

Support Vector Machine

By Waad ALMASRI

Introduction

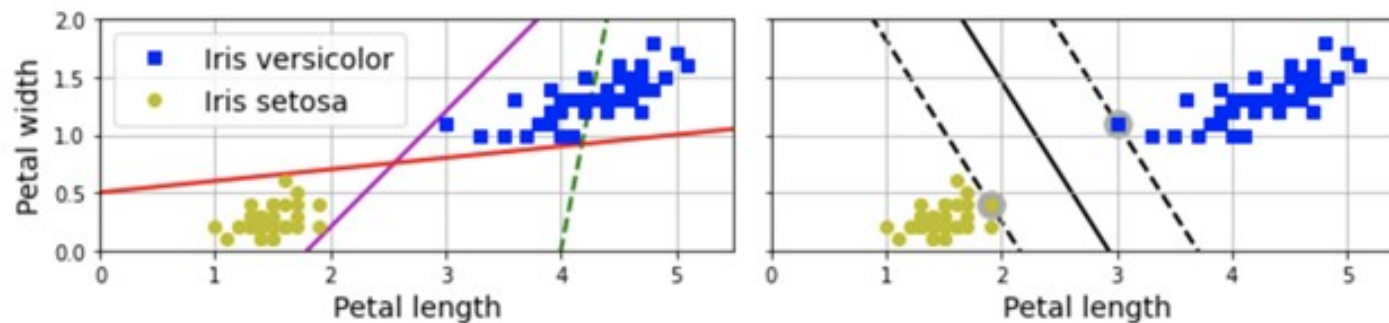
Support Vector Machine (SVM):

- Versatile Machine Learning model
- Performs linear & nonlinear classification, regression, and detection
- Works well on small to medium-sized datasets (hundreds of thousands of data points)
- Does not scale well to very large datasets

Introduction

Basic idea of SVMs:

- To find the optimal hyperplane for linearly separable patterns
- To Extend SVMs to patterns that are not linearly separable by transformations of original data to map into new space (the Kernel function)



To find the optimal hyperplane for linearly separable patterns:

The figure on the right shows 3 linear lines (purple, red, and dashed green).

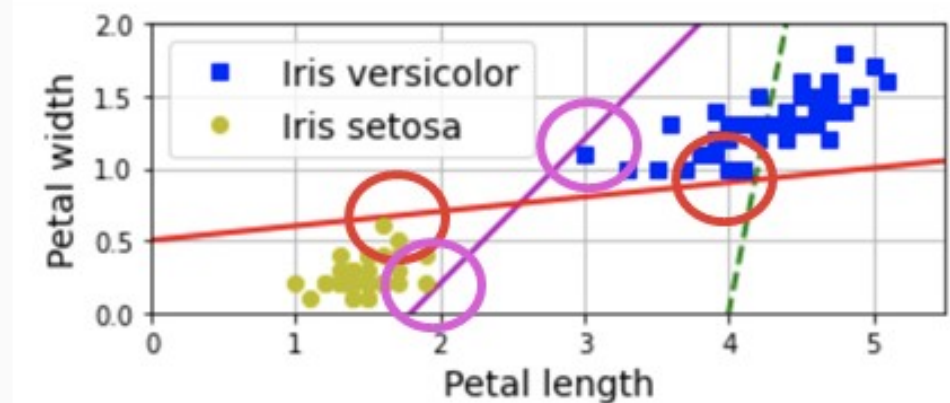
These lines are called **decision boundaries**; they **separate the training data points according to some characteristics/features/attributes**, here the color & shape.

In this dataset, there are 2 classes: the yellow dots and the blue squares.

However, not all three separate the classes properly; the green dash does not, the purple & red work perfectly on the training dataset.

One problem with these red & purple decision boundaries is that they will probably perform **poorly** on new instances **because their decision boundaries come so close to the training instances**.

These decision boundaries are not outputted by an SVM!



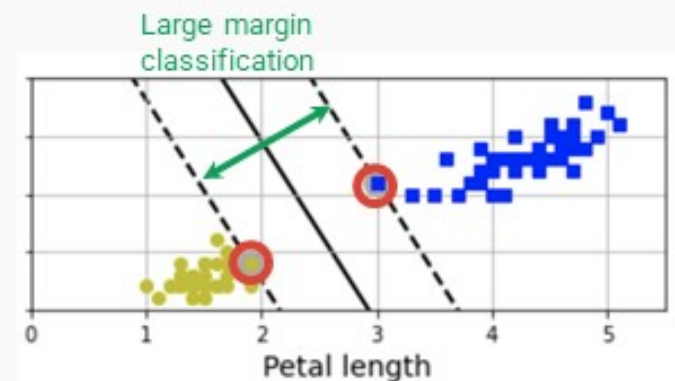
To find the optimal hyperplane for linearly separable patterns: Linear SVM

Now, the new figure on the right shows the decision boundary of **an linear SVM**.

The black line does not only separate the 2 classes but also stays as far away from the closest training instances as possible.

SVM finds the **wildest street** (Winston terminology) between classes (the parallel dashed lines). This is called the **large margin classification**.

NB: it is important to highlight that adding more training instances "off the street" will not affect the decision boundary at all. Because, the decision boundary is fully determined by the instances located on the edge of the street, i.e., on the dashed lines. These instances are called **support vectors** (circled in red).

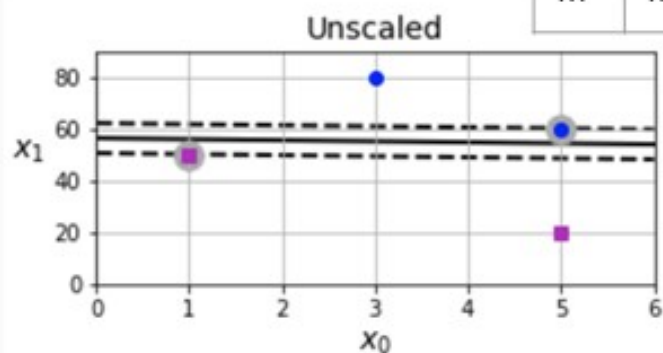


Why scaling features/attributes (*usually represented by X*) is essential when using SVMs?

In the example below, we show the same training dataset fitted to an SVM classifier:

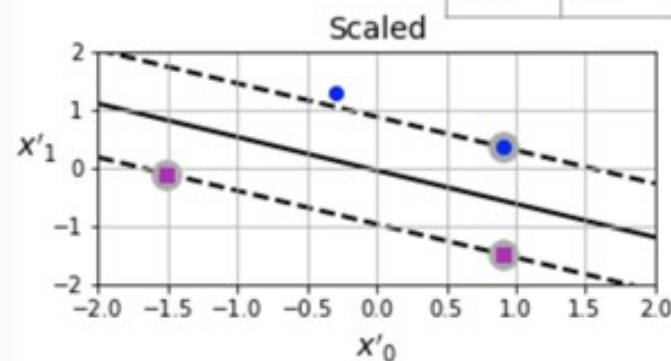
The first time without scaling:

x1	x2
1	50
5	20
...	...



The second with scaling:

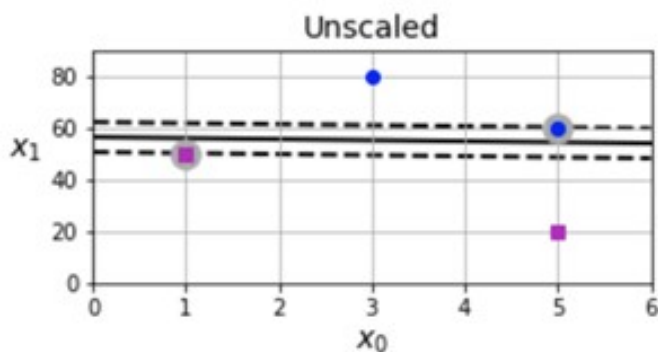
x1	x2
-1.5	-.11
-.3	1.27
...	...



Why scaling features/attributes (*usually represented by X*) is essential when using SVMs?

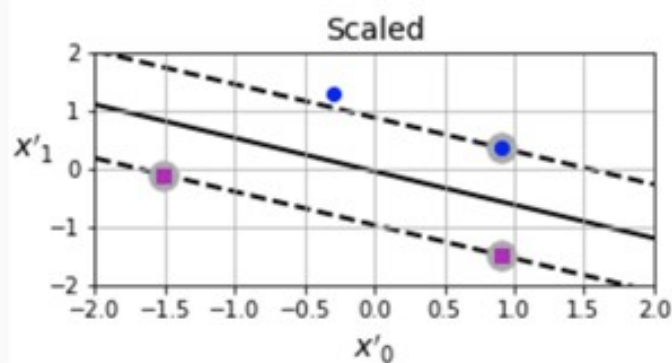
In the example below, we show the same training dataset fitted to an SVM classifier:

The first time without scaling:



x_0	x_1
1	50
5	20
...	...

The second with scaling:



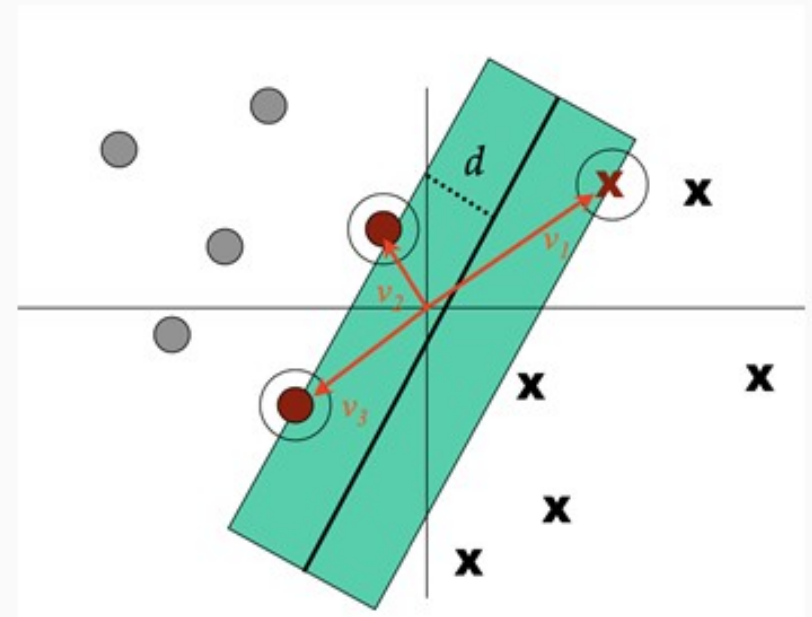
x'_0	x'_1
-1.5	-1.11
-0.3	1.27
...	...

SVMs are sensitive to feature scales. x_1 (the vertical scale) is one order of magnitude higher than x_0 (the horizontal scale) → the widest possible street is close to the horizontal scale.

After applying scaling, the decision boundary is better now.

Support Vectors

- Data points lying closely to the decision surface
- Data points that are hard to classify
- Have a direct bearing on the optimum location of the decision boundary; the position of this boundary changes if any of the training data points that are support vectors are removed
- v_1, v_2, v_3 are the support vectors and d is half the margin, i.e., half the width of the street
- The points on the decision surface are the tips of the support vectors; the circled red points.

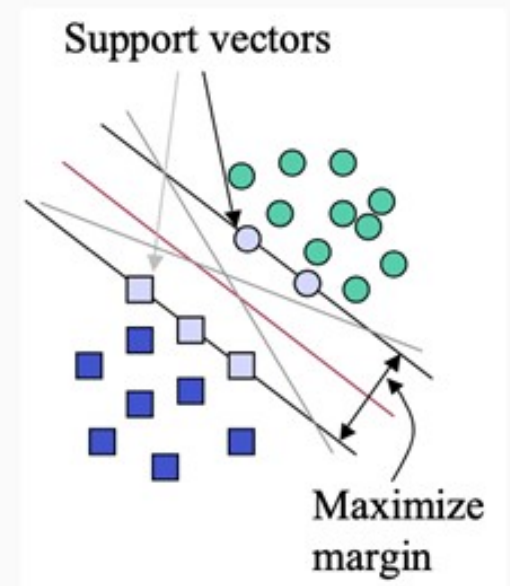


Training of an SVM model

Input: training dataset X, y ; X = features/attributes/columns = $x_1, x_2, \dots, x_i, x_n$ & y = target/label/class

Output: weights W ; one weight w_i for each feature x_i ; the linear combination of x_i & w_i predicts the value y_i .

Optimization: maximizing the margin (the street's width) to reduce the number of non zero weights to the one corresponding to the important features, i.e., the features that decide the separating decision boundaries. These nonzero weights correspond to the support vectors.



Training of an SVM model (2)

Optimization: define the hyperplanes H such that:

$w \cdot x_i + b \geq 1$ when $y_i = +1$, to avoid margin violations, we need the decision function to be greater than 1 for all positive training instances. (1)

$w \cdot x_i + b \leq -1$ when $y_i = -1$, to avoid margin violations, we need the decision function to be less than -1 for all negative training instances. (2)

We can rewrite equations (1) & (2) as $t_i \cdot (w^T \cdot x_i + b) \geq 1$ with $t_i = 1$ if positive instances and -1 for negative instances

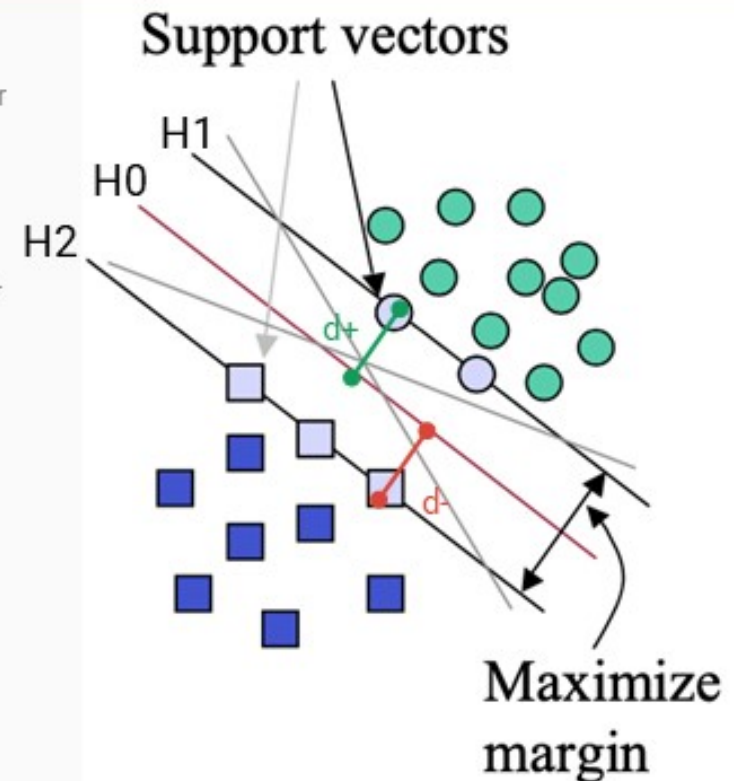
Such that $H1: w \cdot x_i + b = 1$, $H2: w \cdot x_i + b = -1$, and $H0: w \cdot x_i + b = 0$

d_+ = shortest distance to the closest positive point

d_- = shortest distance to the closest negative point

The margin of a separating hyperplane is $2 \times d$.

The objective is to increase the margin in order to separate the classes as much as possible.



Training of an SVM model (3)

Optimization:

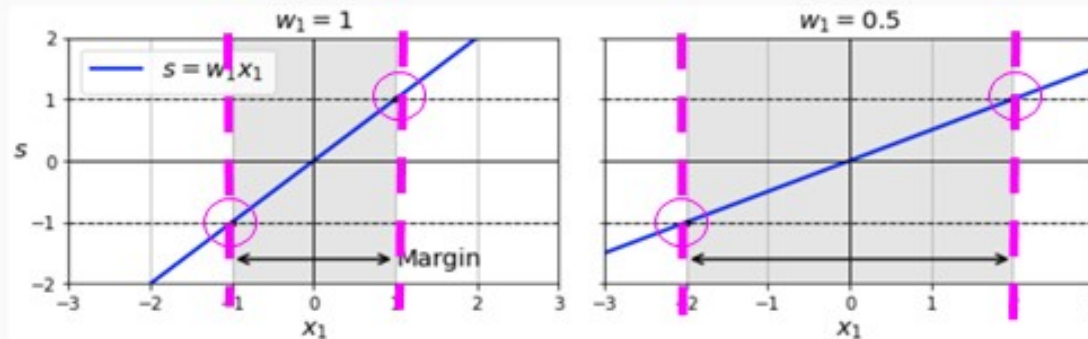
The objective is to increase the margin in order to separate the classes as much as possible → should we maximize or minimize the weights in order to maximize the margin?

To answer this, we will proceed as follows: we will define the borders of the margin (street) as the points where the decision function is either equal to 1 or -1.

In the left plot, the weight w_1 is 1, so the points at which $w_1 x_1 = -1$ or $+1$ are $x_1 = -1$ and $x_1 = +1$ → the margin size is 2.

In the right plot, the weight w_1 is 0.5, so the points at which $w_1 x_1 = -1$ or $+1$ are $x_1 = -2$ and $x_1 = +2$ → the margin size is 4.

→ To make the width of the street (i.e., the margin) larger, we need to keep the weights ...



Training of an SVM model (4)

Optimization:

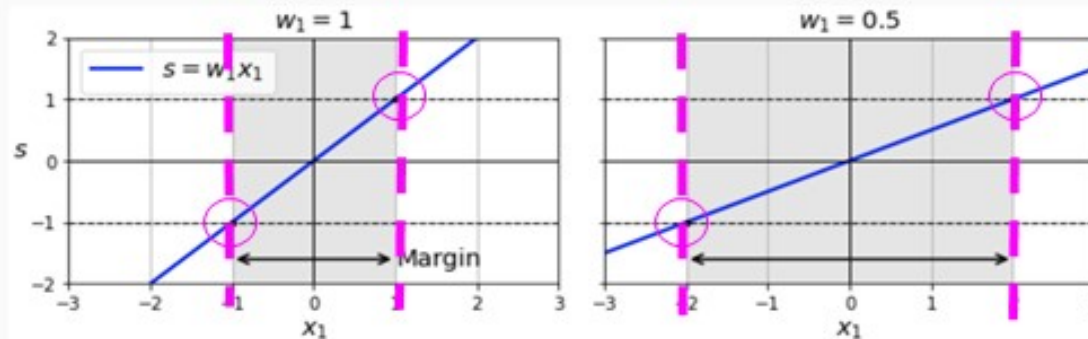
The objective is to increase the margin in order to separate the classes as much as possible → should we maximize or minimize the weights in order to maximize the margin?

To answer this, we will proceed as follows: we will define the borders of the margin (street) as the points where the decision function is either equal to 1 or -1.

In the left plot, the weight w_1 is 1, so the points at which $w_1 x_1 = -1$ or $+1$ are $x_1 = -1$ and $x_1 = +1$ → the margin size is 2.

In the right plot, the weight w_1 is 0.5, so the points at which $w_1 x_1 = -1$ or $+1$ are $x_1 = -2$ and $x_1 = +2$ → the margin size is 4.

→ To make the width of the street (i.e., the margin) larger, we need to keep the weights as small as possible



Training of an SVM model (5)

Optimization:

Hard margin linear SVM classifier objective:

$$\text{Minimize } \frac{1}{2} W^T W = \frac{1}{2} \|W\|^2$$

Subject to $t_i \cdot (w^T \cdot x_i + b) \geq 1$ for $i = 1, 2, \dots, n$; n = total number of training instances.

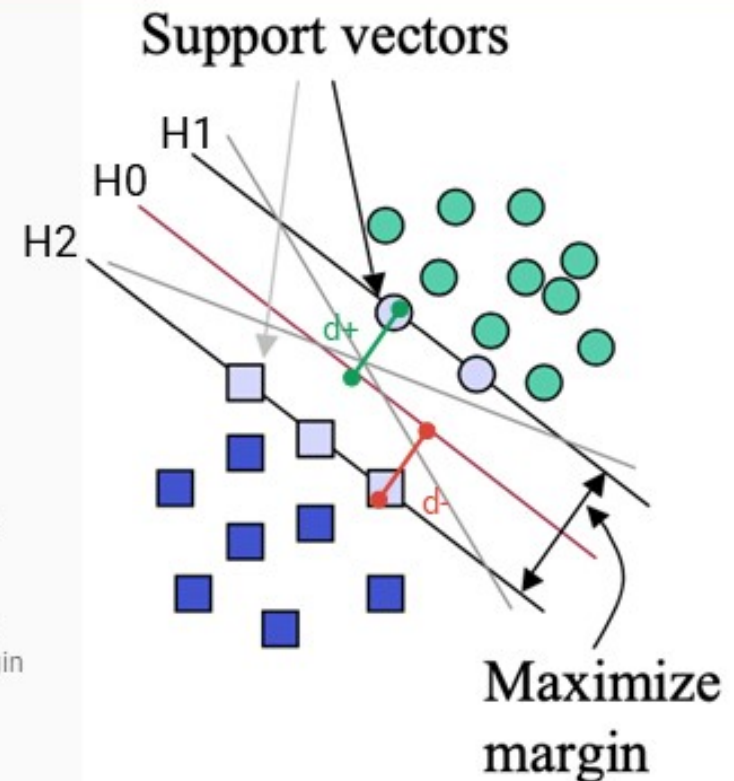
Soft margin linear SVM classifier objective:

$$\text{Minimize } \frac{1}{2} W^T W + C \sum_{i=1}^n \zeta_i$$

Subject to $t_i \cdot (w^T \cdot x_i + b) \geq 1 - \zeta_i$ for $i = 1, 2, \dots, n$ and $\zeta_i \geq 0$

The new variable introduced ζ_i is called the slack variable of the instance i ; it measures how much the i th instance is allowed to violate the margin.

The second variable added C is a hyperparameter that allows a trade-off between the 2 conflicting objectives that the soft margin SVM introduces: we need to minimize $\frac{1}{2} W^T W$ to increase the margin and minimize the slack variables to reduce the margin violations.



Training of an SVM model (6)

The hard margin and soft margin are both convex quadratic optimization problems with linear constraints. Such problems are known as quadratic programming (QP) problems.

To train an SVM, we could use a QP solver. Another is to use gradient descent to minimize the hinge loss or the squared hinge loss.

Given an instance x of the positive class ($t = 1$), the loss = 0 if the output s of the decision function $s = w^T \cdot x + b \geq 1$.

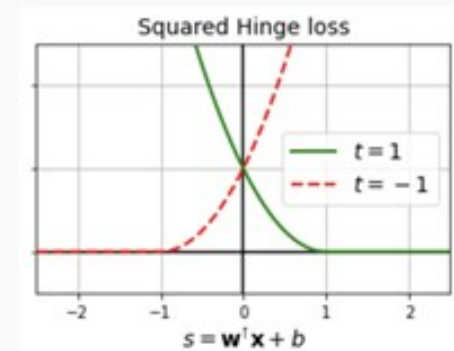
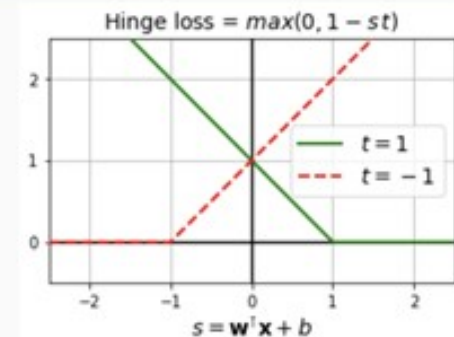
Given an instance x of the negative class ($t = -1$), the loss = 0 if the output s of the decision function $s = w^T \cdot x + b \leq -1$.

The further away an instance is from the correct side of the street, the higher the loss: it grows linearly for the hinge loss

And quadratically for the squared hinge loss

→ the squared hinge loss is more sensitive to outliers, however, if the dataset is clean, it tends to converge faster.

By default, LinearSVC uses the squared hinge loss.



Training of an SVM model (7): Hard vs Soft margin classification

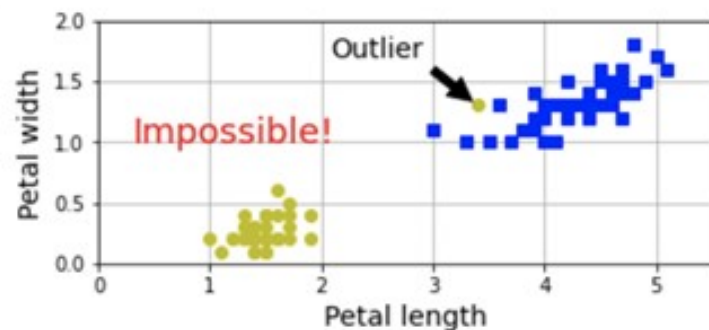
Hard margin classification: all training data points should be strictly off the street and on the correct side

Soft margin classification: a more flexible model that allows some outliers and some violations to the street or classes with the objective to find a good compromise between keeping the margin as large as possible and limiting the violations (i.e., data points in the street or in the wrong class).

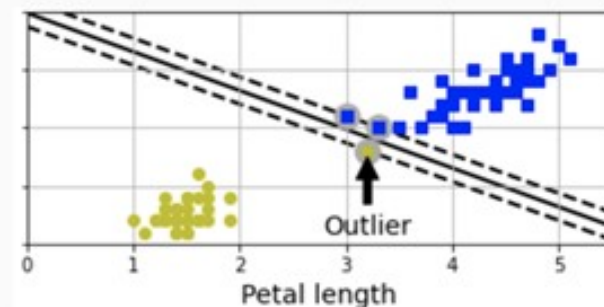
Training of an SVM model (8): Hard vs Soft margin classification

Issues with hard margin classification:

- Classification works only and only if the data is linearly separable
- Classification is sensitive to outliers



The figure on the right shows a dataset with only one data point as outlier.
Result: a hard margin SVM classifier fails to converge to a decision boundary.



The figure on the left shows a dataset with only one data point as outlier.
Result: a hard margin SVM classifier shrunk the decision boundary; this model will probably not generalize well

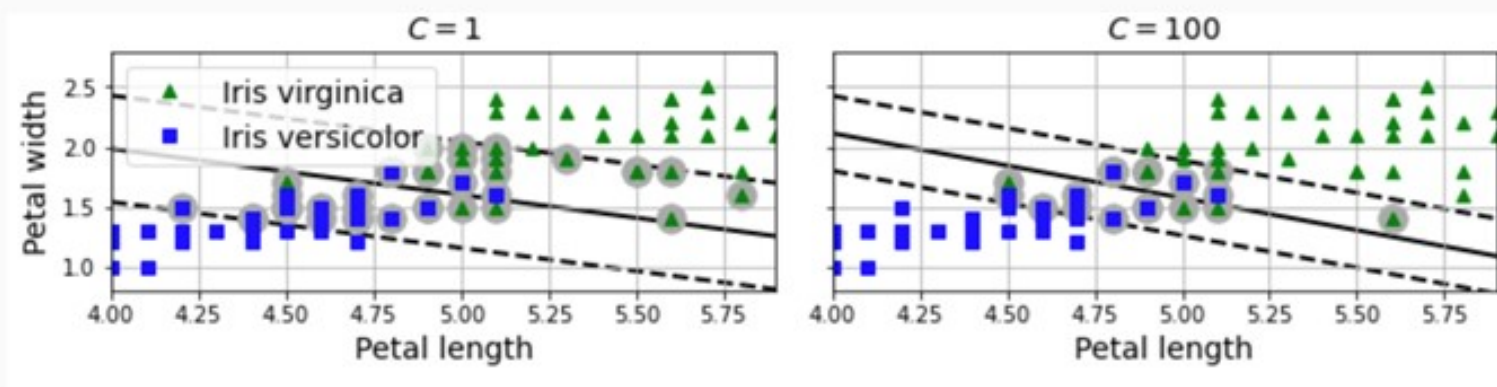
Training of an SVM model (9): Hard vs Soft margin classification

In the SVM Scikit-Learn model, the regularization hyperparameter C is responsible for the hard to soft margin classification.

Reducing C makes the margin larger (soft margin), but, it also allows more margin violations \rightarrow reducing C results in more instances supporting the margin (street) \rightarrow lower risk of overfitting.

Reducing C too much (hard margin) \rightarrow underfitting

Which of the models below generalizes better?



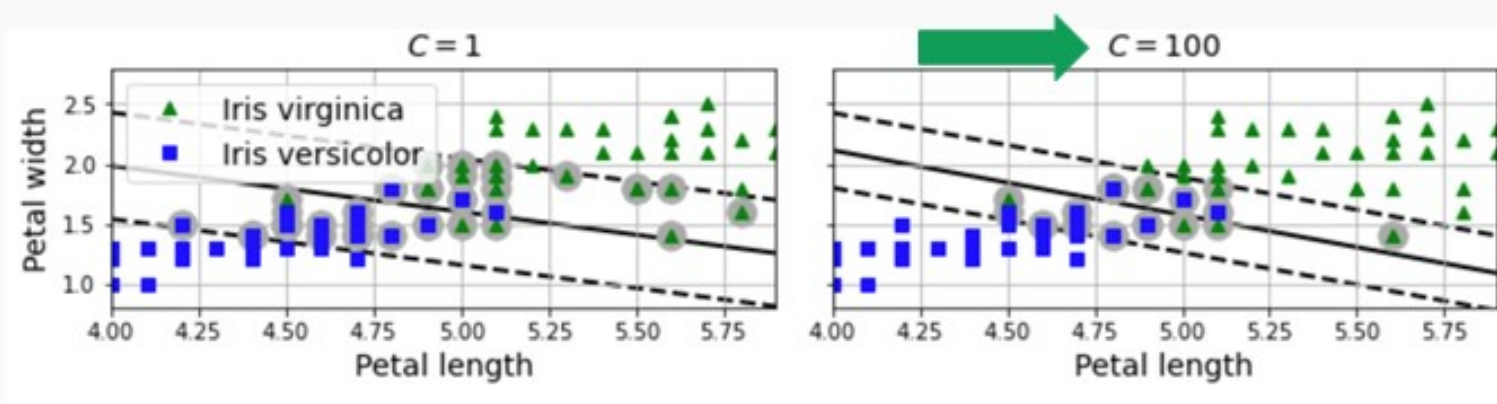
Training of an SVM model (10): Hard vs Soft margin classification

In the SVM Scikit-Learn model, the regularization hyperparameter C is responsible for the hard to soft margin classification.

Reducing C makes the margin larger (soft margin), but, it also allows more margin violations \rightarrow reducing C results in more instances supporting the margin (street) \rightarrow lower risk of overfitting.

Reducing C too much (hard margin) \rightarrow underfitting

Which of the models below generalizes better?



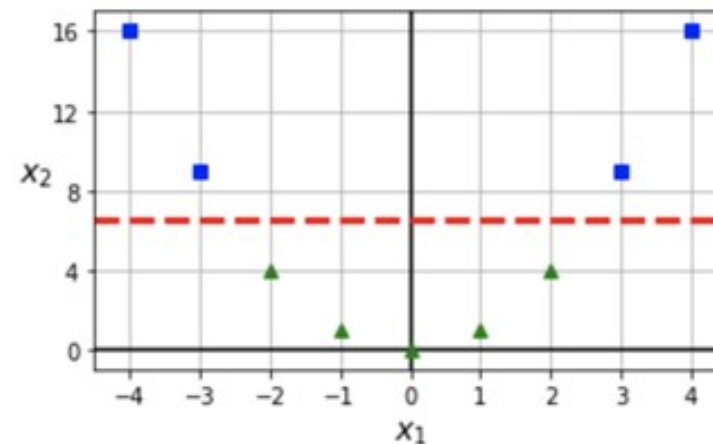
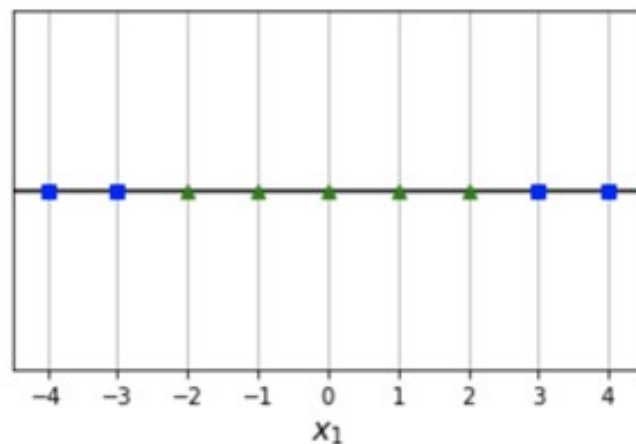
What about nonlinear data?

Nonlinear SVM classification

To deal with nonlinear datasets, more features can be added, e.g., polynomial features. In some cases, this can result in a linearly separable dataset.

The dataset shown below is not linearly separable (left figure).

Converting x_1 to x_1^2 resulted in a linearly separable dataset (right figure).



Nonlinear SVM classification: polynomial kernel

Adding polynomial features is easy to implement with Scikit-Learn and can be used with all sorts of Machine Learning algorithms (not just SVMs).

→ with a low polynomial degree, this method cannot deal with complex datasets; the more complex the data, most probably, the higher the degree of polynomial needed to approximate the classification function.

→ and with a high polynomial degree, a huge number of features is created → a slow model.

Solution: the kernel trick; we get the same result as if polynomial features were used, but without actually adding them → no combinatorial explosion of the number of features. This trick is implemented by the SVC class in Scikit-Learn with the hyperparameter *degree*.

NB: If the model is overfitting → decrease the polynomial degree
If the model is underfitting → increase the polynomial degree

Nonlinear SVM classification: similarity features

Adding features computed using a similarity function measuring how much an instance resembles a particular landmark (a particular data point). Then, recomputing the new features.

A similarity feature could be a Gaussian Radial Basis Function (RBF).

A simple approach to select the landmarks: create a landmark at the location of every instance in the dataset. This will result in many dimensions and increases the chances that the transformed training set will be linearly separable. However, with a training set of N data points and M features becomes a dataset with N features (assuming we drop the original features). If N is large \rightarrow the number of features will become so large \rightarrow the curse of dimensionality \rightarrow risk of overfitting.

The Gaussian RBF kernel K on 2 feature vectors x and x' is defined as:

Since the value of the RBF kernel decreases with distance and ranges between zero and one, it has a natural interpretation as a **similarity measure**. A gaussian RBF kernel is bell-shaped.

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right) = \exp\left(\gamma\|x - x'\|^2\right)$$

Nonlinear SVM classification: similarity features

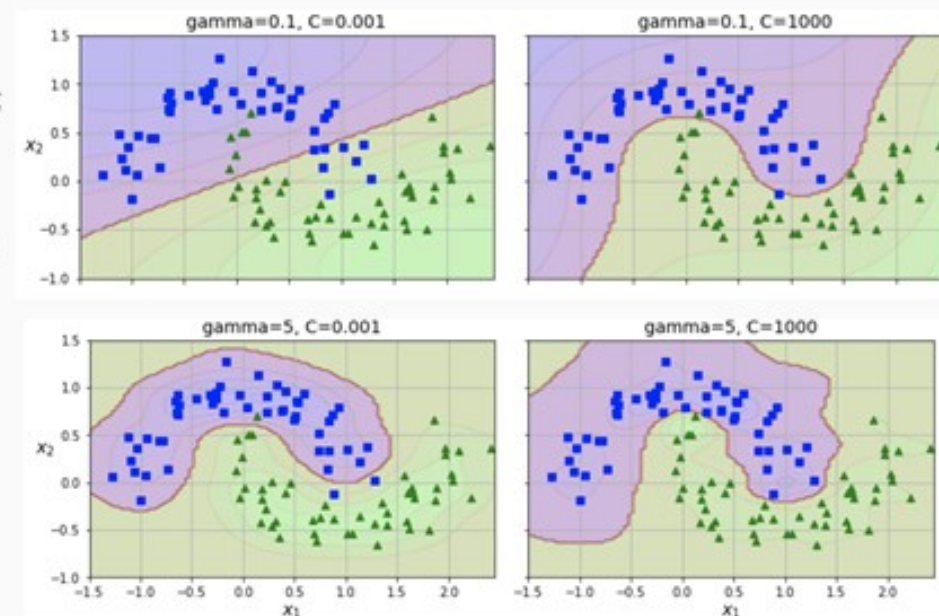
Adding similarity features is easy to implement with Scikit-Learn and can be used with all sorts of Machine Learning algorithms (not just SVMs).

→ This method is computationally expensive especially on large training sets; we have to compute all the additional features for every landmark.

→ Solution: the kernel trick; we get the same result as if similarity features were used, but without actually adding them. This trick is implemented by the SVC class in Scikit-Learn with the hyperparameter *kernel*.

Looking at the figures on the right, we can realize that:

- Increasing gamma makes the bell-shaped curve narrower → each instance's range of influence is smaller → the resulting decision boundary is irregular (it surrounds the training instances) → risk of overfitting
- Decreasing gamma makes the bell-shaped curve wider → each instance's range of influence is larger → the resulting decision boundary is smoother → risk of underfitting



Nonlinear SVM classification: similarity features

Adding similarity features is easy to implement with Scikit-Learn and can be used with all sorts of Machine Learning algorithms (not just SVMs).

→ This method is computationally expensive especially on large training sets; we have to compute all the additional features for every landmark.

→ Solution: the kernel trick; we get the same result as if similarity features were used, but without actually adding them. This trick is implemented by the SVC class in Scikit-Learn with the hyperparameter *kernel*.

NB:

- If the model is overfitting → decrease the parameter gamma (γ) of the RBF kernel
- If the model is underfitting → increase γ
- How do we choose the right kernel function? With a rule of thumb →
 - Start with the linear kernel. If it works, stop. NB: *LinearSVC* class is much faster than *SVC(kernel='linear')*.
 - If the training dataset is not too large, try kernelized SVM, starting with the Gaussian RBF. If it works, stop.
 - Try specialized kernels like sigmoid, tanh, etc., <https://data-flair.training/blogs/svm-kernel-functions/>; time & computing power are needed.

Computational complexity

n =number of training data instances

m =number of features

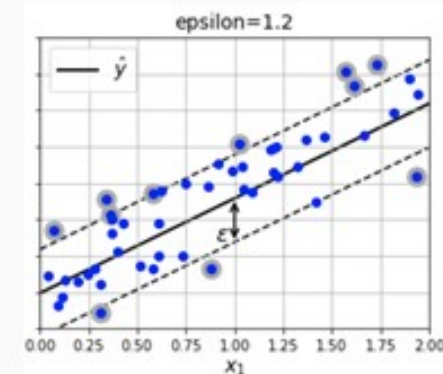
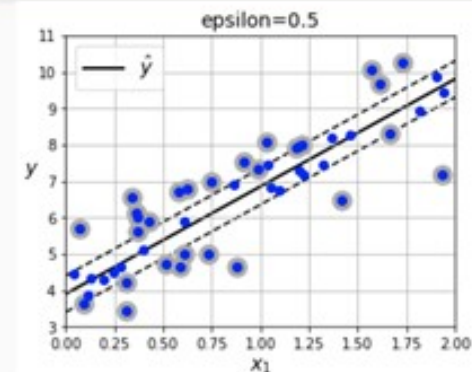
Scikit-Learn Class	Time Complexity	Scaling Required	Kernel trick	Note
LinearSVC	$\Theta(nm)$	Yes	No	The model's precision is controlled by the hyperparameter "tol"; the tolerance. The higher the precision, the lower the tol value.
SVC	$\Theta(n^2m)$ to $\Theta(n^3m)$	Yes	Yes	This model is best for small to medium-sized nonlinear training set (hundreds of thousands of instances), especially with sparse features. With sparse features, the model scales with the average number of nonzero features per instance.

Computational complexity (2)

If we have a linear sparse data what model should we use?

SVM for regression

- In SVM for regression, the objective is to fit as many instances as possible on the street while limiting violations (off the street); contrary to SVM for classification, where we fit the largest possible street between 2 classes.
- The width of the street is controlled by the hyperparameter *epsilon*
- Reducing *epsilon* increases the number of support vectors, which regularizes the model.
- SVM for regression is *epsilon-insensitive*; if more training instances were added within the margin, the model's predictions are not affected.
- LinearSVR is the Scikit-Learn's class for linear SVM Regression.
- For nonlinear regression, a kernelized SVM is used; the SVR class.



References

<https://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf>

Géron, Aurélien. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow. " O'Reilly Media, Inc.", 2022.

An Idiot's guide to Support vector machines (SVMs)

R. Berwick, Village Idiot

SVMs: A New Generation of Learning Algorithms

- Pre 1980:
 - Almost all learning methods learned linear decision surfaces.
 - Linear learning methods have nice theoretical properties
- 1980's
 - Decision trees and NNs allowed efficient learning of non-linear decision surfaces
 - Little theoretical basis and all suffer from local minima
- 1990's
 - Efficient learning algorithms for non-linear functions based on computational learning theory developed
 - Nice theoretical properties.

Key Ideas

- Two independent developments within last decade
 - New efficient separability of non-linear regions that use “kernel functions” : generalization of ‘similarity’ to new kinds of similarity measures based on dot products
 - Use of quadratic optimization problem to avoid ‘local minimum’ issues with neural nets
 - The resulting learning algorithm is an optimization algorithm rather than a greedy search

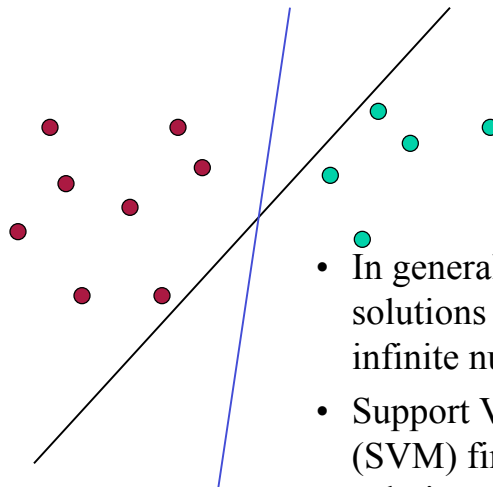
Organization

- Basic idea of support vector machines: just like 1-layer or multi-layer neural nets
 - Optimal hyperplane for linearly separable patterns
 - Extend to patterns that are not linearly separable by transformations of original data to map into new space – the Kernel function
- SVM algorithm for pattern recognition

Support Vectors

- Support vectors are the data points that lie closest to the decision surface (or hyperplane)
- They are the data points most difficult to classify
- They have direct bearing on the optimum location of the decision surface
- We can show that the optimal hyperplane stems from the function class with the lowest “capacity” = # of independent features/parameters we can twiddle [note this is ‘extra’ material not covered in the lectures... you don’t have to know this]

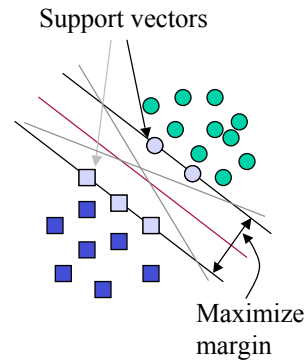
Recall from 1-layer nets : Which Separating Hyperplane?



- In general, lots of possible solutions for a, b, c (an infinite number!)
- Support Vector Machine (SVM) finds an optimal solution

Support Vector Machine (SVM)

- SVMs maximize the margin (Winston terminology: the 'street') around the separating hyperplane.
- The decision function is fully specified by a (usually very small) subset of training samples, the support vectors.
- This becomes a Quadratic programming problem that is easy to solve by standard methods



Separation by Hyperplanes

- Assume linear separability for now (we will relax this later)
- in 2 dimensions, can separate by a line
 - in higher dimensions, need hyperplanes

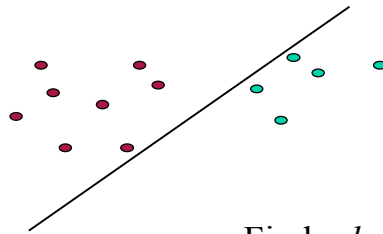
General input/output for SVMs just like for neural nets, but for one important addition...

Input: set of (input, output) training pair samples; call the input sample features $x_1, x_2 \dots x_n$, and the output result y . Typically, there can be lots of input features x_i .

Output: set of weights \mathbf{w} (or w_i), one for each feature, whose linear combination predicts the value of y . (So far, just like neural nets...)

Important difference: we use the optimization of maximizing the margin ('street width') to reduce the number of weights that are nonzero to just a few that correspond to the important features that 'matter' in deciding the separating line(hyperplane)...these nonzero weights correspond to the support vectors (because they 'support' the separating hyperplane)

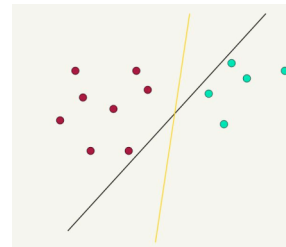
2-D Case



Find a, b, c , such that
 $ax + by \geq c$ for red points
 $ax + by \leq$ (or $<$) c for green points.

Which Hyperplane to pick?

- Lots of possible solutions for a, b, c .
- Some methods find a separating hyperplane, but not the optimal one (e.g., neural net)
- But: Which points should influence optimality?
 - All points?
 - Linear regression
 - Neural nets
 - Or only “difficult points” close to decision boundary
 - Support vector machines

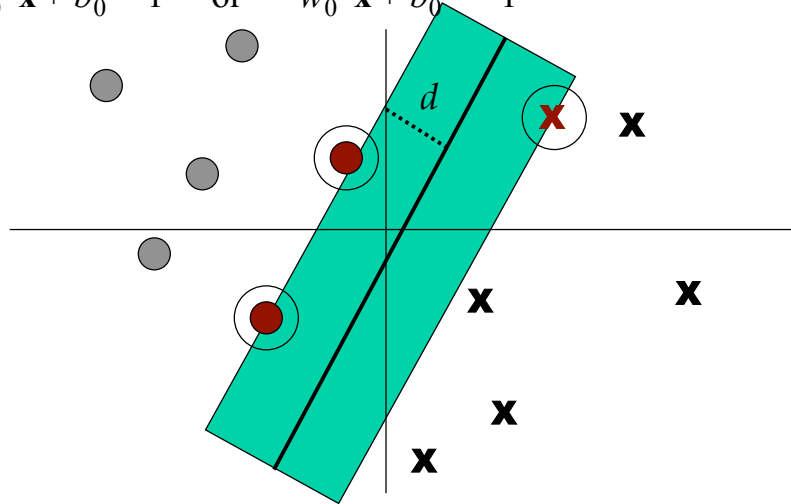


Support Vectors again for linearly separable case

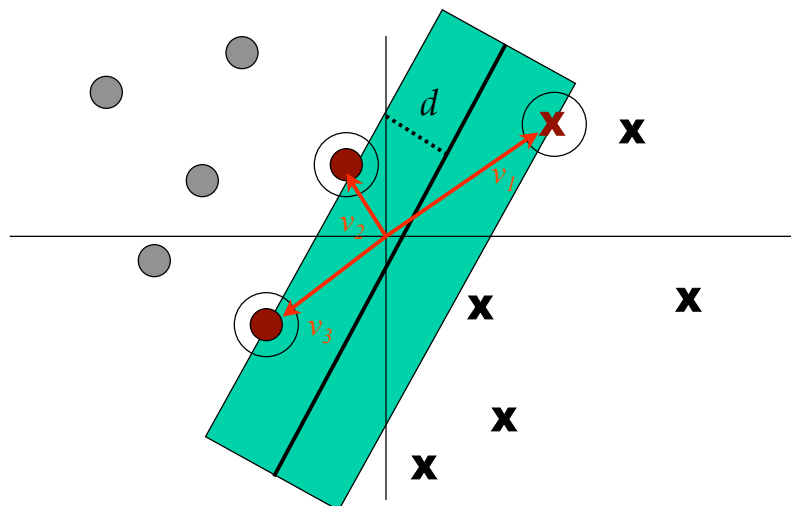
- Support vectors are the elements of the training set that would change the position of the dividing hyperplane if removed.
- Support vectors are the critical elements of the training set
- The problem of finding the optimal hyper plane is an optimization problem and can be solved by optimization techniques (we use Lagrange multipliers to get this problem into a form that can be solved analytically).

Support Vectors: Input vectors that just touch the boundary of the margin (street) – circled below, there are 3 of them (or, rather, the ‘tips’ of the vectors)

$$w_0^T \mathbf{x} + b_0 = 1 \quad \text{or} \quad w_0^T \mathbf{x} + b_0 = -1$$



Here, we have shown the actual support vectors, v_1, v_2, v_3 , instead of just the 3 circled points at the tail ends of the support vectors. d denotes 1/2 of the street ‘width’



Definitions

Define the hyperplanes H such that:

$$w \cdot x_i + b \geq +1 \text{ when } y_i = +1$$

$$w \cdot x_i + b \leq -1 \text{ when } y_i = -1$$

H_1 and H_2 are the planes:

$$H_1: w \cdot x_i + b = +1$$

$$H_2: w \cdot x_i + b = -1$$

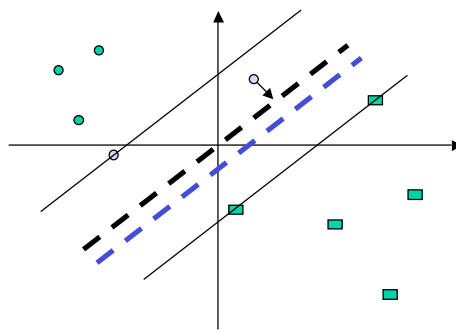
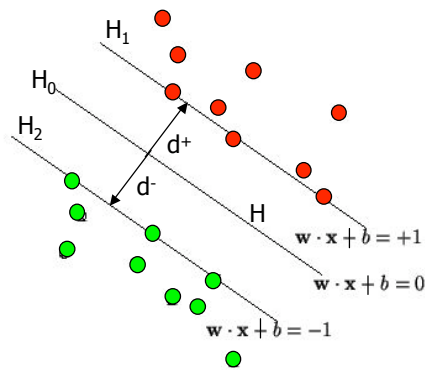
The points on the planes H_1 and H_2 are the tips of the Support Vectors

The plane H_0 is the median in between, where $w \cdot x_i + b = 0$

d^+ = the shortest distance to the closest positive point

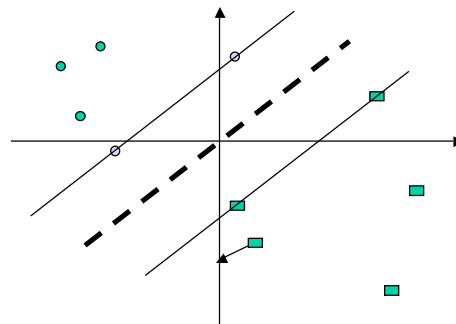
d^- = the shortest distance to the closest negative point

The margin (gutter) of a separating hyperplane is $d^+ + d^-$.



Moving the other vectors
has no effect

Moving a support vector
moves the decision
boundary



The optimization algorithm to generate the weights proceeds in such a way that only the support vectors determine the weights and thus the boundary

Defining the separating Hyperplane

- Form of equation defining the decision surface separating the classes is a hyperplane of the form:

$$\mathbf{w}^T \mathbf{x} + b = 0$$

- **w is a weight vector**
 - **x is input vector**
 - **b is bias**
- Allows us to write
$$\mathbf{w}^T \mathbf{x} + b \geq 0 \text{ for } d_i = +1$$
$$\mathbf{w}^T \mathbf{x} + b < 0 \text{ for } d_i = -1$$

Some final definitions

- Margin of Separation (d): the separation between the hyperplane and the closest data point for a given weight vector \mathbf{w} and bias b .
- Optimal Hyperplane (maximal margin): the particular hyperplane for which the margin of separation d is maximized.

Maximizing the margin (aka street width)

We want a classifier (linear separator)
with as big a margin as possible.

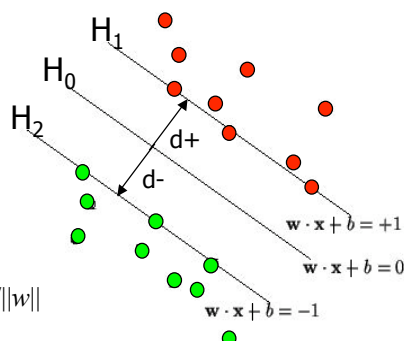
Recall the distance from a point (x_0, y_0) to a line:

$Ax + By + c = 0$ is: $|Ax_0 + By_0 + c| / \sqrt{A^2 + B^2}$, so,

The distance between H_0 and H_1 is then:

$|w \cdot x + b| / \|w\| = 1 / \|w\|$, so

The total distance between H_1 and H_2 is thus: $2 / \|w\|$



In order to maximize the margin, we thus need to minimize $\|w\|$. With the condition that there are no datapoints between H_1 and H_2 :

$x_i \cdot w + b \geq +1$ when $y_i = +1$

$x_i \cdot w + b \leq -1$ when $y_i = -1$

Can be combined into: $y_i(x_i \cdot w) \geq 1$

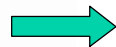
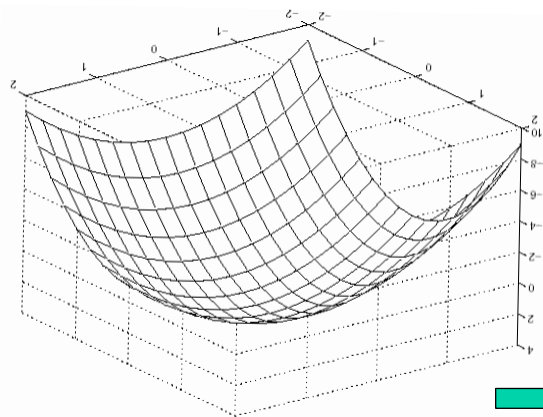
We now must solve a quadratic programming problem

- Problem is: minimize $\|w\|$, s.t. discrimination boundary is obeyed, i.e., $\min f(x)$ s.t. $g(x)=0$, which we can rewrite as:
 $\min f: \frac{1}{2} \|w\|^2$ (Note this is a quadratic function)
s.t. $g: y_i(w \cdot x_i) - b = 1$ or $[y_i(w \cdot x_i) - b] - 1 = 0$

This is a constrained optimization problem

It can be solved by the Lagrangian multiplier method

Because it is quadratic, the surface is a paraboloid, with just a single global minimum (thus avoiding a problem we had with neural nets!)

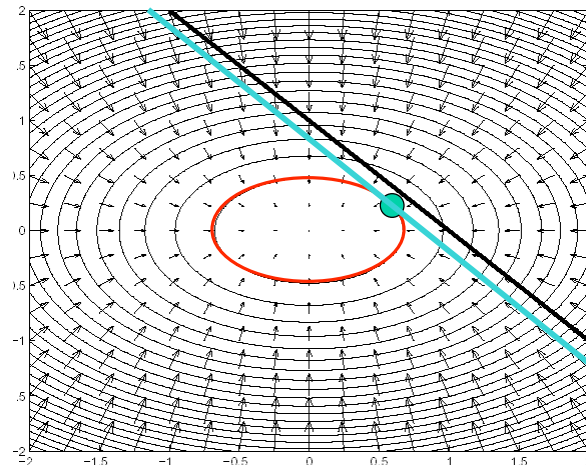


flatten

Example: paraboloid $2+x^2+2y^2$ s.t. $x+y=1$

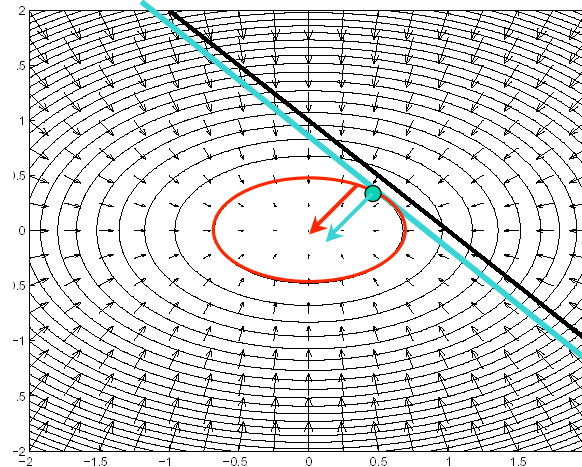
Intuition: find intersection of two functions f, g at a tangent point (intersection = both constraints satisfied; tangent = derivative is 0); this will be a min (or max) for f s.t. the constraint g is satisfied

Flattened paraboloid $f: 2x^2+2y^2=0$ with superimposed constraint $g: x+y=1$



Minimize when the constraint line g (shown in green) is tangent to the inner ellipse contour linez of f (shown in red) – note direction of gradient arrows.

flattened paraboloid $f: 2+x^2+2y^2=0$ with superimposed constraint $g: x+y=1$; at tangent solution p , gradient vectors of f, g are parallel (no possible move to increment f that also keeps you in region g)



Minimize when the constraint line g is tangent to the inner ellipse contour line of f

Two constraints

1. Parallel normal constraint (= gradient constraint on f, g s.t. solution is a max, or a min)
2. $g(x)=0$ (solution is on the constraint line as well)

We now recast these by combining f, g as the new Lagrangian function by introducing new ‘slack variables’ denoted a or (more usually, denoted α in the literature)

Redescribing these conditions

- Want to look for solution point p where

$$\nabla f(p) = \nabla \lambda g(p)$$

$$g(x) = 0$$

- Or, combining these two as the *Langrangian* L & requiring derivative of L be zero:

$$L(x, a) = f(x) - ag(x)$$

$$\nabla(x, a) = 0$$

At a solution p

- The the constraint line g and the contour lines of f must be tangent
- If they are tangent, their gradient vectors (perpendiculars) are parallel
- Gradient of g must be 0 – i.e., steepest ascent & so perpendicular to f
- Gradient of f must also be in the same direction as g

How Lagrangian solves constrained optimization

$$L(x, a) = f(x) - a g(x) \text{ where}$$

$$\nabla L(x, a) = 0$$

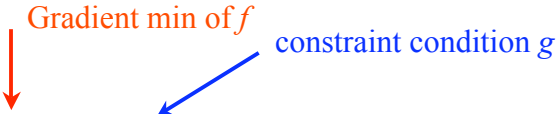
Partial derivatives wrt x recover the parallel normal constraint

Partial derivatives wrt λ recover the $g(x, y) = 0$

In general,

$$L(x, a) = f(x) + \sum_i a_i g_i(x)$$

In general


 $L(x, a) = f(x) + \sum_i a_i g_i(x)$ a function of $n + m$ variables
 n for the x 's, m for the a . Differentiating gives $n + m$ equations, each set to 0. The n eqns differentiated wrt each x_i give the gradient conditions; the m eqns differentiated wrt each a_i recover the constraints g_i

In our case, $f(x)$: $\frac{1}{2} \| \mathbf{w} \|^2$; $g(x)$: $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 = 0$ so Lagrangian is:

$$\min L = \frac{1}{2} \| \mathbf{w} \|^2 - \sum_i a_i [y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1] \text{ wrt } \mathbf{w}, b$$

We expand the last to get the following L form:

$$\min L = \frac{1}{2} \| \mathbf{w} \|^2 - \sum_i a_i y_i (\mathbf{w} \cdot \mathbf{x}_i + b) + \sum_i a_i \text{ wrt } \mathbf{w}, b$$

Lagrangian Formulation

- So in the SVM problem the Lagrangian is
$$\min L_p = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^l a_i y_i (\mathbf{x}_i \cdot \mathbf{w} + b) + \sum_{i=1}^l a_i$$
s.t. $\forall i, a_i \geq 0$ where l is the # of training points
- From the property that the derivatives at min = 0 we get:
$$\frac{\partial L_p}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^l a_i y_i \mathbf{x}_i = 0$$
$$\frac{\partial L_p}{\partial b} = \sum_{i=1}^l a_i y_i = 0 \quad \text{so}$$
$$\mathbf{w} = \sum_{i=1}^l a_i y_i \mathbf{x}_i, \quad \sum_{i=1}^l a_i y_i = 0$$

What's with this L_p business?

- This indicates that this is the primal form of the optimization problem
- We will actually solve the optimization problem by now solving for the dual of this original problem
- What is this dual formulation?

The Lagrangian Dual Problem: instead of minimizing over \mathbf{w} , b , subject to constraints involving a 's, we can maximize over a (the dual variable) subject to the relations obtained previously for \mathbf{w} and b

Our solution must satisfy these two relations:

$$\mathbf{w} = \sum_{i=1}^l a_i y_i \mathbf{x}_i, \quad \sum_{i=1}^l a_i y_i = 0$$

By substituting for \mathbf{w} and b back in the original eqn we can get rid of the dependence on \mathbf{w} and b .

Note first that we already now have our answer for what the weights \mathbf{w} must be: they are a linear combination of the training inputs and the training outputs, x_i and y_i and the values of a . We will now solve for the a 's by differentiating the dual problem wrt a , and setting it to zero. Most of the a 's will turn out to have the value zero. The non-zero a 's will correspond to the support vectors

Primal problem:

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

s.t. $\forall i \ a_i \geq 0$

$$\mathbf{w} = \sum_{i=1}^l a_i y_i \mathbf{x}_i, \quad \sum_{i=1}^l a_i y_i = 0$$

Dual problem:

$$\max L_D(a_i) = \sum_{i=1}^l a_i - \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l a_i a_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

s.t. $\sum_{i=1}^l a_i y_i = 0 \ \& \ a_i \geq 0$


(note that we have removed the dependence on \mathbf{w} and b)

The Dual problem

- Kuhn-Tucker theorem: the solution we find here will be the same as the solution to the original problem
- Q: But why are we doing this???? (why not just solve the original problem????)
- Ans: Because this will let us solve the problem by computing the just the inner products of x_i, x_j (which will be very important later on when we want to solve non-linearly separable classification problems)

The Dual Problem

Dual problem:

$$\begin{aligned} \max L_D(a_i) &= \sum_{i=1}^l a_i - \frac{1}{2} \sum_{i=1}^l a_i a_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \\ \text{s.t. } \sum_{i=1}^l a_i y_i &= 0 \text{ \& } a_i \geq 0 \end{aligned}$$


Notice that all we have are the dot products of x_i, x_j

If we take the derivative wrt a and set it equal to zero, we get the following solution, so we can solve for a_i :

$$\begin{aligned} \sum_{i=1}^l a_i y_i &= 0 \\ 0 \leq a_i &\leq C \end{aligned}$$

Now knowing the a_i we can find the weights \mathbf{w} for the maximal margin separating hyperplane:

$$\mathbf{w} = \sum_{i=1}^l a_i y_i \mathbf{x}_i$$

And now, after training and finding the \mathbf{w} by this method, given an unknown point u measured on features x_i we can classify it by looking at the sign of:

$$f(x) = \mathbf{w} \cdot \mathbf{u} + b = \left(\sum_{i=1}^l a_i y_i \mathbf{x}_i \cdot \mathbf{u} \right) + b$$

Remember: most of the weights \mathbf{w}_i , i.e., the a 's, will be zero
Only the support vectors (on the gutters or margin) will have nonzero weights or a 's – this reduces the dimensionality of the solution

Inner products, similarity, and SVMs

Why should inner product kernels be involved in pattern recognition using SVMs, or at all?

- Intuition is that inner products provide some measure of ‘similarity’
 - Inner product in 2D between 2 vectors of unit length returns the cosine of the angle between them = how ‘far apart’ they are
- e.g. $\mathbf{x} = [1, 0]^T$, $\mathbf{y} = [0, 1]^T$
- i.e. if they are parallel their inner product is 1 (completely similar)

$$\mathbf{x}^T \mathbf{y} = \mathbf{x} \cdot \mathbf{y} = 1$$

If they are perpendicular (completely unlike) their inner product is 0 (so should not contribute to the correct classifier)

$$\mathbf{x}^T \mathbf{y} = \mathbf{x} \cdot \mathbf{y} = 0$$

Insight into inner products

Consider that we are trying to maximize the form:

$$L_D(a_i) = \sum_{i=1}^l a_i - \frac{1}{2} \sum_{i=1}^l a_i a_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$\text{s.t. } \sum_{i=1}^l a_i y_i = 0 \text{ \& } a_i \geq 0$$

The claim is that this function will be maximized if we give nonzero values to a 's that correspond to the support vectors, ie, those that 'matter' in fixing the maximum width margin ('street'). Well, consider what this looks like. Note first from the constraint condition that all the a 's are positive. Now let's think about a few cases.

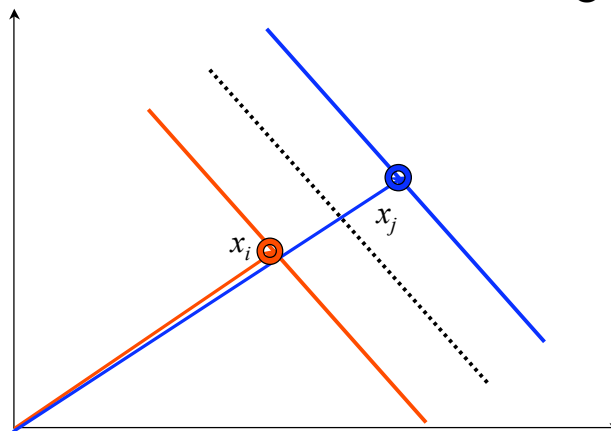
Case 1. If two features x_i, x_j are completely dissimilar, their dot product is 0, and they don't contribute to L .

Case 2. If two features x_i, x_j are completely alike, their dot product is 1. There are 2 subcases.

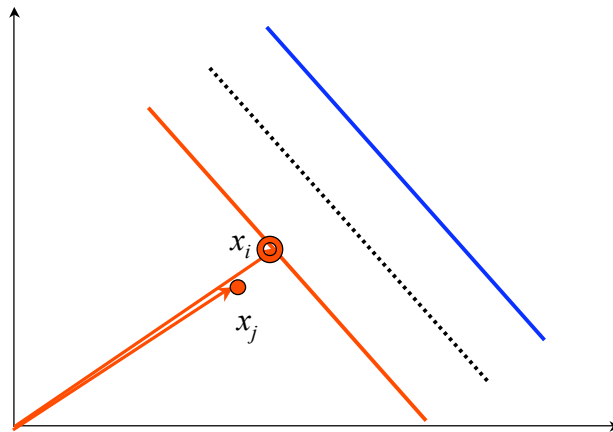
Subcase 1: both x_i and x_j predict the same output value y_i (either +1 or -1). Then $y_i \times y_j$ is always 1, and the value of $a_i a_j y_i y_j x_i \cdot x_j$ will be positive. But this would decrease the value of L (since it would subtract from the first term sum). So, the algorithm downgrades similar feature vectors that make the same prediction.

Subcase 2: x_i and x_j make opposite predictions about the output value y_i (ie, one is +1, the other -1), but are otherwise very closely similar: then the product $a_i a_j y_i y_j x_i \cdot x_j$ is negative and we are subtracting it, so this adds to the sum, maximizing it. This is precisely the examples we are looking for: the critical ones that tell the two classes apart.

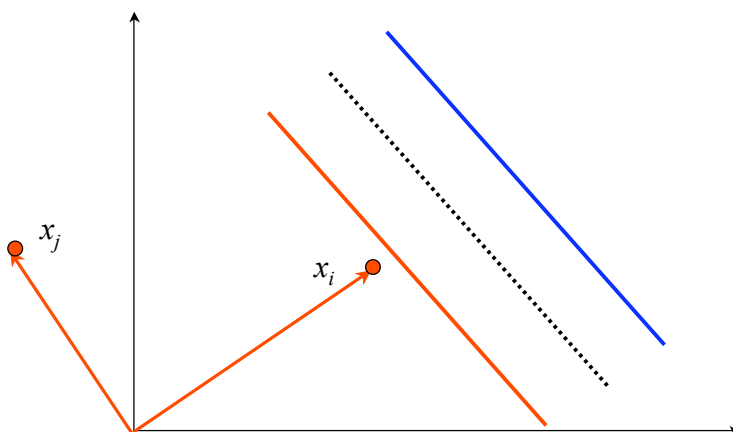
Insight into inner products, graphically: 2 very similar x_i, x_j vectors that predict diff't classes tend to maximize the margin width



2 vectors that are similar but predict the same class are redundant

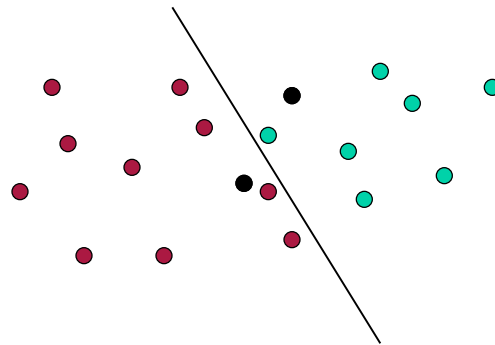


2 dissimilar (orthogonal) vectors don't count at all



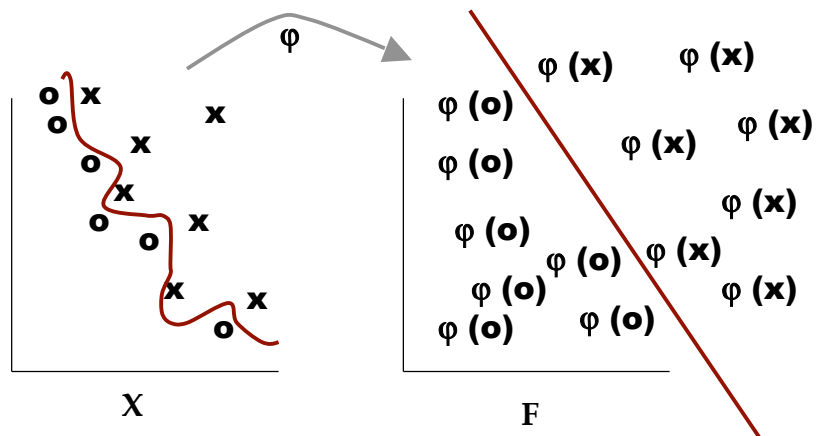
But...are we done???

Not Linearly Separable!



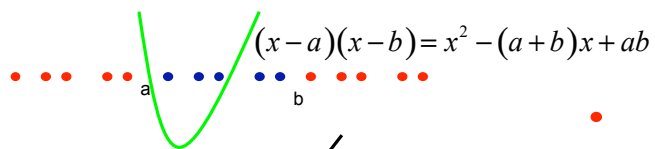
Find a line that penalizes
points on “the wrong side”

Transformation to separate

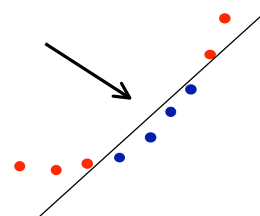


Non-Linear SVMs

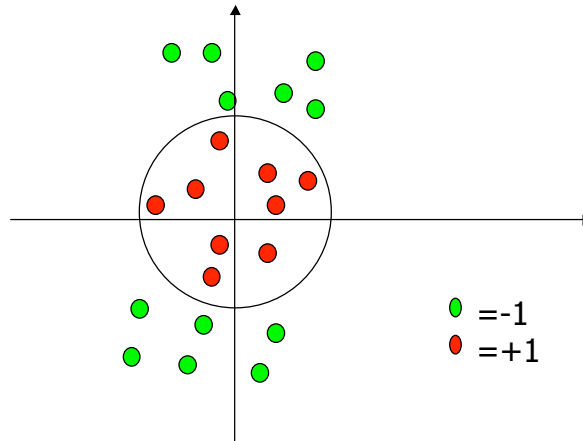
- The idea is to gain linearly separation by mapping the data to a higher dimensional space
 - The following set can't be separated by a linear function, but can be separated by a quadratic one



- So if we map $x \mapsto \{x^2, x\}$ we gain linear separation

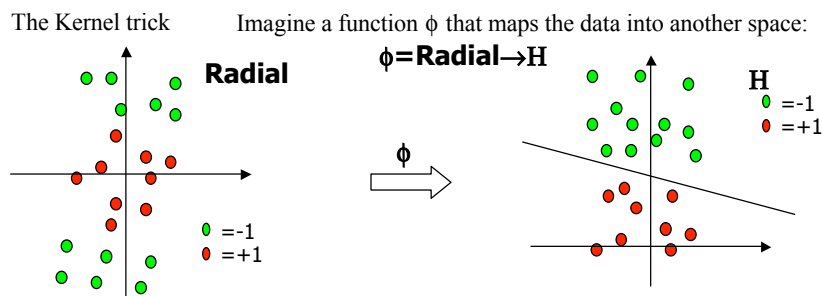


Problems with linear SVM



What if the decision function is not linear? What transform would separate these?

Ans: polar coordinates! Non-linear SVM



Remember the function we want to optimize: $L_d = \sum a_i - \frac{1}{2} \sum a_i a_j y_i y_j (x_i \cdot x_j)$ where $(x_i \cdot x_j)$ is the dot product of the two feature vectors. If we now transform to ϕ , instead of computing this dot product $(x_i \cdot x_j)$ we will have to compute $(\phi(x_i) \cdot \phi(x_j))$. But how can we do this? This is expensive and time consuming (suppose ϕ is a quartic polynomial... or worse, we don't know the function explicitly. Well, here is the neat thing:

If there is a "kernel function" K such that $K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$, then we do not need to know or compute ϕ at all!! That is, the kernel function defines inner products in the transformed space. Or, it defines similarity in the transformed space.

Non-linear SVMs

So, the function we end up optimizing is:

$$L_d = \sum a_i - \frac{1}{2} \sum a_i a_j y_i y_j K(x_i \cdot x_j),$$

Kernel example: The polynomial kernel

$K(x_i, x_j) = (x_i \cdot x_j + 1)^p$, where p is a tunable parameter

Note: Evaluating K only requires one addition and one exponentiation more than the original dot product

Examples for Non Linear SVMs

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

$$K(\mathbf{x}, \mathbf{y}) = \exp \left\{ -\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2} \right\}$$

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

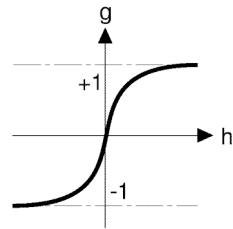
1st is polynomial (includes $\mathbf{x} \cdot \mathbf{x}$ as special case)

2nd is radial basis function (gaussians)

3rd is sigmoid (neural net activation function)

We've already seen such nonlinear transforms...

- What is it???
- $\tanh(\beta_0 x^T x_i + \beta_1)$
- It's the sigmoid transform (for neural nets)
- So, SVMs subsume neural nets! (but w/o their problems...)



Inner Product Kernels

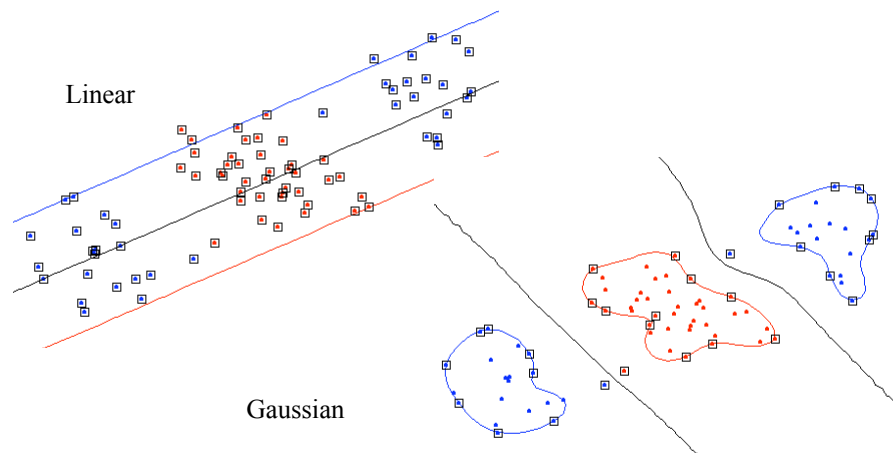
Type of Support Vector Machine	Inner Product Kernel $K(x, x_i), i = 1, 2, \dots, N$	Usual inner product
Polynomial learning machine	$(x^T x_i + 1)^p$	Power p is specified <i>a priori</i> by the user
Radial-basis function (RBF)	$\exp(1/(2\sigma^2) x-x_i ^2)$	The width σ^2 is specified <i>a priori</i>
Two layer neural net	$\tanh(\beta_0 x^T x_i + \beta_1)$	Actually works only for some values of β_0 and β_1

Kernels generalize the notion of ‘inner product similarity’

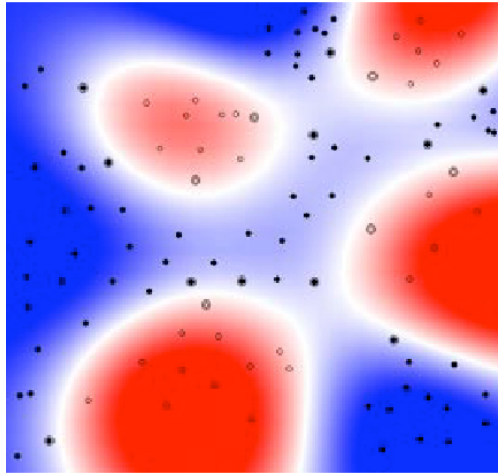
Note that one can define kernels over more than just vectors: strings, trees, structures, ... in fact, just about anything

A very powerful idea: used in comparing DNA, protein structure, sentence structures, etc.

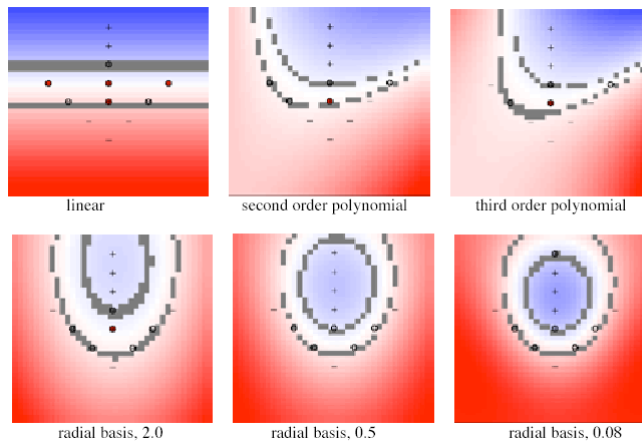
Examples for Non Linear SVMs 2 – Gaussian Kernel



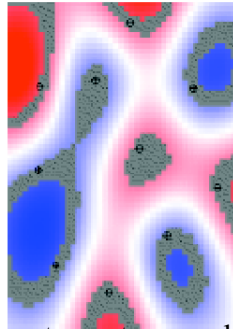
Nonlinear rbf kernel



Admiral's delight w/ diff kernel functions



Overfitting by SVM



Every point is a support vector... too much freedom to bend to fit the training data – no generalization.

In fact, SVMs have an ‘automatic’ way to avoid such issues, but we won’t cover it here... see the book by Vapnik, 1995. (We add a penalty function for mistakes made after training by over-fitting: recall that if one over-fits, then one will tend to make errors on new data.

This penalty fn can be put into the quadratic programming problem directly. You don’t need to know this for this course.)