Linear Discriminant Functions

Computing probability density, even in the case when we know the form of the distribution, can be difficult. Worse still, in most cases it is doubtful if the true distribution agrees with the form assumed for designed classifier.

Instead of assuming a form of probability distribution, we can assume that form of the *discriminant function* is known. The training set will be used to determine parameters of this function. We expect that our classifier will be much simpler at the cost of insignificant (we hope so at least) decrease of classification quality.

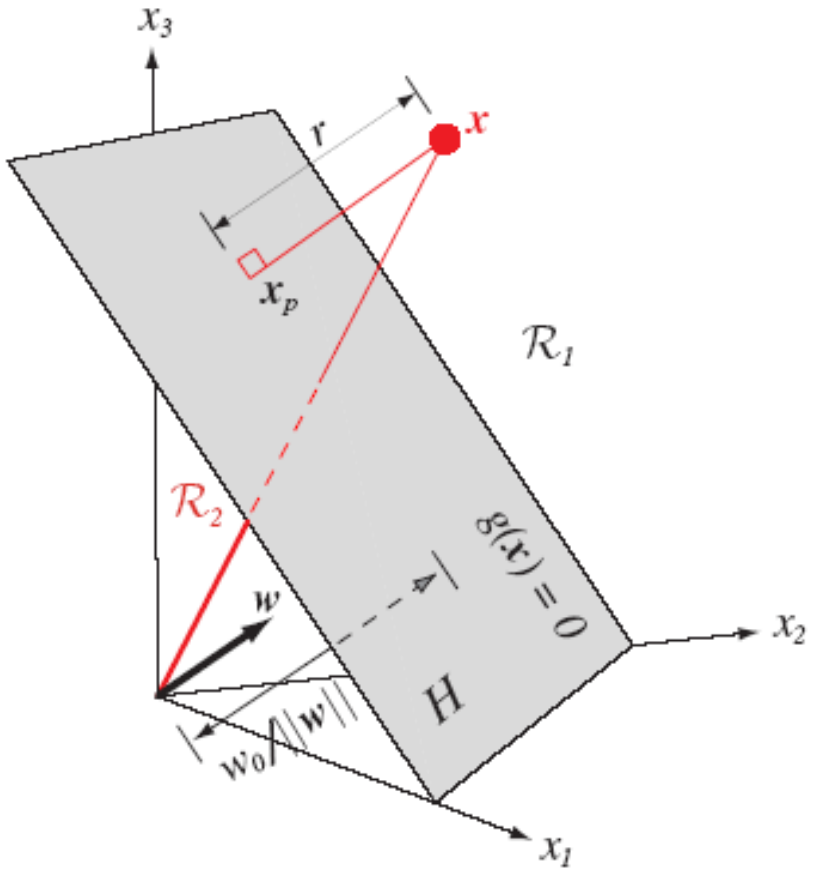
The simplest form of discriminant function is a linear one.

When used directly in the feature space, decision boundary for linear discriminant function is a hyperplane.

One can use also functions, which are not linear in the feature space, but are linear in some function set over this space.

Linear Discriminant Functions

Finding the discriminant function can be formulated as the problem of minimizing the value of a criterion (or error) function.
It is natural to use mean classifier error on the training set as the criterion function.
We must however remember that classifier is designed not for a training set, but for new, previously unseen samples.

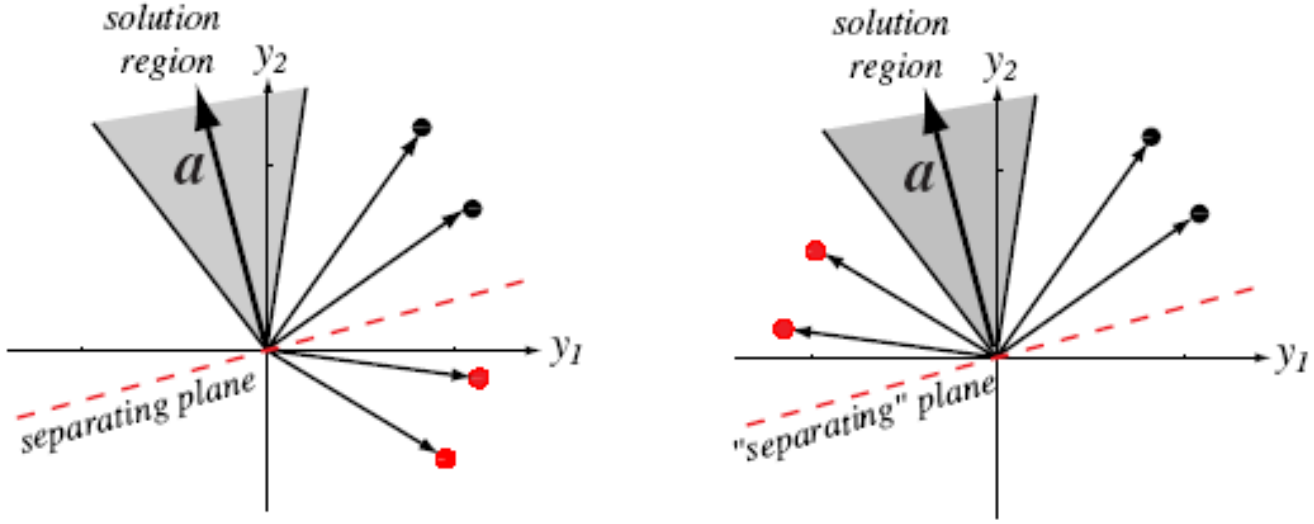


Linear Discriminant Functions

The linear discriminant function can be written as:

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$$

where $\mathbf{w}$ is the weight vector and w_0 is bias (or threshold weight).
We can simplify notation by introducing augmented (homogenous) coordinates.
Augmented feature vector: $\mathbf{y}^T = [1\ x_1 \ldots x_d] = [1\ \mathbf{x}^T]$



Linear Discriminant Functions

Augmented feature vector: $\mathbf{a}^T = [w_0\ w_1\ \ldots\ w_d] = [w_0\ \mathbf{w}^T]$

And the discriminant function: $g(\mathbf{x}) = \sum_{i=0}^{d} w_i x_i = \mathbf{a}^T \mathbf{y}$

The new ($d+1$)-dimensional space will be called weight space. In general case we will have c discriminant functions - one for each class. We decide that unknown object belongs to class c_i, if $g_i(x) > g_j(x), i \neq j$.

To distinct two classes one discriminant function $g(x)$ will do, with the classification rule: object belongs to class c_1 if $g(x) > 0$ and to class c_2 if $g(x) < 0$.

Our task is to find values of weight vector $\mathbf{a}$ minimizing criterion function.

Linearly Separable Case

If exists such $\mathbf{a}$ for which all training samples are classified correctly, the samples are said to be *linearly separable*.

For all $\mathbf{y}_i$ belonging to class c_1 we have $\mathbf{a}^T\mathbf{y}_i > 0$, and for all $\mathbf{y}_j$ belonging to class c_2 we have $\mathbf{a}^T\mathbf{y}_j < 0$.

We can "normalize" the problem replacing weight vectors of class ω_2 with their “reflections” ($\mathbf{y}_j^T = [-1 - x_1 \ldots - x_d]$). With this normalization **all elements** of the training set satisfy inequality: $\mathbf{a}^T\mathbf{y}_i > 0$.

As a rule there is infinite number of possible solutions - they define a *solution region* in the weight space.

Linearly Separable Case

We can impose additional requirements on the solution searched, in hope that better classification will be gained:

1. Seek for unit-length vector that maximizes the minimum distance from the samples to the separating plane.

2. Seek for minimum-length vector satisfying $\mathbf{a}^T\mathbf{y}_i \geq b$ for all i, where b is a positive constant called the margin.

The motivation behind such attempts is to find solution vector closer to the "middle" of the solution region, with belief that such a vector will better classify new (unseen) samples.

Gradient Methods

We define criterion function $J(\mathbf{a})$ that reaches minimum value for the $\mathbf{a}$ solution vector. We start with some arbitrarily selected weight vector $\mathbf{a}_{start}$. In each step we compute the gradient vector $\nabla J(\mathbf{a})$ and move the solution $\mathbf{a}$ in direction of a steepest descent of $J(\mathbf{a})$:

$$\mathbf{a}(k+1) = \mathbf{a}(k) - \eta(k)\nabla J(\mathbf{a}(k))$$

where $\eta(k)$ is positive scale factor or learning rate.
The termination condition for the whole process is decreasing of the solution $\mathbf{a}$ correction below some given threshold θ:

$$|\eta(k)\nabla J(\mathbf{a}(k))| < \theta .$$

Perceptron Criterion Function

Perceptron criterion function:

$$J_p(\mathbf{a}) = \sum_{\mathbf{y} \in Y} -\left(\mathbf{a}^T \mathbf{y}\right)$$

where Y is set of samples misclassified with current value of $\mathbf{a}$.

$J_p(\mathbf{a})$ is proportional to the sum of distances between misclassified samples and decision boundary.

Gradient of perceptron function: $\nabla J_p = \sum_{\mathbf{y} \in Y} -(\mathbf{y})$

gives the $\mathbf{a}$ update rule: $\mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k) \sum_{\mathbf{y} \in Y_k} \mathbf{y}$

and the termination condition: $\left| \eta(k) \sum_{\mathbf{y} \in Y_k} \mathbf{y} \right| < \theta$

Perceptron Method

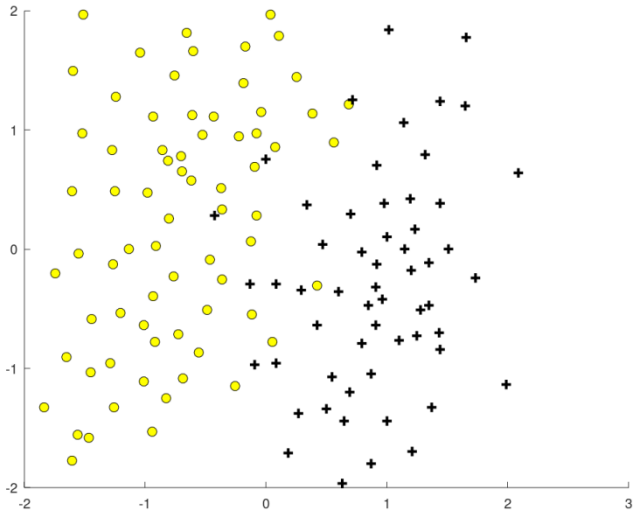
Generally, keeping learning rate constant is not favorable. In the last stages of learning, when $\mathbf{a}$ is in the proximity of solution (i.e. properly classifies nearly all the samples), would be wise to delicately decrease learning rate.

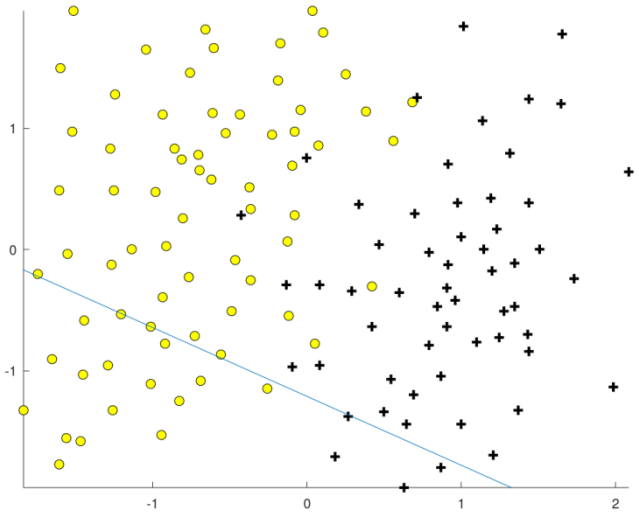
As a rule of the thumb the value of learning rate $\eta(k)$ is decreased proportionally to $1/k$.

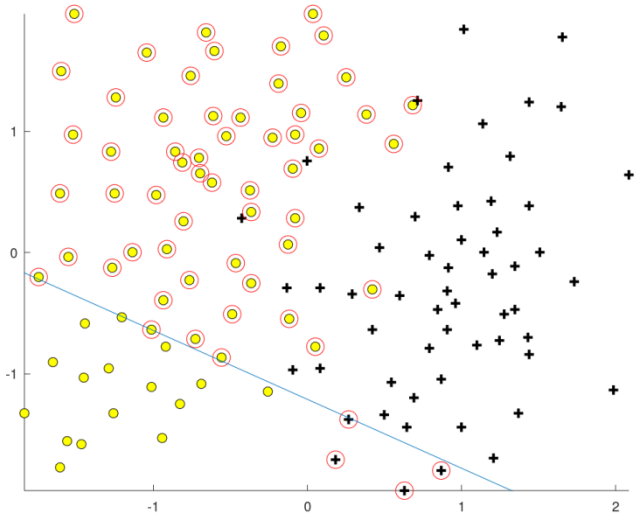
Sometimes learning rate is modified in both directions: when error decreases learning rate is increase (because we move in „good" direction), and when error increases learning rate is decreased.

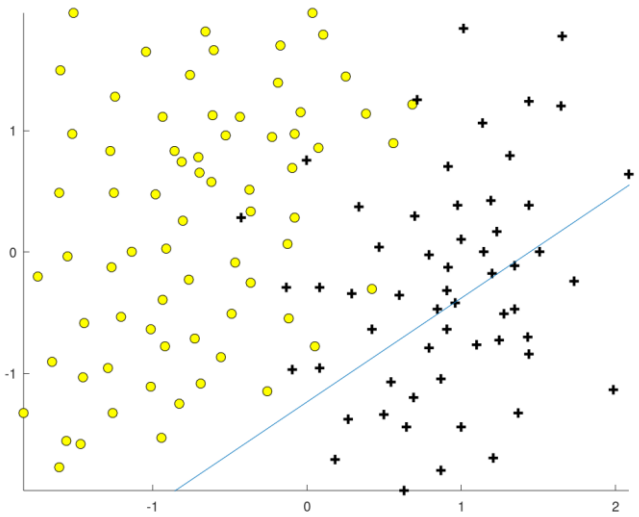
We can also attempt to compute optimal value of $\eta(k)$ for current solution, but it is computationally expensive (inverse of hessian matrix ☺ is needed).

Perceptron method in work

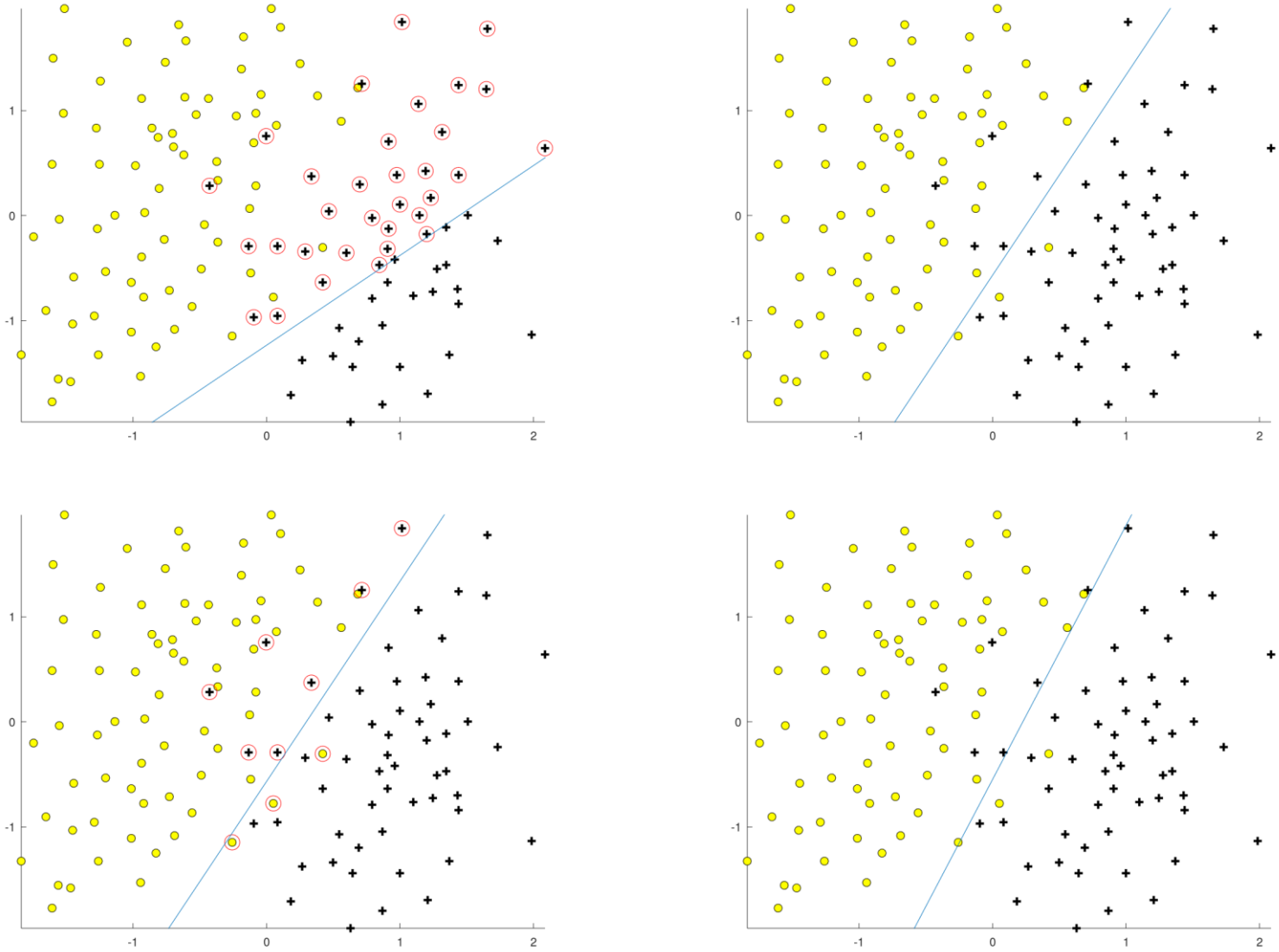




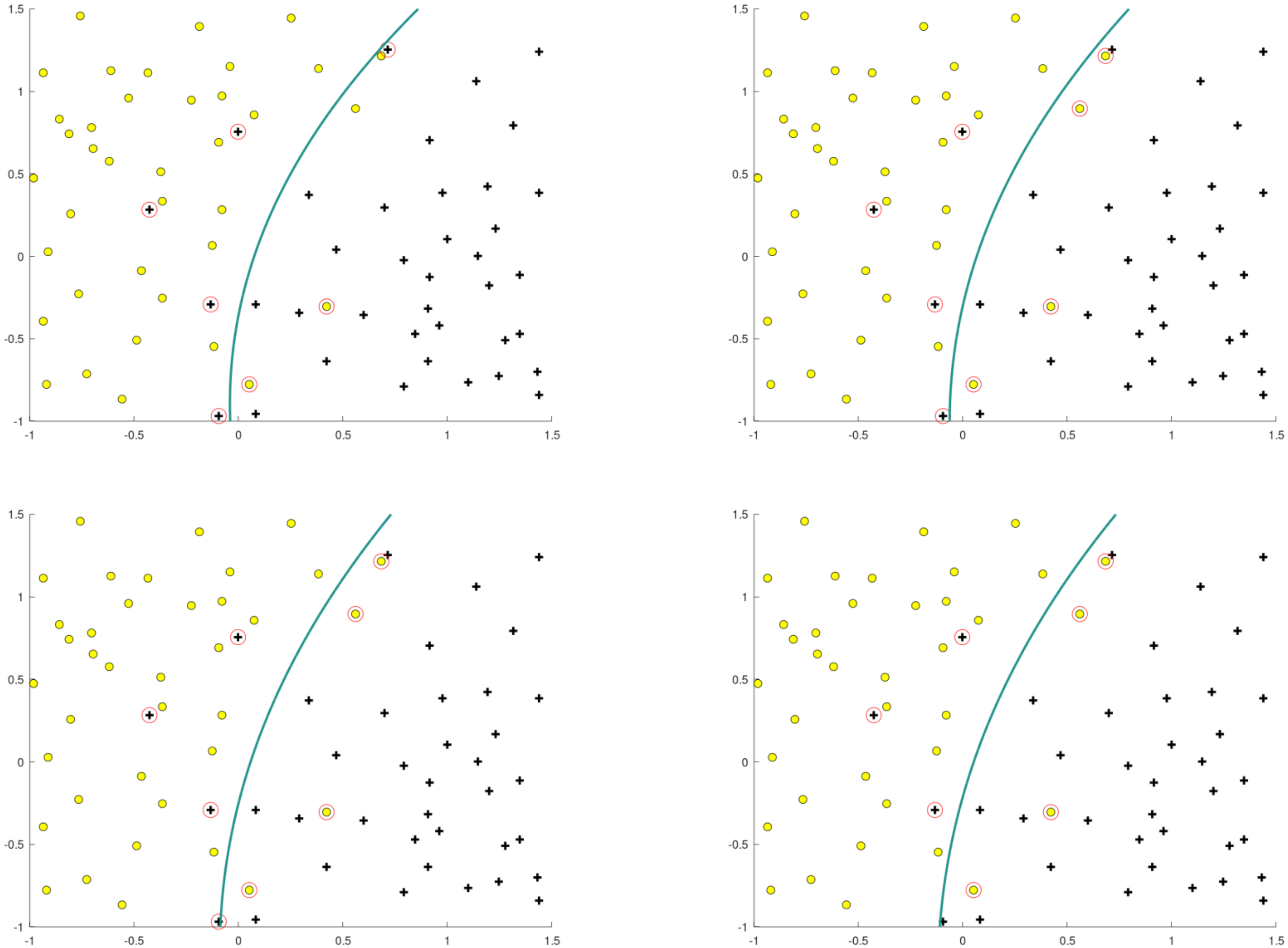




Perceptron method in work – ctnd.



Perceptron method in work – additional features

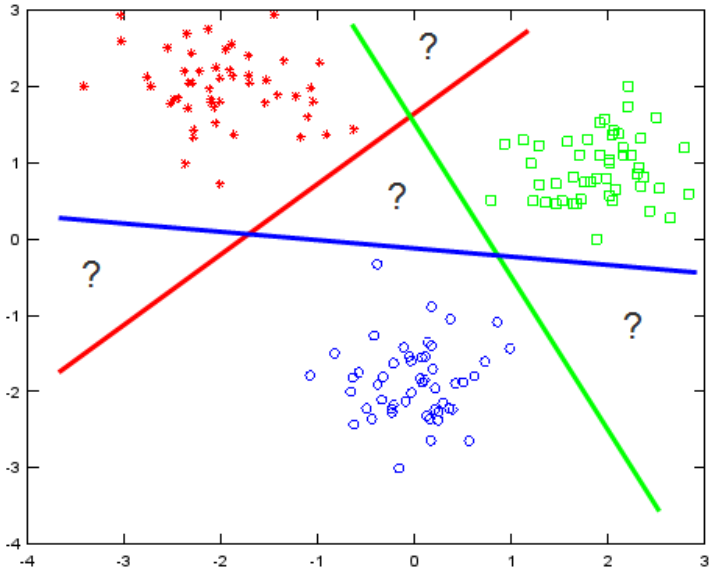


More than two classes: *one versus rest*

Difficulties arise when we want to use the linear classifier in the problem of $c > 2$ class classification.
One of the possible approaches is to prepare c linear classifiers, each of which distinguishes samples of one class from samples of all other classes. This approach is called *one vs. rest* or *one vs. all*.
An unquestionable advantage is that during training of each classifier the entire training set is used. The problem is that in a few cases one class can be "separated" from the rest by a hyperplane (with a decent error of course). What's more we have unbalanced sizes of positive and negative samples.

More than two classes: *one versus rest*

If we directly apply a decision rule for two classes, (in the version: the object is of a particular class - the object is not of that particular class), then "problematic regions" appear. The classifier ensemble acts as an 1 of n selector.



You can also change this basic rule by including a larger number of classifiers.
For example: we will consider the classified object *x* as belonging to the class c_i if $g_i(x) > g_i(x), i \neq j$.
Such an approach requires appropriate normalization of decision functions, so that their values can be directly compared.

More than two classes: *one versus one*

Difficulties in separating one class from all the rest of the training set lead to another idea for classifying c classes.
We prepare classifier for each pair of classes: $g_{ij}(x), i \neq j$
In this case, it is usually easier to get a decent quality of individual classifiers. There are more classifiers however: $\frac{c(c-1)}{2}$. Each of these classifiers "cast a vote" for a particular class.

As a result of the classification, we take the class with the maximum number of votes.

The problem is that each of these elementary classifiers is trained only on a part of the training set (just two classes out of c classes), but it casts its vote for each classified sample.

Including these "incompetent" votes causes degradation of the quality of the classifier ensemble.

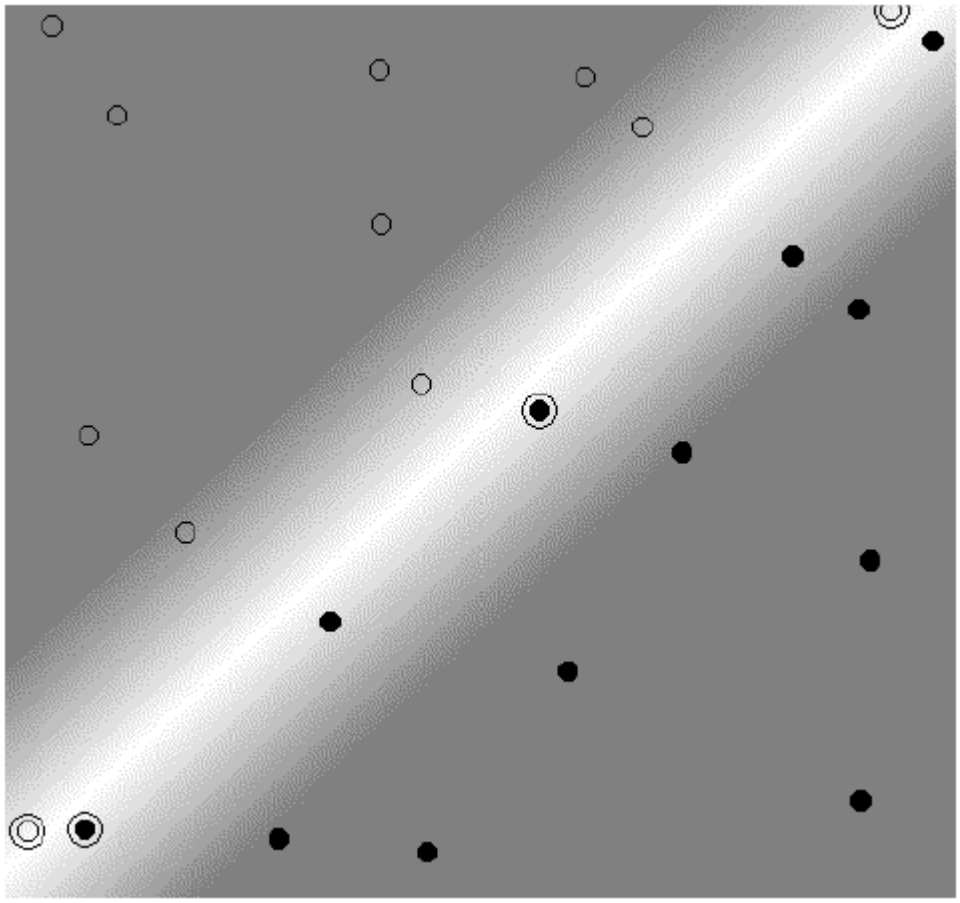
Support Vector Method

(*Support Vector Machines*)

Vladimir Vapnik, 1995
"The nature of Statistical Learning Theory"
Christopher Burges, 1998
"A Tutorial on Support Vector Machines for Pattern Recognition"

Support vector classification is an extension (conceptually at least) of the idea of linear discriminant functions.
In the linearly separable case we demand such a separating plane, that all training vectors were not closer to the plane than some distance d/2.
It turns out, that such a separating plane is determined by a few vectors - **support vectors**.

Linearly Separable Case



Linearly Separable Case

Training data: $\{\mathbf{x}_i, y_i\}, i = 1, \cdots, l,\ y_i \in \{-1, 1\},\ \mathbf{x}_i \in \mathbf{R}^d$

Points on the separating hyperplane satisfy the equation: $\mathbf{w} \cdot \mathbf{x} + b = 0$

d_+ (d_-)is the shortest distance from the separating hyperplane to the closest '+' ('-') vector. Margin of the separating hyperplane is given by $d_+ + d_-$

Let all the training data satisfy the following conditions:

$$\mathbf{x}_i \cdot \mathbf{w} + b \geq +1 \quad for\ y_i = +1$$

$$\mathbf{x}_i \cdot \mathbf{w} + b \leq -1 \quad for\ y_i = -1$$

These can be combined into one constraint set:

$$y_i\left(\mathbf{x}_i \cdot \mathbf{w} + b\right) - 1 \geq 0 \quad \forall i \qquad (1)$$

Margin width is $\dfrac{2}{\|\mathbf{w}\|}$

Linearly Separable Case

We can find a pair of hyperplanes which gives the maximum margin by minimizing $\|\mathbf{w}\|^2$ subject to constraints (1)

The solution uses Lagrangian for two reasons:

- Constraints (1) are replaced by simpler constraints on Lagrange multipliers
- Training data will only appear in the form of dot products between vectors.

We'll use non-negative Lagrange multipliers $\alpha_i, i = 1, \cdots, l$

Since constraints are in the form $c_i \geq 0$ we subtract multipliers (times constraints) from the objective function to form the Lagrangian:

$$L_P \equiv \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{l} \alpha_i y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b\right) + \sum_{i=1}^{l} \alpha_i$$

Linearly Separable Case

We must:

1. Minimize L_P with respect to $\mathbf{w}$ and b
2. With derviatives of L_P with respect to all α_i equal 0
3. All subject to constraints $\alpha_i \geq 0$

We can formulate "dual" problem (Wolfe dual):

1. Maximize L_P with respect to $\mathbf{w}$ and b
2. With gradient of L_P with respect to $\mathbf{w}$ and b equal 0
3. All subject to constraints $\alpha_i \geq 0$

Linearly Separable Case

Gradient conditions $\frac{\partial L_P}{\partial \mathbf{w}} = 0$ and $\frac{\partial L_P}{\partial b} = 0$ give:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

$$\sum_i \alpha_i y_i = 0$$

Dual formulation of the problem:

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_i \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

Karush-Kuhn-Tucker (KKT) Conditions

KKT conditions play main role in theory **and practice** of constrained optimization:

$$\frac{\partial}{\partial w_v} L_P = w_v - \sum_i \alpha_i y_i x_{iv} = 0 \quad v = 1, \cdots, d$$

$$\frac{\partial}{\partial b} L_P = -\sum_i \alpha_i y_i = 0$$

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

$$\alpha_i \geq 0 \quad \forall i$$

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

Fulfilling KKT conditions is necessary and sufficient to solve the problem (i.e. to find support vectors, or – in other words – to compute α_i coefficients).

SVM Classification

Let's assume that all α_i values are already computed. Of course values $\alpha_i > 0$ are only for support vectors. From KKT conditions we can compute **w**: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$. The last remaining element is the value of *b*. It can be determined from the last KKT condition: we can take any non-zero α_i to calculate *b* (numerically safer solution is to compute *b* as the mean for all $\alpha_i \neq 0$).

Now we can compute decision function:

$$f(\mathbf{x}_c) = \mathbf{x}_c \cdot \mathbf{w} + b = \sum_{i \in N_S} \alpha_i y_i \mathbf{x}_c \cdot \mathbf{x}_i + b$$

Classification decision depends on the sign of $f(\mathbf{x}_c)$ value.

Non-Separable Case

We must relax constraints by introducing positive slack variables $\xi_i, i = 1, \cdots, l$:

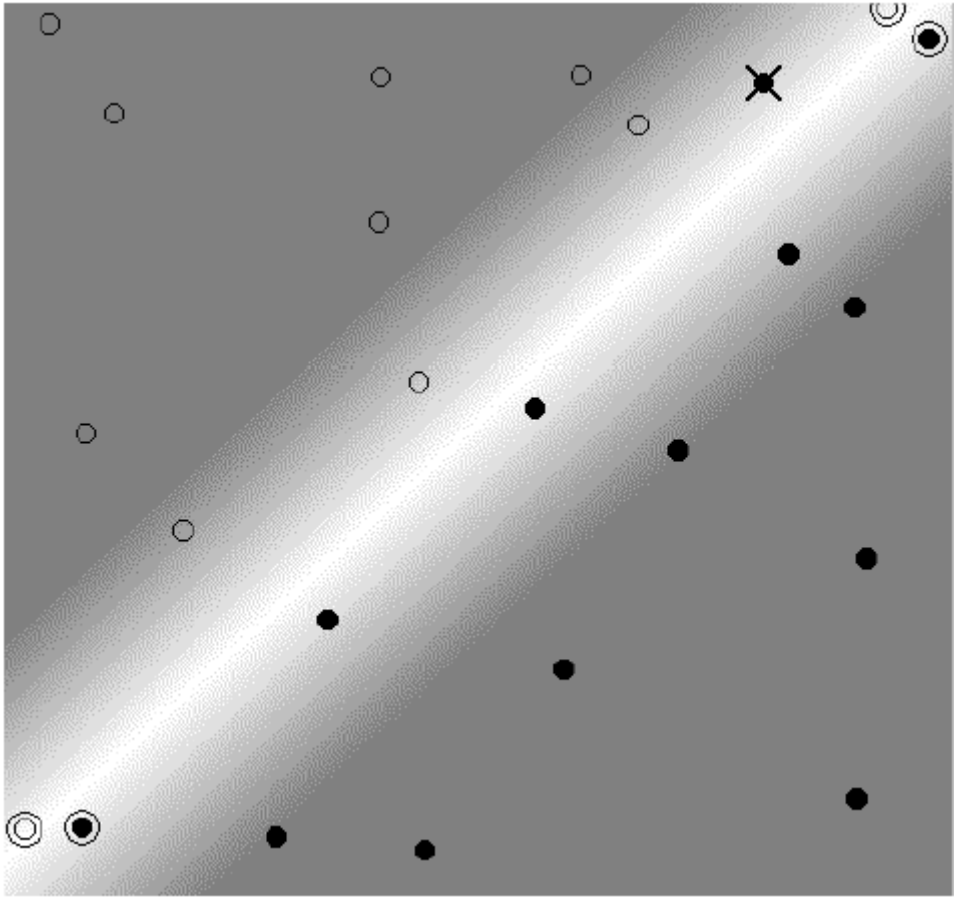
$$\mathbf{x}_i \cdot \mathbf{w} + b \geq +1 - \xi_i \quad for\ y_i = +1$$

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

$$\xi_i \geq 0 \quad \forall i$$

Instead of $\frac{\|\mathbf{w}\|^2}{2}$ we minimize: $\frac{\|\mathbf{w}\|^2}{2} + C\left(\sum_i \xi_i\right)$, where C is user selected constant (error cost). The higher C value the more costly errors, which leads to smaller margin between classes.

Non-Separable Case



Non-Separable Case

The dual problem has the form of maximization of:

$$L_D \equiv \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

subject to constraints:

$$0 \le \alpha_i \le C$$

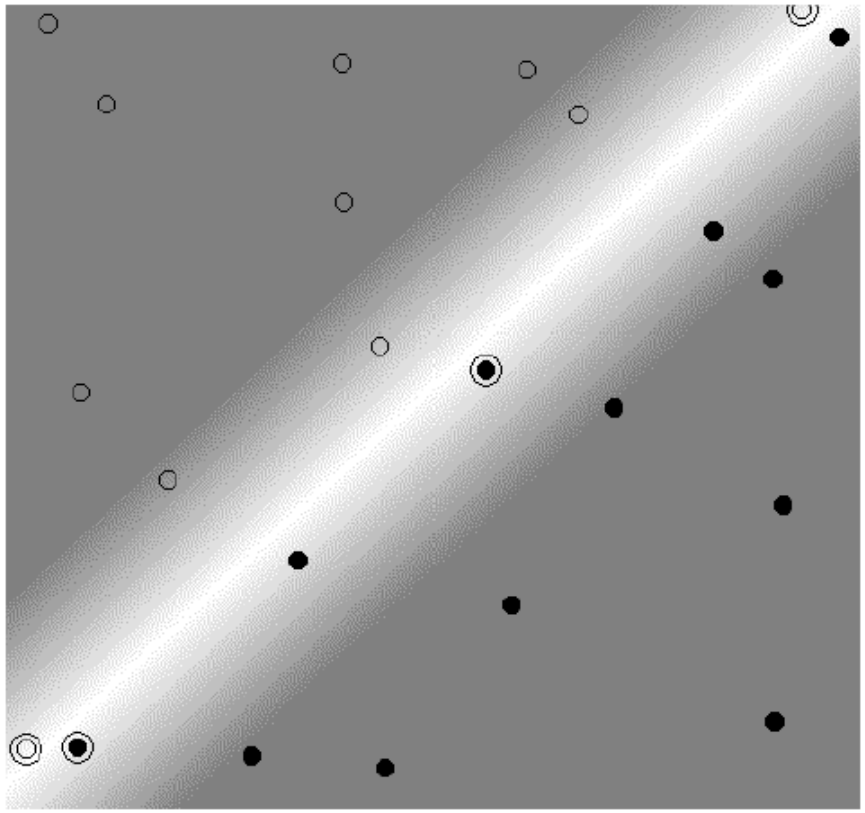
$$\sum_i \alpha_i y_i = 0$$

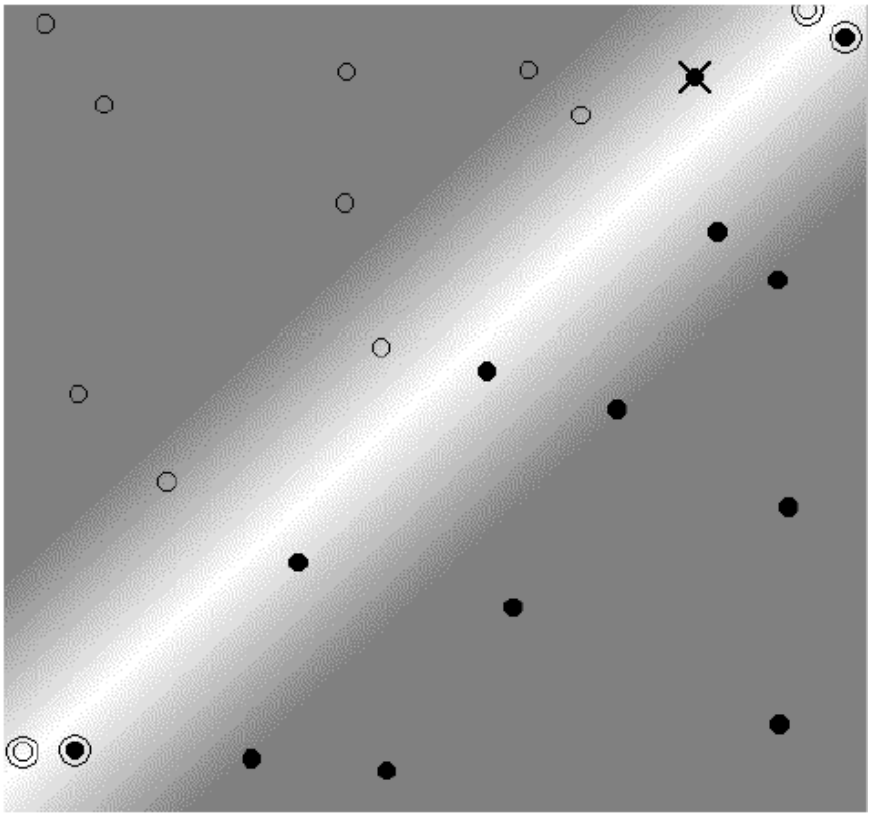
Solution is (again):

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

Note, that the only difference is upper boundary for Lagrange multipliers. (To complete data for implementation we should repeat here KKT conditions).

Non-Separable Case





Non-linear Case

Training data appear only in the form of dot product $\mathbf{x}_i \cdot \mathbf{x}_j$

Let's assume that exists mapping $\Phi : \mathbf{R}^d \mapsto \mathrm{H}$, where $\mathbf{H}$ is Euclidean space.

Training algorithm depends on data in the form $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$

Having kernel function K such that $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ we don't need to use Φ explicitly in training and testing.

How to compute $\mathbf{w}$? $\mathbf{w} = \sum_{i=1}^{N_s} \alpha_i y_i \Phi(\mathbf{s}_i)$, where $\mathbf{s}_i$ are support vectors.

In classification we need sign of the function:

$$f(\mathbf{x}) = \sum_{i=1}^{N_s} \alpha_i y_i \Phi(\mathbf{s}_i) \cdot \Phi(\mathbf{x}) + b = \sum_{i=1}^{N_s} \alpha_i y_i K(\mathbf{s}_i, \mathbf{x}) + b$$

Non-linear Case

What is the dimensionality of $\mathbf{H}$?

For polynomial kernel: $\binom{d+p-1}{p}$

Concrete numbers: image 16 by 16 pixels (d=256) and the polynomial of degree p=4 give dim(H)=183181376

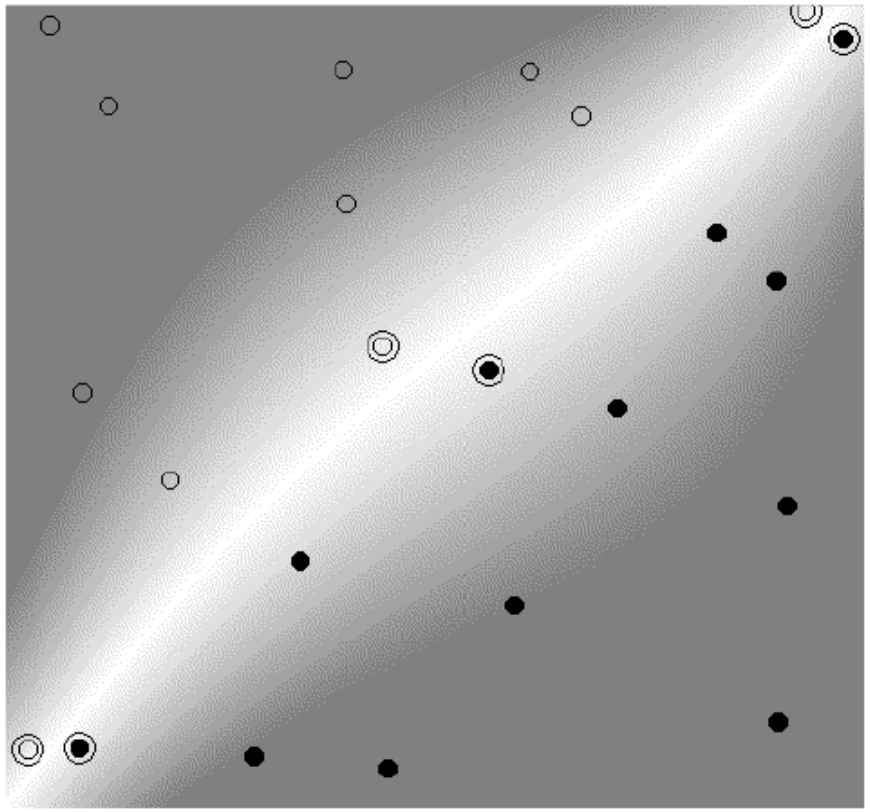
Kernel functions investigated for the pattern recognition problem:

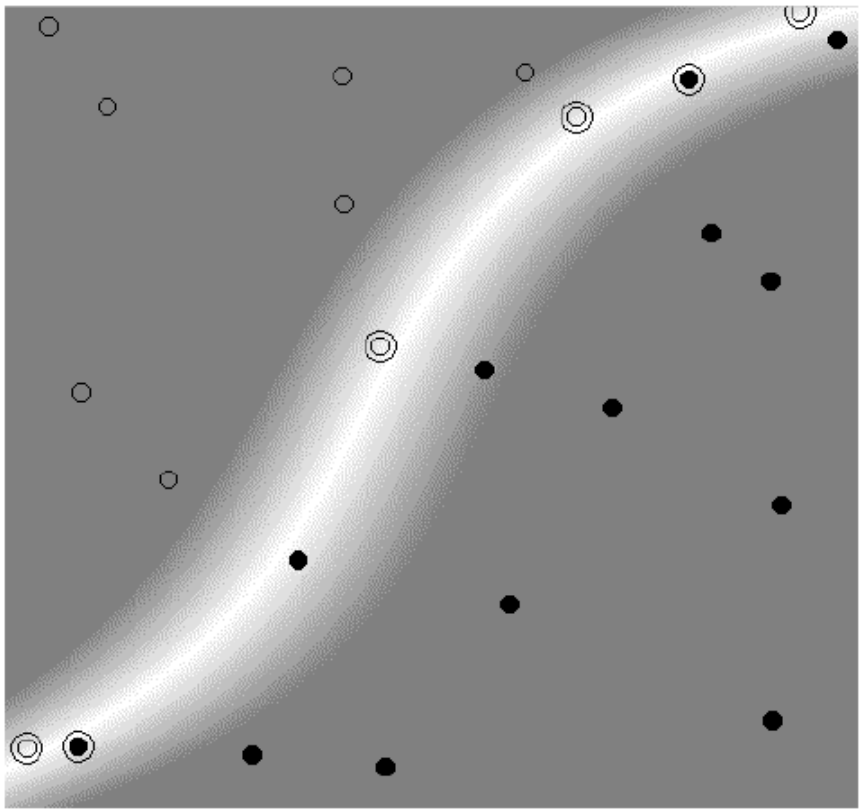
$$K(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y}+1)^p$$

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

$$K(\mathbf{x},\mathbf{y}) = \tanh(\kappa\mathbf{x}\cdot\mathbf{y}-\delta)$$

Non-linear Case





Sequential Minimal Optimization

J. C. Platt: *Fast Training of Support Vector Machines using Sequential Minimal Optimization*

SVM training requires solving of very large quadratic programming (QP) problem. SMO breaks this large problem into a series of smallest possible QP problems, which solutions are computed analytically.
Memory complexity of SMO is linear with the training set size.
Computational complexity of SMO lies between linear and quadratic with respect to the training set size.
(SMO's computations are dominated by the SVM evaluation, so we can expect SMO to work best on linear SVM and sparse data sets).

Sequential Minimal Optimization

Problem:

$$\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \alpha_i \alpha_j$$

$$0 \leq \alpha_i \leq C, \quad \forall i$$

$$\sum_{i=1}^{l} y_i \alpha_i = 0$$

KKT conditions:

$$\alpha_i = 0 \Rightarrow y_i f(\mathbf{x}_i) \geq 1, \quad \forall i$$

$$0 < \alpha_i < C \Rightarrow y_i f(\mathbf{x}_i) = 1, \quad \forall i$$

$$\alpha_i = C \Rightarrow y_i f(\mathbf{x}_i) \leq 1, \quad \forall i$$

SMO Algorithm

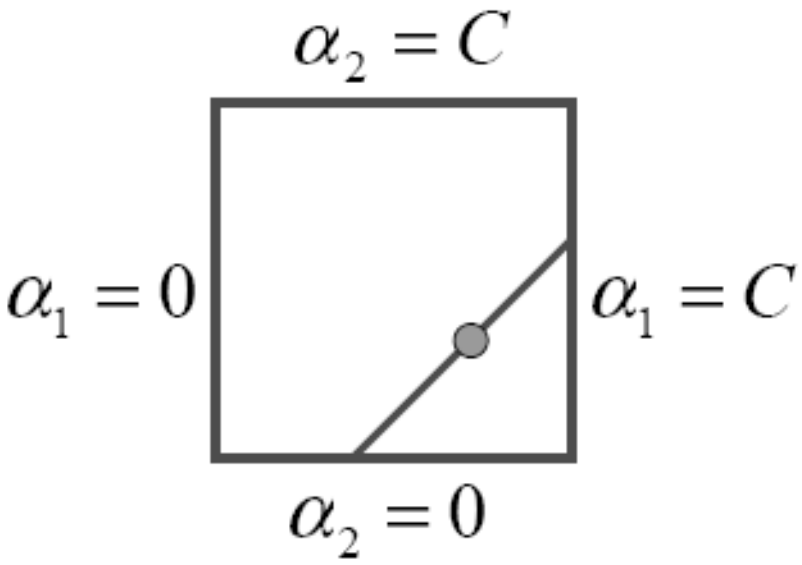
Because Lagrange multipliers must obey linear equation constraint, the smallest possible optimization problem involves **two** Lagrange multipliers.

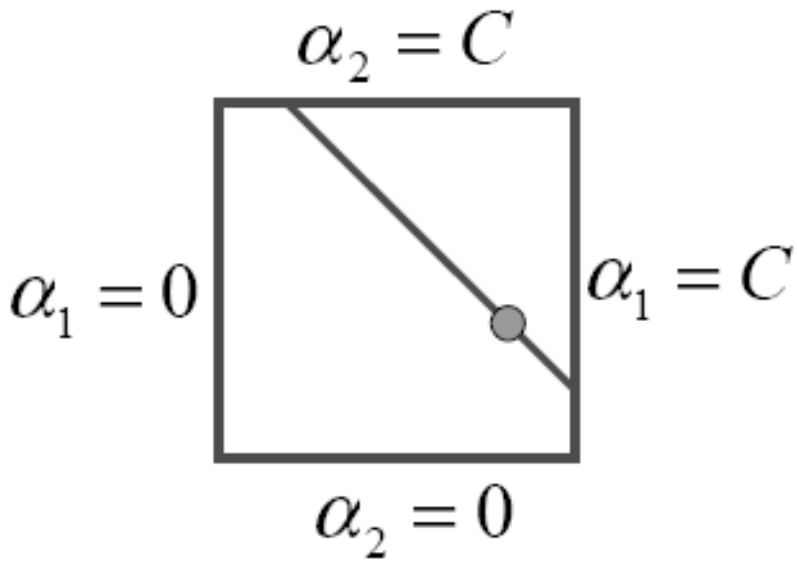
In every step SMO chooses two multipliers to jointly optimize, finds optimal values for these multipliers and updates SVM.

SMO has 3 basic elements:

1. An analytic method for solving for the two Lagrange multipliers.
2. A heuristic for choosing which multipliers to optimize.
3. A method of computing *b* value.

Solving for the two Lagrange multipliers





$$y_1 \neq y_2 \Rightarrow \alpha_1 - \alpha_2 = \gamma$$

$$y_1 = y_2 \Rightarrow \alpha_1 + \alpha_2 = \gamma$$

The inequality constraints cause the Lagrange multipliers to lie in the box. The linear equality constraint causes them to lie on a diagonal line. Therefore, one step of SMO must find an optimum of the objective function on a diagonal line segment.

Heuristic to select the first multiplier

If the sample does not fulfill KKT conditions ($y_i f(\mathbf{x}_i) ? 1$) it is taken to optimization. The second multiplier is selected with other heuristic.

After optimization SVM is updated with the two new multiplier values and algorithm returns to searching for multipliers to optimize.

To speed-up the process in the first pass over the whole training set the loop marks non-bound samples (i.e. those for which $0 < \alpha_i < C$).
Next outer loop iterates only over non-bound samples until all these samples obey the KKT conditions within ε.
Again outer loop iterates over the whole training set.
This switching (one pass over the whole set, many passes over non-bound samples) is repeated until all training set samples obey the KKT conditions within ε.

Heuristic to select the second multiplier

The second multiplier is selected so as to maximize the step taken during joint optimization. For non-bound samples SMO stores error values. The second multiplier is selected to maximize $|E_1 - E_2|$.

In unusual cases the above heuristic does not make a progress in the direction of desired solution (for example, two identical samples in the training set make objective function semi-definite).

In such a case:

A) SMO starts iterating non-bound samples to find the one that can make a positive progress

B) if A) doesn't make positive progress the second multiplier is searched in the whole training set

In both cases searching begins at random positions (to avoid bias to the samples at the beginning of the training set ☺).

Threshold and error cache

To compute decision function we need threshold value b. (In KKT b value was presented in somewhat complicated form).

SMO calculate b value iteratively.

Similarly error values (for non-bound examples) are calculated iteratively and stored in error cache.

Results

Experiment	Kernel	Sparse Code Used	Training Set Size	C	% Sparse	SMO Time (sec)	Chunking Time (sec)	SMO Scaling Exponent	PCG Scaling Exponent
Adult Linear Small	Linear	Y	11221	0.05	89%	17.0	20711.3	1.9	3.1
Adult Linear Large	Linear	Y	32562	0.05	89%	163.6	N/A	1.9	3.1
Web Linear	Linear	Y	49749	1	96%	268.3	17164.7	1.6	2.5
Lin. Sep. Sparse	Linear	Y	20000	100	90%	280.0	374.1	1.0	1.2
Lin. Sep. Dense	Linear	N	20000	100	0%	3293.9	397.0	1.1	1.2
Random Linear Sparse	Linear	Y	10000	0.1	90%	67.6	10353.3	1.8	3.2
Random Linear Dense	Linear	N	10000	0.1	0%	400.0	10597.7	1.7	3.2
Adult Gaussian Small	Gaussian	Y	11221	1	89%	781.4	11910.6	2.1	2.9
Adult Gaussian Large	Gaussian	Y	32562	1	89%	7749.6	N/A	2.1	2.9
Web Gaussian	Gaussian	Y	49749	5	96%	3863.5	23877.6	1.7	2.0
Random Gaussian Sparse	Gaussian	Y	5000	0.1	90%	986.5	13532.2	2.2	3.4
Random Gaussian Dense	Gaussian	N	5000	0.1	90%	3957.2	14418.2	2.3	3.1
MNIST	Polynomial	Y	60000	100	81%	29471.0	33109.0	N/A	N/A