \rangle^2$
- However, in a full higher order solution we would would like separate coefficients for each of these second order terms, including weighting within the terms (i.e. $(2x_1)(y_1 \cdot 3y_2)$)
- SVM just sums them all with individual coefficients of 1
  - Thus, not as powerful as a second order system with arbitrary weighting
  - This kind of arbitrary weighting can be done in an MLP because learning is done in the layers between inputs and hidden nodes
  - Of course, individual weighting requires a theoretically exponential increase in terms/hidden nodes which we need to find weights for as the polynomial degree increases. Also learning algorithms which can actually find these most salient higher-order features.

Maximum Margin

- Maximum margin can lead to overfit due to noise
- Problem may not be linearly separable within a reasonable feature space
- Soft Margin is a common solution, allows slack variables
- α_i constrained to be >= 0 and less than C. The C allows outliers. How to pick C. Can try different values for the particular application to see which works best.

Soft Margins

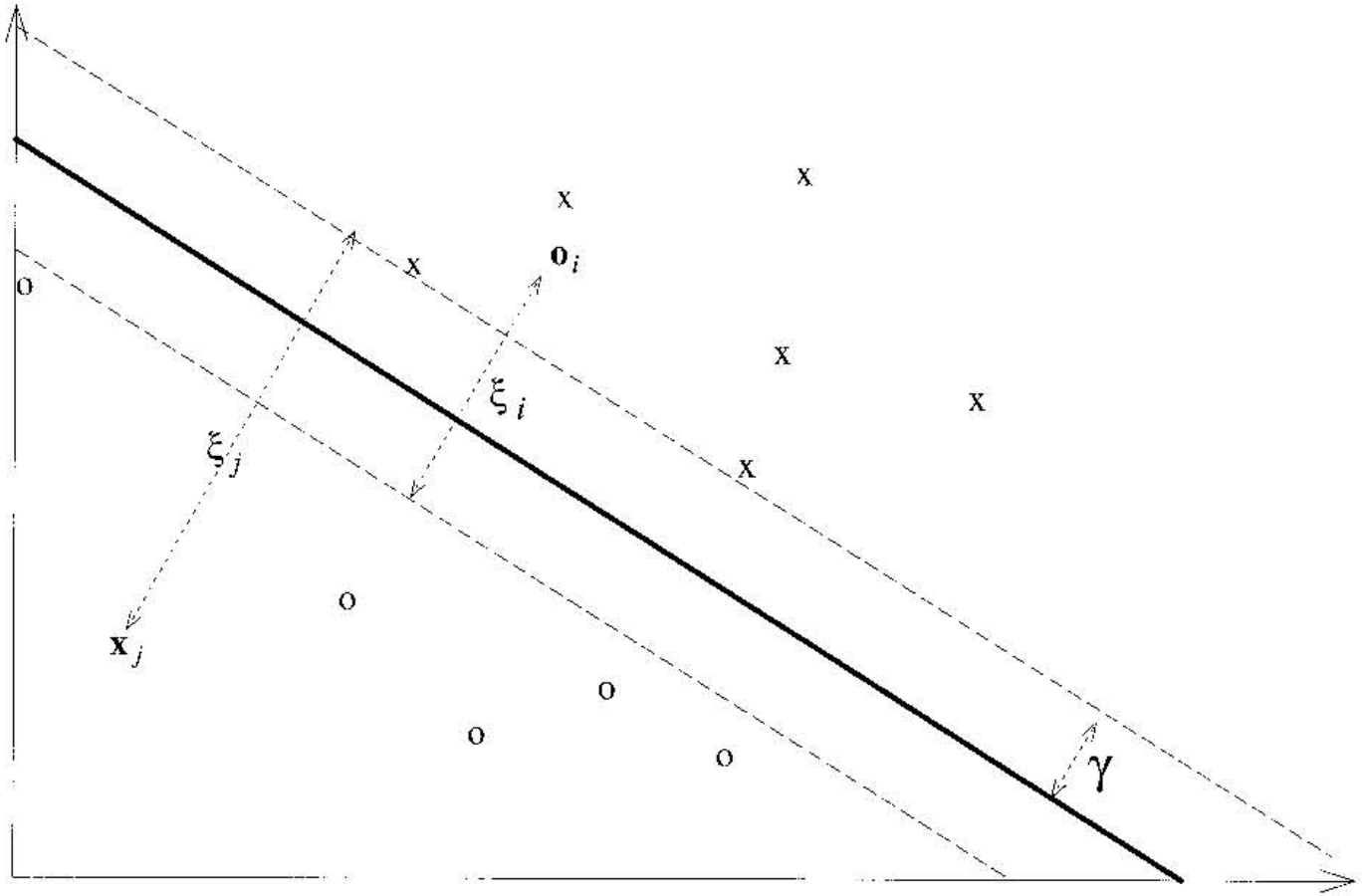


Figure 2.4: The slack variables for a classification problem

Quadratic Optimization

- Optimizing the margin in the higher order feature space is convex and thus there is one guaranteed solution at the minimum (or maximum)
- SVM Optimizes the dual representation (avoiding the higher order feature space)

$$\begin{aligned} &\textit{maximise} && W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} \alpha_i - \tfrac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j), \\ &\textit{subject to} && \sum_{i=1}^{\ell} y_i \alpha_i = 0, \\ & && C \geq \alpha_i \geq 0, \; i = 1, \ldots, \ell. \end{aligned}$$

- The optimization is quadratic in the α_i terms and linear in the constraints – can drop C maximum for non soft margin
- While quite solvable, requires complex code and usually done with a purchased numerical methods software package – Quadratic programming

Execution

- Typically use dual form
- If the number of support vectors is small then dual is fast
- In cases of low dimensional feature spaces, could derive weights from α_i and use normal primal execution
- Approximations to dual are possible to obtain speedup (smaller set of prototypical support vectors)

Standard SVM Approach

1. Select a 2 class training set, a kernel function (calculate the Gram Matrix), and C value (soft margin parameter)
2. Pass these to a Quadratic optimization package which will return an α for each training pattern based on the following (non-bias version)

$$\begin{aligned} &\textit{maximise} && W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} \alpha_i - \tfrac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j), \\ &\textit{subject to} && \sum_{i=1}^{\ell} y_i \alpha_i = 0, \\ & && C \geq \alpha_i \geq 0,\ i = 1, \ldots, \ell. \end{aligned}$$

3. Patterns with non-zero α are the support vectors for the maximum margin SVM classifier.
4. Execute by using the support vectors $$f(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b.$$

A Simple On-Line Approach

- Stochastic on-line gradient ascent
- Can be effective
- This version assumes no bias
- Sensitive to learning rate
- Stopping criteria tests whether it is an appropriate solution – can just go until little change is occurring or can test optimization conditions directly
- Can be quite slow and usually quadratic programming is used to get an exact solution
- Newton and conjugate gradient techniques also used – Can work well since it is a guaranteed convex surface – bowl shaped

- Maintains a margin of 1 (typical in standard implementation) which can always be done by scaling $\boldsymbol{\alpha}$ or equivalently $\mathbf{w}$ and b
- Change update to not increase α_i if term in parenthesis is > 1

Given training set S and learning rates $\boldsymbol{\eta} \in (\mathbb{R}^+)^\ell$
$\boldsymbol{\alpha} \leftarrow \mathbf{0}$
repeat
    for $i = 1$ to ℓ
        $\alpha_i \leftarrow \alpha_i + \eta_i \left(1 - y_i \sum_{j=1}^{\ell} \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j)\right)$
        if $\alpha_i < 0$ then $\alpha_i \leftarrow 0$
        else
        if $\alpha_i > C$ then $\alpha_i \leftarrow C$
    end for
until stopping criterion satisfied
return $\boldsymbol{\alpha}$

Table 7.1: **Simple on-line algorithm for the 1-norm soft margin**

Large Training Sets

- Big problem since the Gram matrix (all $(x_i \cdot x_j)$ pairs) is $O(n^2)$ for n data patterns
  - 10^6 data patterns require 10^{12} memory items
  - Can't keep them in memory
  - Also makes for a huge inner loop in dual training
- Key insight: most of the data patterns will not be support vectors so they are not needed

Chunking

- Start with a reasonably sized subset of the Data set (one that fits in memory and does not take too long during training)
- Train on this subset and just keep the support vectors or the m patterns with the highest α_i values
- Grab another subset, add the current support vectors to it and continue training
- Note that this training may allow previous support vectors to be dropped as better ones are discovered
- Repeat until all data is used and no new support vectors are added or some other stopping criteria is fulfilled