SUPPORT VECTOR MACHINES

Primary Source: Burges 1998 and Haykin 2009

- Support Vector Machines (SVMs) are a category of universal feed-forward networks pioneered by Vapnik (Vapnik, 1995, 1998).
  - Central to the development of the SV learning algorithm is the inner-product kernel between a "support vector" $\mathbf{x}_i$ and a vector $\mathbf{x}$ drawn from the input data space.
  - SVs consist of a small sub-set of data points extracted by the learning algorithm from the training sample itself
  - Because of this central property of the learning algorithm, SVM is also referred to as a kernel method.
  - The kernel method basic to the design of SVM is optimal, with the optimality being rooted in convex optimization.
- Although the subject of Support Vector Machines (SVMs) can be said to have started in the late seventies by Vladamir Vapnik (Vapnik 1979), only recently has it begun to receive increasing attention.
  - The first original English manuscript that introduced the methods to the broader western audience is the 1995 classic by Vapnik (Vapnik 1995) that cast SVMs as derivatives of the *Statistical Learning Theory*[1].
- As Bernhard Schölkopf (1998) put it:

  "SVM methods are very useful in two respects. First, they are quite satisfying from a theoretical point of view. Second, they can lead to high performances in practical applications. In that sense, the SVM algorithm can be considered as lying at the intersection of learning theory and practice."
- In recent years, SVMs have been used for solving a variety of problems, including classification, pattern recognition, regression, and time-series forecasting tasks.
- When tested on numerous benchmark datasets in these areas, in most of the cases, SVM generalization performance (i.e. error rates on hidden sets) either matches or is significantly better than that of competing statistical and machine learning methods (Burges 1998).

STRUCTURAL RISK MINIMIZATION AND SUPPORT VECTOR MACHINES

- SVM constructs a decision surface hyperplane that maximizes the separation margin by following a principled approach rooted in statistical learning theory.
- More precisely, SVM is an approximate implementation of the *method of structural risk minimization*[2] (Haykin 1999).
- Suppose the task of a machine is to learn the mapping $\mathbf{x}_i \mapsto y_i$.
- The machine is actually defined by a set of possible mappings $\mathbf{x}_i \mapsto f(\mathbf{x},\boldsymbol{\alpha})$, where the functions $f(\mathbf{x},\boldsymbol{\alpha})$ themselves are labeled by the adjustable parameters $\boldsymbol{\alpha}$.
- The expectation of the test error or 'expected risk' for a trained machine is therefore:

$$R(\boldsymbol{\alpha}) = \frac{1}{2}\int \left| y - f(\mathbf{x},\boldsymbol{\alpha}) \right| p(\mathbf{x},y)\, d\mathbf{x}\, dy \qquad (1)$$

- In the absence of an estimate for $p(\mathbf{x},y)$, in spite of its elegance, this expression is not very useful.
- In the absence of knowledge of $p(\mathbf{x},y)$, one could alternately work with an 'empirical risk' measure $R_{emp}(\boldsymbol{\alpha})$ that can be expressed as follows:

$$R_{emp}(\boldsymbol{\alpha}) = \frac{1}{2N}\sum_{i=1}^{N} \left| y_i - f(\mathbf{x}_i,\boldsymbol{\alpha}) \right| \qquad (2)$$

  where N denotes the number of training patterns.
- Now, choose some η such that $0 \le \eta \le 1$. Then, for losses taking these values, with probability $1-\eta$, the following bound holds (Vapnik 1998):

$$R(\boldsymbol{\alpha}) \le R_{emp}(\boldsymbol{\alpha}) + \sqrt{\frac{h}{N}\left[\log\left(\frac{2N}{h}\right)+1\right] - \frac{1}{N}\log\left(\frac{\eta}{4}\right)} \qquad (3)$$

  where h is a non-negative integer called the Vapnik-Chervonenkis (VC) dimension and is a measure of the capacity of the learning machine.
  - It is defined as the cardinality of the largest set of points that the algorithm can shatter.
  - Example: Consider a straight line as the binary classification model (e.g., the model used by a perceptron). The line should separate +ve data points from –ve data points. There exist sets of 3 points that can indeed be shattered using this model (any 3 points that are not collinear can be shattered). However, no set of 4 points can be shattered. Thus, the VC dimension of this particular classifier is 3.

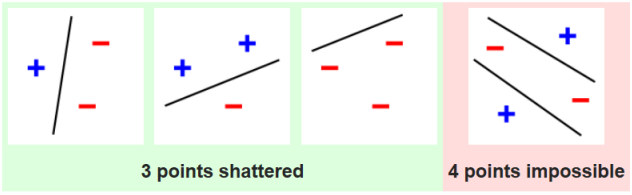


Figure illustrates only 3 of the $2^3 = 8$ possible label assignments for the 3-point case.

- In the literature, the right hand side of equation (3) is referred to as the 'risk bound' and the second term on the right hand side is called the 'VC confidence'.
- With the knowledge of h, one can easily compute the right hand side. Thus, given several different learning machines and a fixed (sufficiently small) η, selecting the machine that minimizes the right hand side leads to a machine that gives the lowest upper bound on the actual risk.
- This procedure, in essence, results in a principled method that is essential to the idea of structural risk minimization.
- In the case of separable patterns, a SVM produces a value of zero for the first term on the right hand side of equation (3) and minimizes the second term, thus minimizing the overall risk.
- In the case of nonseparable patterns, the cost function that is to be minimized will now carry an extra term weighted by parameter C that penalizes any SVM that generates too many nonseparable patterns.

LINEAR SUPPORT VECTOR MACHINES

Optimal Hyperplane for Linearly Separable Patterns:

- SVMs are perhaps easiest to explain by starting with the case of *separable* patterns that could arise in the context of binary *pattern classification*.
- In binary pattern classification, the task of the model when presented with an input vector is to classify the vector as belonging to one of two possible classes.
- Label the training data $\{\mathbf{x}_i, y_i\}$, $i = 1,\ldots,l$, $y_i \in \{-1,1\}$, $\mathbf{x}_i \in \mathbf{R}^n$.

[1] Statistical learning theory is a framework for machine learning drawing from the fields of statistics and functional analysis. Statistical learning theory deals with the problem of finding a predictive function based on data. - Wikipedia

[2] Structural risk minimization (SRM) is an inductive principle of use in machine learning that aims for good generalization. The SRM principle addresses this problem by balancing the model's complexity against its success at fitting the training data. - Wikipedia

- Suppose there exists some hyperplane that separates the positive examples from the negative examples (a *separating hyperplane*).

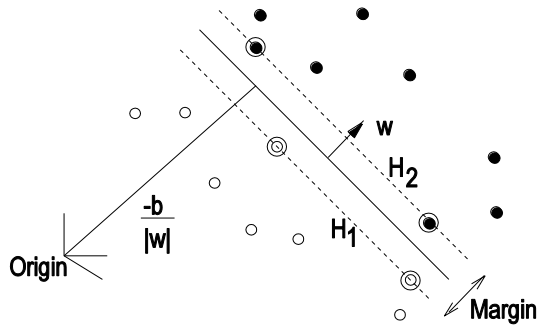


Illustration of the idea of an optimal hyperplane for separable patterns.

- The points $\mathbf{x}$ that lie on the hyperplane satisfy $\mathbf{w}\cdot\mathbf{x}+b=0$, where $\mathbf{w}$ is normal to the hyperplane, $|b|/\|\mathbf{w}\|$ is the perpendicular distance from the hyperplane to the origin, and $\|\mathbf{w}\|$ is the Euclidean norm of $\mathbf{w}$.
- Let d_+ (d_-) be the shortest distance from the separating hyperplane to the closest positive (negative) example.
- Define the *margin* of a separating hyperplane to be $d_+ + d_-$.
- For the linearly separable case, the support vector algorithm simply looks for the separating hyperplane with the largest margin.
  Mathematical Formulation:
  Find the optimal pair of hyperplanes that give the maximum margin by minimizing $\|\mathbf{w}\|^2$, subject to the following constraints:
  $$\mathbf{x}_i\cdot\mathbf{w}+b\geq+1 \text{ for } y_i=+1 \quad (4)$$
  $$\mathbf{x}_i\cdot\mathbf{w}+b\leq-1 \text{ for } y_i=-1 \quad (5)$$
  Or equivalently,
  $$y_i(\mathbf{x}_i\cdot\mathbf{w}+b)-1\geq 0 \ \forall i. \quad (6)$$
- The points for which the equality in equation (4) holds lie on the hyperplane $H_1:\ \mathbf{x}_i\cdot\mathbf{w}+b=1$ and the points for which the equality in equation (5) holds lie on the hyperplane $H_2:\ \mathbf{x}_i\cdot\mathbf{w}+b=-1$.
- Note that H_1 and H_2 are parallel and that no training points fall between them.
- Those training points for which the equality in equation (6) holds are called *support vectors* (indicated in figure by the extra circles).
- One can now switch to a Lagrangian formulation of the problem. There are two reasons for doing this (Burges 1998):
  - First, the constraints from equation (6) will be replaced by constraints on the Lagrange multipliers themselves, a case that is much easier to handle.
  - Second, the training data will only appear (in the actual training and test algorithms) in the form of dot products between vectors in this reformulation of the problem. This crucial property allows one to generalize the procedure to the nonlinear case.
- Introduce positive Lagrange multipliers α_i, $i=1,\ldots,l$, one for each of the inequality constraints in equation (6).
- Recall that the rule is that for constraints of the form $c_i\geq 0$, the constraint equations are multiplied by positive Lagrange multipliers and subtracted from the objective function, to form the Lagrangian. For equality constraints, the Lagrange multipliers are unconstrained.
- This leads to the following Lagrangian:
  $$L_P\equiv\frac{1}{2}\|\mathbf{w}\|^2-\sum_{i=1}^{l}\alpha_i y_i(\mathbf{x}_i\cdot\mathbf{w}+b)+\sum_{i=1}^{l}\alpha_i\ . \quad (7)$$
- One must now *minimize* L_P with respect to $\mathbf{w}$, b, and simultaneously require that the derivatives of L_P with respect to all the i vanish, all subject to the constraints $\alpha_i\geq 0$.
- This is a convex quadratic programming problem, since the objective function is itself convex and those points that satisfy the constraints also form a convex set.
- One can equivalently solve the following *dual* problem: *maximize* L_P, subject to the constraints that the gradient of L_P with respect to $\mathbf{w}$ and b vanish, and subject to constraints that the $\alpha_i\geq 0$.
- The requirement that the gradient of L_P with respect to $\mathbf{w}$ and b vanish, leads to the following conditions:
  $$\mathbf{w}=\sum_i\alpha_i y_i\mathbf{x}_i \quad (8)$$
  $$\sum_i\alpha_i y_i=0 \quad (9)$$
- Since these are equality constraints in the dual formulation, one can substitute them into equation (7) to give
  $$L_D=\sum_i\alpha_i-\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j(\mathbf{x}_i\cdot\mathbf{x}_j). \quad (10)$$
- Support vector training for the linear separable case therefore amounts to maximizing L_D with respect to α_i, subject to constraints from equation (9) and positivity of the α_i, with the solution given by equation (8).
- Notice that there is a Lagrange multiplier α_i for every training point.
- In the solution, those points for which $\alpha_i>0$ are "support vectors", and lie on one of the hyperplanes H_1, H_2. All other training points have $\alpha_i=0$ and lie either on H_1 or H_2 (such that the equality in equation (6) holds), or on that side of H_1 or H_2 such that the strict inequality in equation (6) holds.
- In SVMs, the support vectors are the critical elements of the training set. They lie closest to the decision boundary; if all other training points were removed (or moved around, but so as not to cross H_1 or H_2), and training was repeated, the same separating hyperplane would be found.
- The Karush-Kuhn-Tucker (KKT) conditions play a central role in both the theory and practice of constrained optimization.
- The KKT conditions are satisfied at the solution of any constrained optimization problem (convex or not), with any kind of constraints, if the intersection of the set of feasible directions with the set of descent directions coincides with the intersection of the set of feasible directions for *linearized constraints* with the set of descent directions (see Fletcher (1987)).
- This rather technical regularity assumption holds for all support vector machines, since the constraints are always linear.
- Furthermore, the problem for SVMs is convex (a convex objective function, with constraints which give a convex feasible region), and for convex problems (if the regularity condition holds), the KKT conditions are *necessary and sufficient* for $\mathbf{w}$, b, α to be a solution (Fletcher 1987).
- Thus, solving the SVM problem is equivalent to finding a solution to the KKT conditions. This fact results in several approaches to finding the solution.

- As an immediate application, note that while $\mathbf{w}$ is explicitly determined by the training procedure, the threshold b is not (although it is implicitly determined). However, b is easily found by using the KKT 'complementarity' condition,

$$\alpha_i \left(y_i (\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \right) = 0 \quad \forall i \tag{11}$$

by choosing any i for which $\alpha_i \neq 0$ and computing b.

Optimal Hyperplane for Non-Separable Patterns:

- The above algorithm for separable data, when applied to non-separable data, will find no feasible solution.
- One can extend these ideas to handle non-separable data by relaxing the constraints from equation (4) and equation (5), when necessary. That is, one should introduce a further cost component (i.e. an increase in the primal objective function) for doing so.
- This can be done by introducing positive slack variables ξ_i, $i = 1,\ldots,l$ in the constraints (Cortes and Vapnik 1995), which then become:

$$\mathbf{x}_i \cdot \mathbf{w} + b \geq +1 - \xi_i \text{ for } y_i = +1 \tag{12}$$

$$\mathbf{x}_i \cdot \mathbf{w} + b \leq -1 + \xi_i \text{ for } y_i = -1 \tag{13}$$

$$\xi_i \geq 0 \quad \forall i. \tag{14}$$

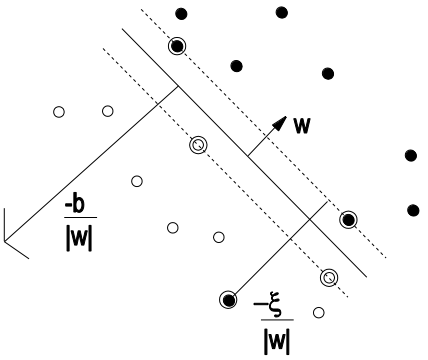


Illustration of the idea of an optimal hyperplane for non-separable patterns.

- Thus, for an error to occur, the corresponding ξ_i must exceed unity, so $\sum_i \xi_i$ is an upper bound on the number of training errors.
- A natural way to assign an extra cost for errors is to change the objective function to be minimized from $\frac{1}{2}\|\mathbf{w}\|^2$ to $\frac{1}{2}\|\mathbf{w}\|^2 + C\left(\sum_i \xi_i\right)^k$, where C is a parameter to be chosen by the user, a larger C corresponding to assigning a higher penalty to errors.
- This is a convex programming problem for any positive integer k; for $k = 2$ and $k = 1$ it is also a quadratic programming problem, and the choice $k = 1$ has the further advantage that neither the ξ_i, nor their Lagrange multipliers, appear in the dual problem, that becomes:

Maximize:

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \tag{15}$$

subject to:

$$0 \leq \alpha_i \leq C, \tag{16}$$

$$\sum_i \alpha_i y_i = 0 \tag{17}$$

- The solution is again given by

$$\mathbf{w} = \sum_{i=1}^{N_S} \alpha_i y_i \mathbf{x}_i \tag{18}$$

where N_S is the number of support vectors.

- Thus, the only difference from the optimal hyperplane case is that the α_i now have an upper bound of C.
- Care should be exercised in selecting C; it should be optimized.
  - Given that the range to be explored for C might be large, consider searching uniformly on a log scale (e.g., $C = 10^n$ for n = -10 to 10). Once you find the best C parameter that yields good results, increase the resolution for C around the promising setting.

Non-Linear Support Vector Machines

- Boser, Guyon, and Vapnik (1992) showed that a rather old trick (Aizerman, Braverman, and Rozoner 1964) can be used to generalize the linear case to the case where the decision function is not a linear function of the data.
- First, notice that the only way in which the data appears in the training problem, equations (15)-(17), are in the form of dot products, $\mathbf{x}_i \cdot \mathbf{x}_j$.
- Suppose that we first mapped the data to some other (possibly infinite dimensional) Euclidean space H, using a mapping which we will call Φ:

$$\Phi : \mathbf{R}^d \mapsto \mathrm{H}. \tag{19}$$

- Then, the training algorithm would only depend on the data through dot products in H, i.e. on functions of the form $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$.
- Now if there were a "kernel function" K such that $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$, one would only need to use K in the training algorithm, and would never need to explicitly know Φ.
- One example is

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma^2}. \tag{20}$$

- In this particular example, H is infinite dimensional, so it would not be very easy to work with Φ explicitly. However, if one replaces $\mathbf{x}_i \cdot \mathbf{x}_j$ by $K(\mathbf{x}_i, \mathbf{x}_j)$ everywhere in the training algorithm, the algorithm will produce a support vector machine that lives in an infinite dimensional space, and furthermore, do so in roughly the same amount of time it would take to train the SVM on the un-mapped data.
- Since we are still doing a linear separation, but in a different space, all the considerations of the previous sections hold.
- Now, how can we use this machine? After all, we need $\mathbf{w}$, and that will live in H also.
- However, in test phase an SVM is used by computing dot products of a given test point $\mathbf{x}$ with $\mathbf{w}$, or more specifically by computing the sign of

$$f(\mathbf{x}) = \sum_{i=1}^{N_S} \alpha_i y_i \left(\Phi(\mathbf{s}_i) \cdot \Phi(\mathbf{x}) + b \right) = \sum_{i=1}^{N_S} \alpha_i y_i \left(K(\mathbf{s}_i, \mathbf{x}) + b \right). \tag{21}$$

where $\mathbf{s}_i$ are the support vectors.

- Therefore, once again, one can avoid computing $\Phi(\mathbf{x})$ explicitly and use the $K(\mathbf{s}_i, \mathbf{x}) = \Phi(\mathbf{s}_i) \cdot \Phi(\mathbf{x})$ instead.
- To help identify kernels that exhibit the properties described above, one can utilize the following Mercer's condition (Vapnik 1995, Courant and Hilbert 1953): There exists a mapping Φ and an expansion

$$K(\mathbf{x}, \mathbf{y}) = \sum_i \Phi(\mathbf{x})_i \, \Phi(\mathbf{y})_i \tag{22}$$

if and only if, for any $g(\mathbf{x})$ such that

$$\int g(\mathbf{x})^2 dx \text{ is finite,} \quad (23)$$

the following expression holds:

$$\int K(\mathbf{x},\mathbf{y})g(\mathbf{x})g(\mathbf{y})d\mathbf{x}d\mathbf{y} \geq 0\,. \quad (24)$$

- Note that for specific cases, it may not be easy to check whether Mercer's condition is satisfied.
- In addition, Mercer's condition tells us whether a prospective kernel is actually a dot product in some space, but it does not tell us how to construct Φ or even what H is.
- Some of the popular kernels first investigated for pattern recognition were the following (Burges 1998):

$$K(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y}+1)^p \quad (25)$$

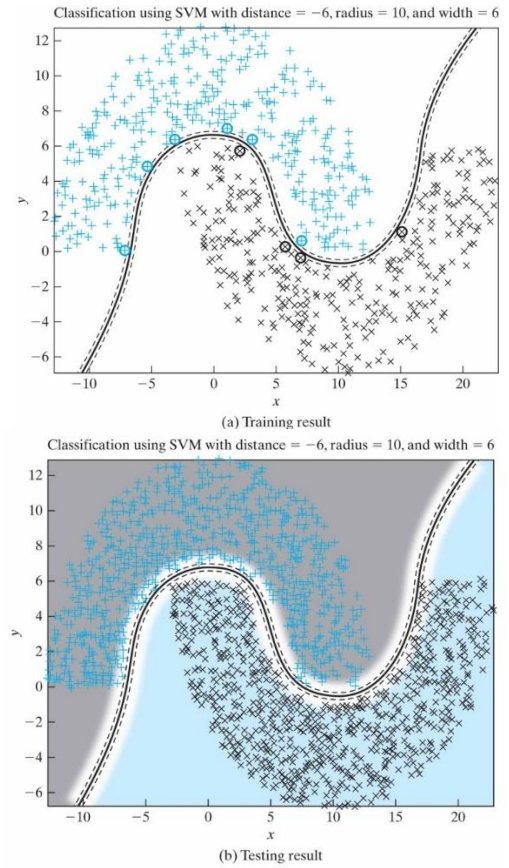
$$K(\mathbf{x},\mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|^2/2\sigma^2} \quad (26)$$

$$K(\mathbf{x},\mathbf{y}) = \tanh(k\mathbf{x}\cdot\mathbf{y}-\delta)\,. \quad (27)$$

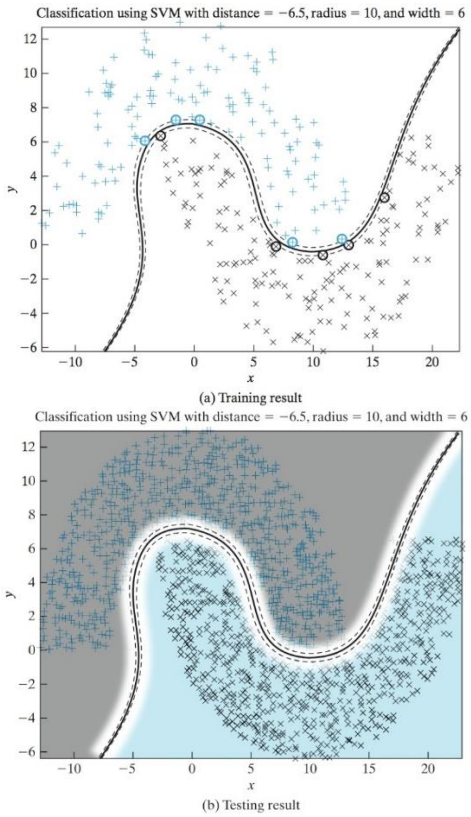
- Equation (25) results in a classifier that is a polynomial of degree p in the data.
- Equation (26) gives a Gaussian radial basis function classifier.
- Equation (27) gives a particular kind of two-layer perceptron (sigmoidal) neural network.
- For the RBF case, the number of centers (i.e., N_S in equation (21)), the centers themselves (the $\mathbf{s}_i$), the weights (α_i), and the threshold (b) are all produced automatically by the SVM training and give excellent results compared to classical RBFs, for the case of Gaussian RBFs (Schölkopf et. al 1997).
- For the neural network case, the first layer consists of N_S sets of weights, each set consisting of d_L (the dimension of the data) weights, and the second layer consists of N_S weights (the α_i), so that an evaluation simply requires taking a weighted sum of sigmoids, themselves evaluated on dot products of the test data with the support vectors.
- Thus for the neural network case, the architecture (number of weights) is determined by SVM training.
- Note, however, that the hyperbolic tangent kernel only satisfies Mercer's condition for certain values of the parameters k and δ (and of the data $\|\mathbf{x}\|^2$).
- Finally, note that although the SVM classifiers described above are binary classifiers, they are easily combined to handle the multiclass case.
- A simple, effective combination trains N one-versus-rest classifiers (say, "one" positive, "rest" negative) for the N-class case and takes the class for a test point to be that corresponding to the largest positive distance (Boser, Guyon, and Vapnik 1992).

COMPUTER EXPERIMENTS – PATTERN RECOGNITION

Synthetic Half-Moon Datasets (Source: Haykin, 2009)



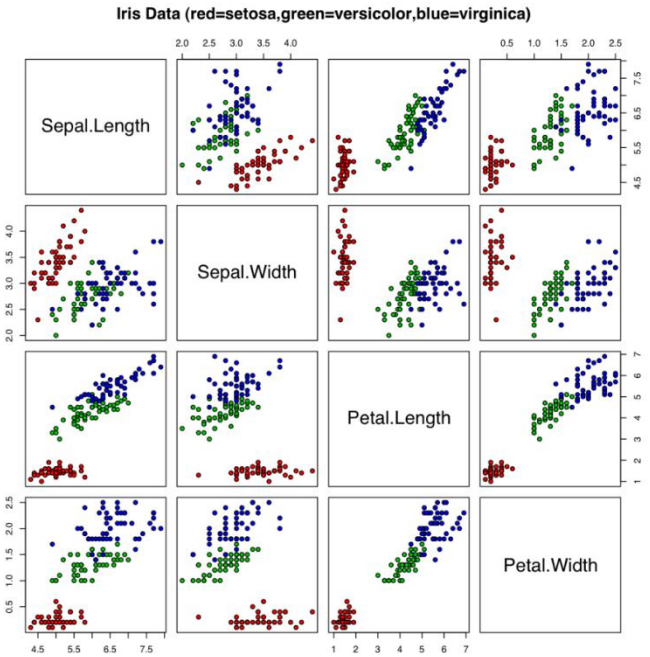
Experiment on SVM for the double-moon with distance d = –6.

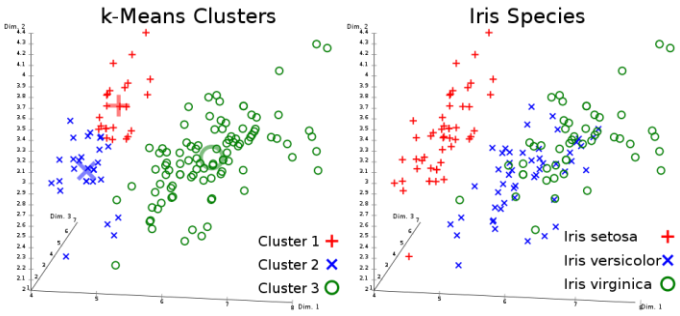


Experiment on SVM for the double-moon with distance d = –6.5.

IRIS Data Set – Classification (Source: http://en.wikipedia.org/wiki/Iris_flower_data_set)

- The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by Sir Ronald Fisher (1936) as an example of discriminant analysis.[3]
- Dataset consists of ***50 samples from each*** of ***three species*** of Iris (Iris setosa, Iris virginica and Iris versicolor).
- ***Four features*** were measured from each sample: the length and the width of the sepals and petals, in centimetres.





SVM implementation results to be demonstrated in class!

[3] R. A. Fisher (1936). "The use of multiple measurements in taxonomic problems". *Annals of Eugenics* **7** (2): 179–188. doi:10.1111/j.1469-1809.1936.tb02137.x.

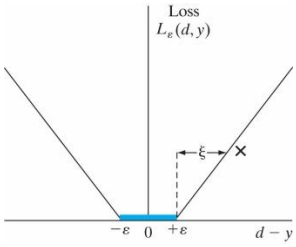
SUPPORT VECTOR MACHINES FOR NONLINEAR REGRESSION

ε- Insensitive Loss Function

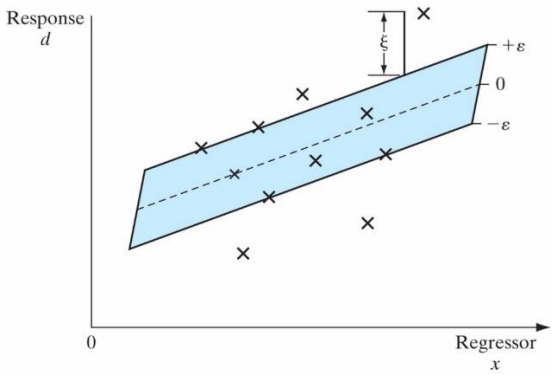
- For the sake of computational convenience, a loss function that is commonly used for optimizing learning machines is the quadratic loss function.
- However, a least-squares estimator is sensitive to the presence of outliers, and performs poorly when the underlying distribution of the additive noise has a long tail (Haykin, 1999).
- To overcome these limitations, in constructing a support vector machine for approximating a desired response d, a loss function that is often used is the following ε-insensitive loss function, originally proposed by Vapnik (1995):

$$L_\varepsilon(d,y) = \begin{cases} |d-y| - \varepsilon & \text{for } |d-y| \geq \varepsilon \\ 0 & \text{otherwise} \end{cases} \quad (28)$$

where ε is a prescribed parameter.



ε-insensitive loss function.



Illustrating the use of ε-insensitive loss function for linear regression.

SVM Formulation for Nonlinear Regression

- Consider a nonlinear regressive model in which the dependence of a scalar d on a vector $\mathbf{x}$ is described by

$$d = f(\mathbf{x}) + \upsilon \quad (29)$$

where the scalar-valued nonlinear function $f(\mathbf{x})$ is defined by the conditional expectation $E[D \mid \mathbf{x}]$, D is a random variable with a realization denoted by d, and υ is an additive noise term that is statistically independent of the input vector $\mathbf{x}$.

- One can postulate an estimate of d, denoted by y, as an expansion in terms of a set of nonlinear basis functions $\{\varphi(\mathbf{x})\}_{j=0}^{m}$ as follows:

$$y = \sum_{j=0}^{m} w_j \varphi_j(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}). \quad (30)$$

- It is assumed that $\varphi_0(\mathbf{x}) = 1$, so that the weight w_0 represents the bias b.

- For reasons outlined in Vapnik (1998) and Haykin (1999), the issue to be resolved is to minimize the empirical risk

$$R_{emp} = \frac{1}{N}\sum_{i=1}^{N} L_{\varepsilon}(d_i, y_i) \quad (31)$$

subject to the inequality

$$\|\mathbf{w}\|^2 \le c_0 \quad (32)$$

where c_0 is a constant.

- One can reformulate this constrained optimization problem by introducing two sets of nonnegative *slack variables* $\{\xi_i\}_{i=1}^{N}$ and $\{\xi_i'\}_{i=1}^{N}$ that are defined as follows:

$$d_i - \mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) \le \varepsilon + \xi_i, \quad i = 1,2,\ldots,N \quad (33)$$

$$\mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) - d_i \le \varepsilon + \xi_i', \quad i = 1,2,\ldots,N \quad (34)$$

$$\xi_i \ge 0, \quad i = 1,2,\ldots,N \quad (35)$$

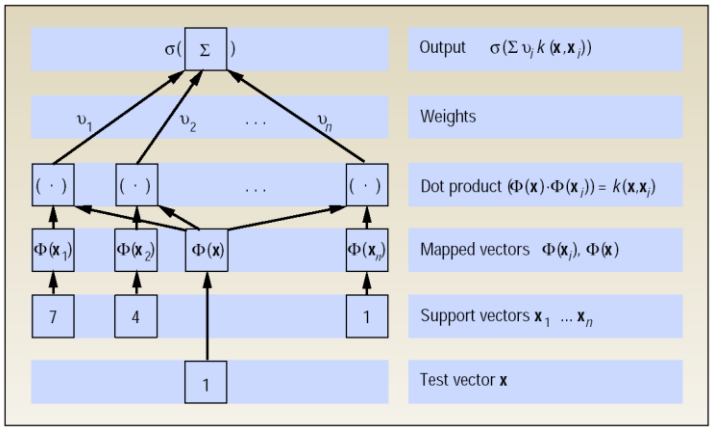
$$\xi_i' \ge 0, \quad i = 1,2,\ldots,N \quad (36)$$

- The slack variables describe the ε-insensitive loss function defined in equation (28).
- This constrained optimization problem may therefore be viewed as equivalent to that of minimizing the cost functional

$$\Phi(\mathbf{w},\xi,\xi') = C\left(\sum_{i=1}^{N}(\xi_i + \xi_i')\right) + \frac{1}{2}\mathbf{w}^T\mathbf{w} \quad (37)$$

subject to the constraints of equations (33) to (36).

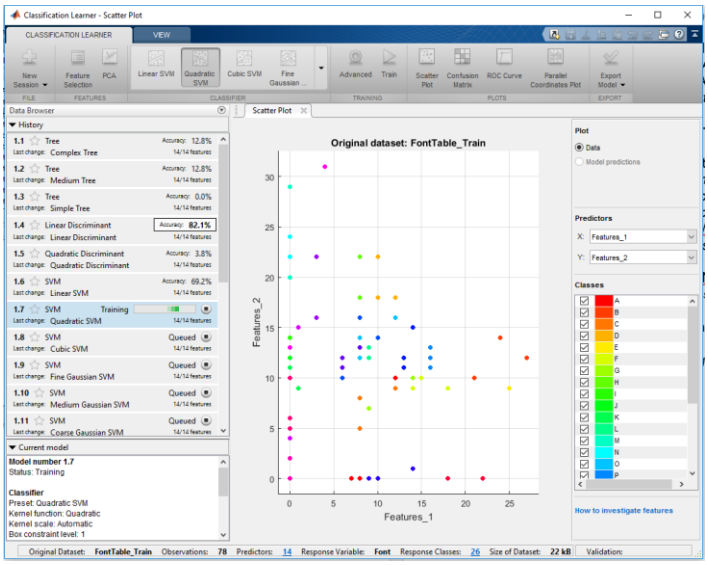
- By incorporating the term $\mathbf{w}^T\mathbf{w}/2$ in the functional $\Phi(\mathbf{w},\xi,\xi')$ of equation (37), we dispense with the need for the inequality constraint of equation (32).
- The two parameters ε and C control the VC dimension of the approximating function and must be tuned simultaneously.



Architecture of SV methods. (Source: Schölkopf, 1998)

IMPLEMENTING SVMs IN MATLAB

- Matlab offers two options for implementing SVMs for binary or multiclass classification
- For greater accuracy and kernel-function choices on low- through medium-dimensional data sets, train a binary SVM model or a multiclass error-correcting output codes (ECOC) model containing SVM binary learners using the Classification Learner App.



Classification Learner App

- For greater flexibility, use the command-line interface to train a binary SVM model using fitcsvm or train a multiclass ECOC model composed of binary SVM learners using fitcecoc.
- For reduced computation time on high-dimensional data sets that fit in the MATLAB® Workspace, efficiently train a binary, linear classification model, such as a linear SVM model, using "fitclinear" or train a multiclass ECOC model composed of SVM models using "fitcecoc".
- Classification Learner App also allows feature selection and PCA analysis (see the menu bar).
- **Note**: In the non-separable case (often called Soft-Margin SVM), one allows misclassifications, at the cost of a penalty factor *C*. Making *C* large increases the weight of misclassifications, which leads to a stricter separation. Matlab refers to this factor *C* as "Box Constraint" (available under "Advanced" tab. The reason for this name is, that in the formulation of the dual optimization problem, the Langrange multipliers are bounded to be within the range [0,*C*]. *C* thus poses a box constraint on the Lagrange multipliers.

References:

Aizerman, M.A., Braverman, E.M., and Rozoner, L.I. (1964) Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25, 821-837.

Boser, B.E., Guyon, I.M., and Vapnik, V.N. (1992) A training algorithm for optimal margin classifiers. In *Proc. Fifth Ann. Workshop Computational Learning Theory*, ACM Press, New York, 144–152.

Burges, C.J.C. (1998) A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2, 121-167.

Cortes, C. and Vapnik, V. (1995) Support vector networks. *Machine Learning*, 20, 273-297.

Courant, R. and Hilbert, D. (1953) .*Methods of Mathematical Physics*, Interscience.

Fletcher, R. (1987) *Practical Methods of Optimization*, John Wiley and Sons, 2nd edition, New York.Vapnik, V. (1979) *Estimation of Dependencies Based on Empirical Data* [in Russian], Nauka, Moscow. (English translation: Springer-Verlag, New York, 1982).

Haykin, S. (1999) *Neural Networks: A Comprehensive Foundation*, 2nd edition, Prentice Hall, NJ: Upper Saddle River.

Schölkopf, B., Sung, K., Burges, C., Girosi, F., Niyogi, P., Poggio, T., and Vapnik, V. (1997) Comparing support vector machines with Gaussian kernels to radial basis function classifiers. *IEEE Trans. Sign. Processing*, 45, 2758-2765.

Schölkopf, B. (1998) SVMs—A practical consequence of learning theory. *IEEE Intelligent Systems*, 13(3), 18–21.
Vapnik, V. (1995) *The Nature of Statistical Learning Theory*, Springer-Verlag, New York.
Vapnik, V. (1998) *Statistical Learning Theory*, John Wiley and Sons, New York.