Lecture 5: More on Kernels. SVM regression and classification

- More on kernels: How to tell if a function is a kernel?
- SVM classification
- SVM regression

The "kernel trick"

- Recall: kernel functions are ways of expressing dot-products in some feature space:

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

- In many cases, K can be computed in time that depends on the size of the inputs $\mathbf{x}$ not the size of the feature space $\phi(\mathbf{x})$
- If we work with a "dual" representation of the learning algorithm, we do not actually have to compute the feature mapping ϕ. We just have to compute the similarity K.

Polynomial kernels

- More generally, $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^d$ is a kernel, for any positive integer d:

$$K(\mathbf{x}, \mathbf{z}) = \left(\sum_{i=1}^{n} x_i z_i \right)^d$$

- If we expanded the sum above in the obvious way, we get n^d terms (i.e. feature expansion)
- Terms are monomials (products of x_i) with total power equal to d.
- *Curse of dimensionality:* it is very expensive both to optimize and to predict in primal form
- However, *evaluating the dot-product of any two feature vectors can be done using K in $O(n)$*!

Some other (fairly generic) kernel functions

- $K(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x} \cdot \mathbf{z})^d$ – feature expansion has all monomial terms of $\leq d$ total power.
- Radial basis/Gaussian kernel:

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

  The kernel has an infinite-dimensional feature expansion, but dot-products can still be computed in $O(n)$!
- Sigmoidal kernel:

$$K(\mathbf{x}, \mathbf{z}) = \tanh(c_1 \mathbf{x} \cdot \mathbf{z} + c_2)$$

Recall: Dual-view regression

- By re-writing the parameter vector as a linear combination of instances and solving, we get:

$$\mathbf{a} = (\mathbf{K} + \lambda \mathbf{I}_m)^{-1}\mathbf{y}$$

- The *feature mapping is not needed* either to learn or to make predictions!
- This approach is useful if the feature space is very large

Making predictions in the dual view

- For a new input $\mathbf{x}$, the prediction is:

$$h(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) = \mathbf{a}^\top \mathbf{\Phi} \phi(\mathbf{x}) = \mathbf{k}(\mathbf{x})^\top (\mathbf{K} + \lambda \mathbf{I}_m)^{-1} \mathbf{y}$$

  where $\mathbf{k}(\mathbf{x})$ is an m-dimensional vector, with the ith element equal to $K(\mathbf{x}, \mathbf{x}_i)$

- That is, the ith element has the similarity of the input to the ith instance
- The features are not needed for this step either!
- This is a *non-parametric* representation - its size scales with the number of instances.

Kernels

- A lot of current research has to do with defining new kernels functions, suitable to particular tasks / kinds of input objects
- Many kernels are available:
  - Information diffusion kernels (Lafferty and Lebanon, 2002)
  - Diffusion kernels on graphs (Kondor and Jebara 2003)
  - String kernels for text classification (Lodhi et al, 2002)
  - String kernels for protein classification (e.g., Leslie et al, 2002)

  ... and others!

Example: String kernels

- Very important for DNA matching, text classification, ...
- Example: in DNA matching, we use a sliding window of length k over the two strings that we want to compare
- The window is of a given size, and inside we can do various things:
  - Count exact matches
  - Weigh mismatches based on how bad they are
  - Count certain markers, e.g. AGT
- The kernel is the sum of these similarities over the two sequences
- How do we prove this is a kernel?

Establishing "kernelhood"

- Suppose someone hands you a function K. How do you know that it is a kernel?
- More precisely, given a function $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, under what conditions can $K(\mathbf{x}, \mathbf{z})$ be written as a dot product $\phi(\mathbf{x}) \cdot \phi(\mathbf{z})$ for some feature mapping ϕ?
- We want a general recipe, which does not require explicitly defining ϕ every time

Kernel matrix

- Suppose we have an arbitrary set of input vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_m$
- The *kernel matrix (or Gram matrix)* $\mathbf{K}$ corresponding to kernel function K is an $m \times m$ matrix such that $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ (notation is overloaded on purpose).
- What properties does the kernel matrix $\mathbf{K}$ have?
- Claims:
  1. $\mathbf{K}$ is symmetric
  2. $\mathbf{K}$ is positive semidefinite
- Note that these claims are consistent with the intuition that K is a "similarity" measure (and will be true regardless of the data)

Proving the first claim

If K is a valid kernel, then the kernel matrix is symmetric

$$\mathbf{K}_{ij} = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) = \phi(\mathbf{x}_j) \cdot \phi(\mathbf{x}_i) = \mathbf{K}_{ji}$$

Proving the second claim

If K is a valid kernel, then the kernel matrix is positive semidefinite

Proof: Consider an arbitrary vector $\mathbf{z}$

$$\begin{aligned}
\mathbf{z}^\top \mathbf{K} \mathbf{z} &= \sum_i \sum_j z_i K_{ij} z_j = \sum_i \sum_j z_i \left(\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)\right) z_j \\
&= \sum_i \sum_j z_i \left(\sum_k \phi_k(\mathbf{x}_i) \phi_k(\mathbf{x}_j) \right) z_j \\
&= \sum_k \sum_i \sum_j z_i \phi_k(\mathbf{x}_i) \phi_k(\mathbf{x}_j) z_j \\
&= \sum_k \left(\sum_i z_i \phi_k(\mathbf{x}_i) \right)^2 \geq 0
\end{aligned}$$

Mercer's theorem

- We have shown that if K is a kernel function, then for any data set, the corresponding kernel matrix $\mathbf{K}$ defined by $\mathbf{K}_{ij} = K(\mathbf{x_i}, \mathbf{x_j})$ is symmetric and positive semidefinite
- Mercer's theorem states that the reverse is also true:
  Given a function $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, *$K$ is a kernel if and only if, for any data set, the corresponding kernel matrix is symmetric and positive semidefinite*
- The reverse direction of the proof is much harder (see e.g. Vapnik's book for details)
- This result gives us a way to check if a given function is a kernel, by checking these two properties of its kernel matrix.
- Kernels can also be obtained by combining other kernels, or by learning from data
- Kernel learning may suffer from overfitting (kernel matrix close to diagonal)

More on RKHS

- Mercer's theorem tells us that a function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel (i.e. it corresponds to the inner product in some feature space) if and only if it is positive semi-definite.
- The feature space is the reproducing kernel Hilbert space (RKHS)

$$\mathcal{H} = \left\{ \sum_j \alpha_j K(z_j, \cdot) \ : \ z_j \in \mathcal{X}, \ \alpha_j \in \mathbb{R} \right\}$$

  with inner product $\langle K(z, \cdot), K(z', \cdot) \rangle_{\mathcal{H}} = K(z, z')$.
- The term reproducing comes from the reproducing property of the kernel function:

$$\forall f \in \mathcal{H}, \ x \in \mathcal{X} \ : \quad f(x) = \langle f(\cdot), K(x, \cdot) \rangle_{\mathcal{H}}$$

- Recall that the solution of the regularized least square in the feature space associated to a kernel function K has the form $h(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i K(\mathbf{x}_i, \mathbf{x})$. This is a particular case of the representer theorem...

Representer Theorem

Theorem 1. *Let $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a positive definite kernel and let $\mathcal{H}$ be the corresponding RKHS.*

Then for any training sample $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^m \subset \mathcal{X} \times \mathbb{R}$, any loss function $\ell : (\mathcal{X} \times \mathbb{R} \times \mathbb{R})^m \to \mathbb{R}$ and any real-valued non-decreasing function g, the solution of the optimization problem

$$\arg\min_{f \in \mathcal{H}} \ell\left((x_1, y_1, f(x_1)), \cdots, (x_m, y_m, f(x_m))\right) + g(\|f\|_{\mathcal{H}})$$

admits a representation of the form

$$f^*(\cdot) = \sum_{i=1}^m \alpha_i K(\mathbf{x}_i, \cdot).$$

[Schlkopf, Herbrich and Smola. *A generalized representer Theorem.* COLT 2001.]

Regularization in the dual view

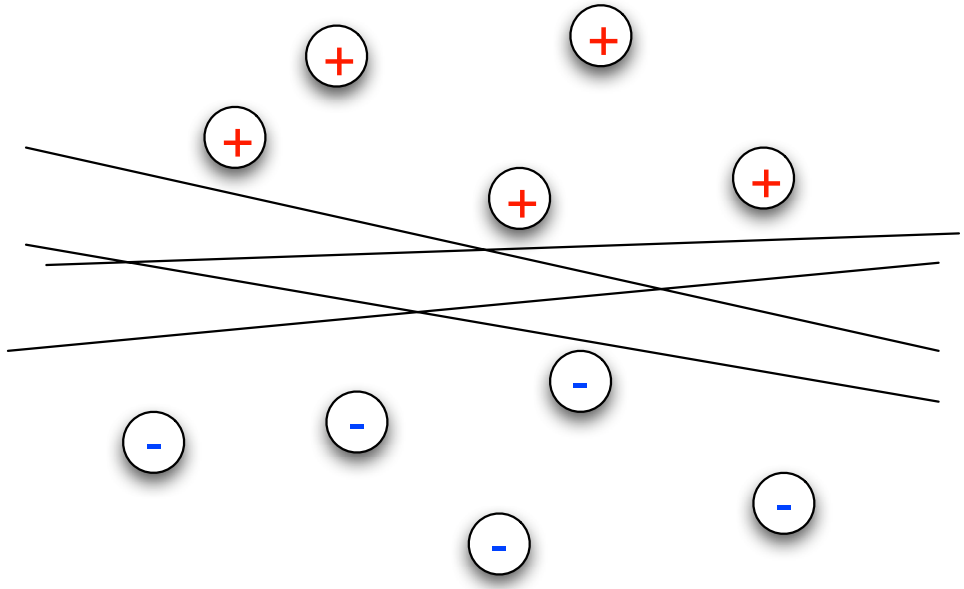
- We want to penalize the function we are trying to estimate (to keep it simple)
- Assume this is part of a reproducing kernel Hilbert space
- We want to minimize:

$$J(h) = \frac{1}{2}\sum_{i=1}^{n}(y_i - h(\mathbf{x}_i))^2 + \frac{\lambda}{2}||h||^2_{\mathcal{H}}$$

- If we put a Gaussian prior on h, and solve, we obtain *Gaussian process regression*

Binary classification revisited

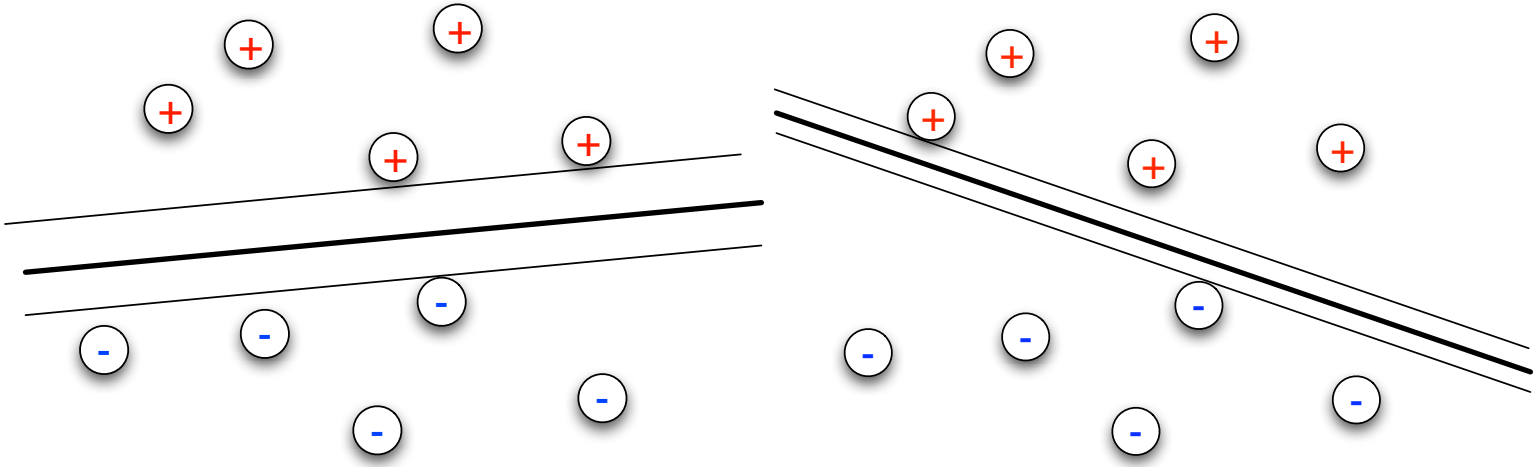
- Consider a linearly separable binary classification data set $\{\mathbf{x}_i, y_i\}_{i=1}^m$.
- There is an infinite number of hyperplanes that separate the classes:



- Which plane is best?
- Relatedly, for a given plane, for which points should we be most confident in the classification?

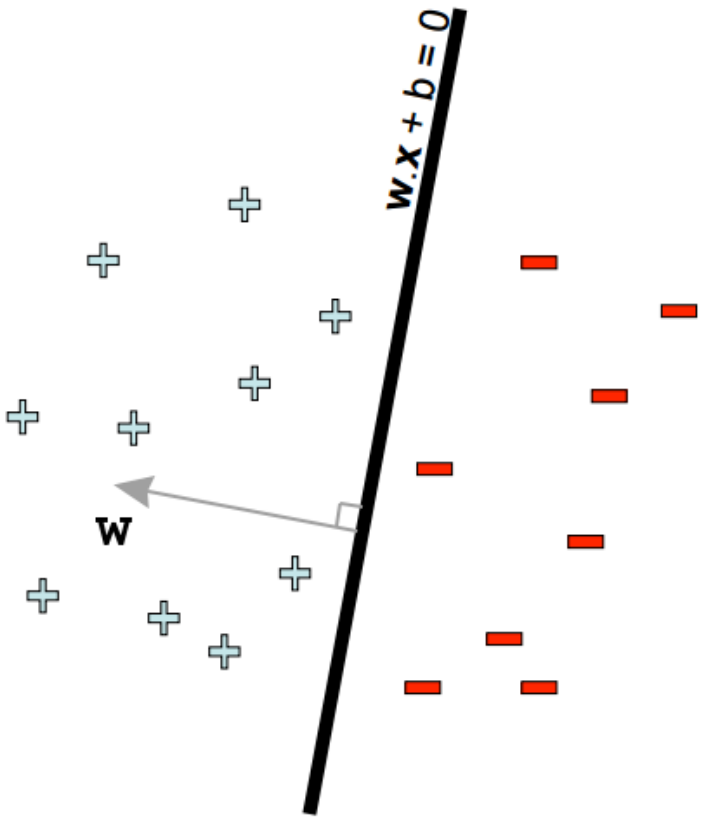
The margin, and linear SVMs

- For a given separating hyperplane, the *margin* is two times the (Euclidean) distance from the hyperplane to the nearest training example.



- It is the width of the "strip" around the decision boundary containing no training examples.
- A linear SVM is a perceptron for which we choose $\mathbf{w}, w_0$ so that margin is maximized

Hyperplanes and scale invariance

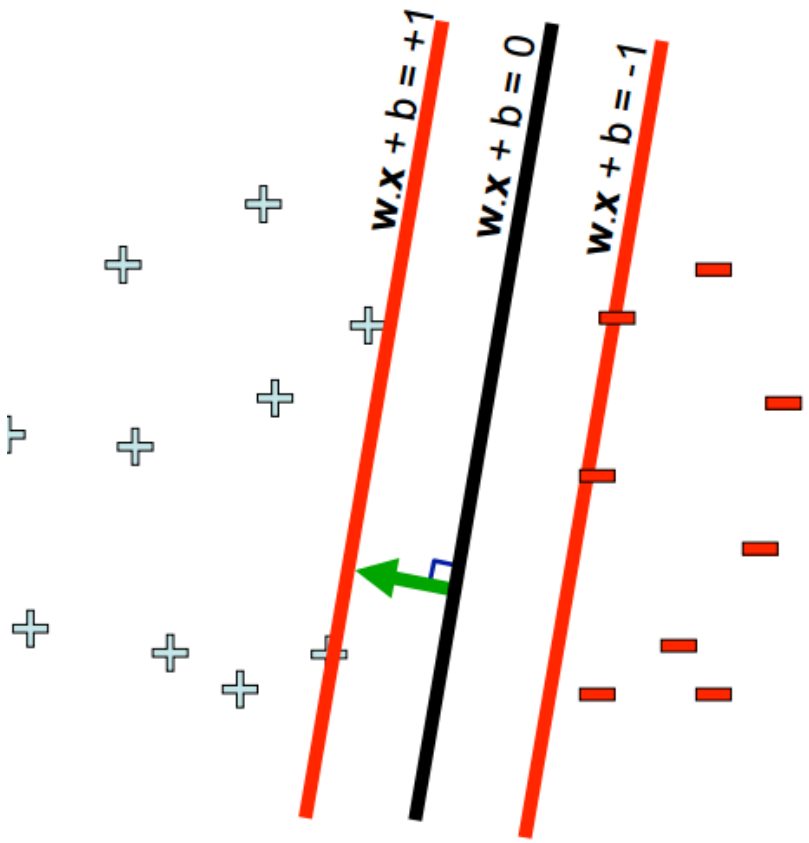


- Hyperplane: $\mathbf{w}^\top \mathbf{x} + b = 0$.

$\Rightarrow$ Scale invariance: $(\alpha \mathbf{w})^\top \mathbf{x} + \alpha b = 0$ define the same hyperplane for any scalar $\alpha \in \mathbb{R}$.

Figures taken form David Sontag's lecture slides:
`http://people.csail.mit.edu/dsontag/courses/ml13/slides/lecture3.pdf`

Hyperplanes and scale invariance



- Hyperplane: $\mathbf{w}^\top \mathbf{x} + b = 0$.

$\Rightarrow$ Scale invariance: $\alpha \mathbf{w}^\top \mathbf{x} + \alpha b = 0$...

- Workaround: we set the scale by enforcing

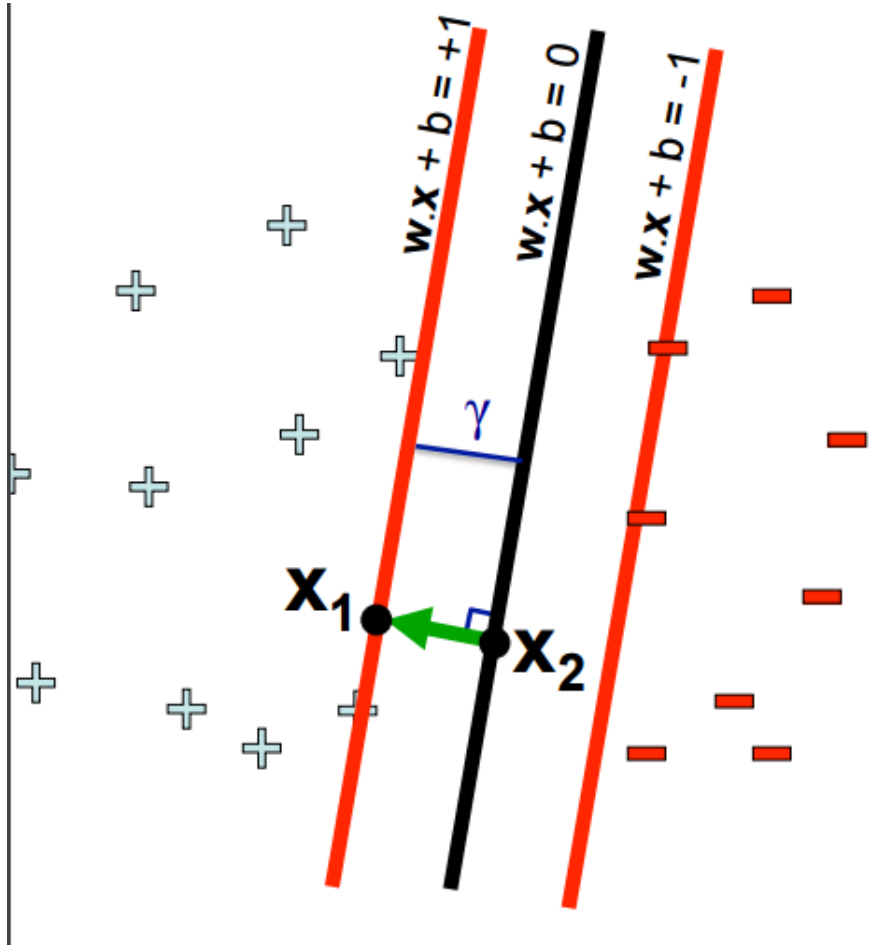
$$\mathbf{w}^\top \mathbf{x} + b \geq 1 \quad \text{for all positive examples}$$
$$\mathbf{w}^\top \mathbf{x} + b \leq -1 \quad \text{for all negative examples}$$

$\rightarrow$ equivalently, we want to satisfy the linear constraints

$$y_i(\mathbf{w}^\top \mathbf{x} + b) \geq 1$$

for all i (here $\mathcal{Y} = \{-1, 1\}$).

Margin as a function of the normal vector



- Let $\mathbf{x}_1$ be a point on the boundary and $\mathbf{x}_2$ its projection on the hyperplane:

$$\mathbf{w}^\top \mathbf{x}_1 + b = 1$$
$$\mathbf{w}^\top \mathbf{x}_2 + b = 0$$

  We get $\mathbf{w}^\top(\mathbf{x}_1 - \mathbf{x}_2) = 1$.

- But we also know that

$$\mathbf{x}_1 - \mathbf{x}_2 = \gamma \frac{\mathbf{w}}{\|\mathbf{w}\|}$$

- Hence $\gamma = \frac{1}{\|\mathbf{w}\|}$: maximizing the margin is equivalent to minimizing the norm of $\mathbf{w}$!

SVM final formulation

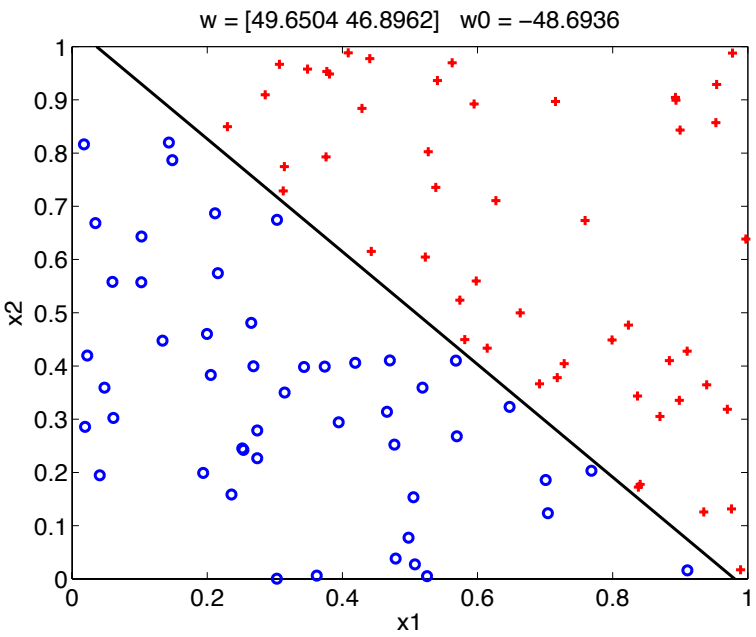
- Let's minimize $\|\mathbf{w}\|^2$ instead of $\|\mathbf{w}\|$.
  (it will be easier and taking the square is a monotone transformation, as $\|\mathbf{w}\|$ is postive, so this doesn't change the optimal solution.)
- This gets us to:

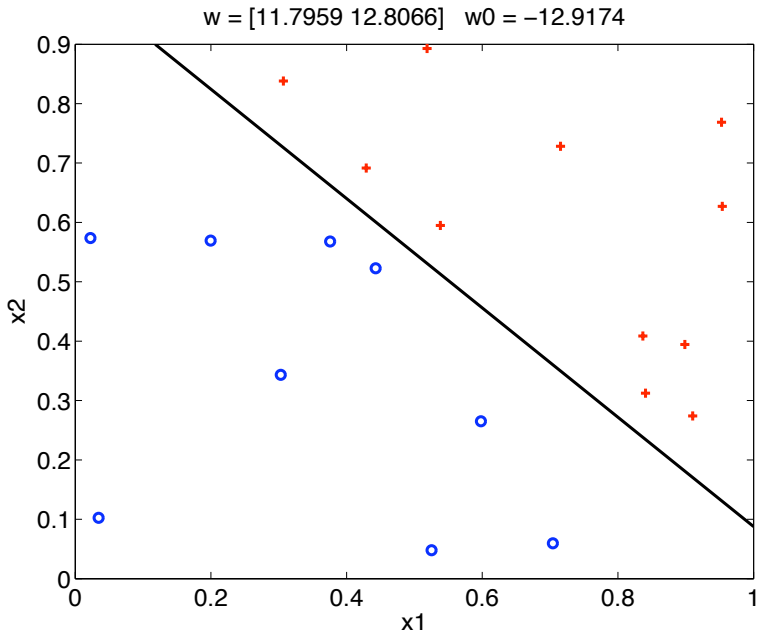
$$\begin{aligned} \text{min} \quad & \|\mathbf{w}\|^2 \\ \text{w.r.t.} \quad & \mathbf{w}, w_0 \\ \text{s.t.} \quad & y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 \end{aligned}$$

- This we can solve! How?
  - It is a *quadratic programming* (QP) problem—a standard type of optimization problem[1] for which many efficient packages are available.
  - Better yet, it's a convex (positive semidefinite) QP

[1] optimization of a quadratic function of several variables subject to linear constraints on these variables.

Example





We have a solution, but no support vectors yet...

Lagrange multipliers for inequality constraints (revisited)

- Suppose we have the following optimization problem, called *primal*:

$$\begin{aligned} &\min_{\mathbf{w}} f(\mathbf{w}) \\ \text{such that } &g_i(\mathbf{w}) \quad \leq \quad 0, \; i = 1 \dots k \end{aligned}$$

- We define the *Lagrangian*:

$$L(\mathbf{w}, \boldsymbol{\alpha}) = f(\mathbf{w}) + \sum_{i=1}^{k} \alpha_i g_i(\mathbf{w}), \tag{1}$$

where $\alpha_i, \; i = 1 \dots k$ are the Lagrange multipliers.

A different optimization problem

$$L(\mathbf{w}, \boldsymbol{\alpha}) = f(\mathbf{w}) + \sum_{i=1}^{k} \alpha_i g_i(\mathbf{w})$$

- Consider $\mathcal{P}(\mathbf{w}) = \max_{\alpha:\alpha_i \geq 0} L(\mathbf{w}, \alpha)$
- Observe that the following is true:

$$\mathcal{P}(\mathbf{w}) = \begin{cases} f(\mathbf{w}) & \text{if all constraints are satisfied} \\ +\infty & \text{otherwise} \end{cases}$$

- Hence, instead of computing $\min_{\mathbf{w}} f(\mathbf{w})$ subject to the original constraints, we can compute:

$$p^* = \min_{\mathbf{w}} \mathcal{P}(\mathbf{w}) = \min_{\mathbf{w}} \max_{\alpha:\alpha_i \geq 0} L(\mathbf{w}, \alpha)$$

Dual optimization problem

- We defined the primal problem:

$$p^* = \min_{\mathbf{w}} \max_{\alpha:\alpha_i \geq 0} L(\mathbf{w}, \alpha)$$

- The dual problem is obtained by swapping $\min$ and $\max$:

$$d^* = \max_{\alpha:\alpha_i \geq 0} \min_{\mathbf{w}} L(\mathbf{w}, \alpha)$$

- We can show that $d^* \leq p^*$:
  - Let $p^* = L(w^p, \alpha^p)$, let $d^* = L(w^d, \alpha^d)$.
  - Then $d^* = L(w^d, \alpha^d) \leq L(w^p, \alpha^d) \leq L(w^p, \alpha^p) = p^*$.
- The non-negative number $p^* - d^*$ is called the duality gap.

Dual optimization problem (cont'd)

- Slater's condition: if f (objective) and the g_i's (constraints) are convex, and the g_i's can all be satisfied simultaneously for some $\mathbf{w}$, then we have equality: $d^* = p^* = L(\mathbf{w}^*, \alpha^*)$.

$\rightarrow$ Which means that the two problems are equivalent!

- Moreover $\mathbf{w}^*, \alpha^*$ solve the primal and dual if and only if they satisfy the following conditions (called Karush-Kunh-Tucker):

$$\frac{\partial}{\partial w_i} L(\mathbf{w}^*, \alpha^*) = 0, \; i = 1 \ldots n \quad (2)$$

$$\alpha_i^* g_i(\mathbf{w}^*) = 0, \; i = 1 \ldots k \quad (3)$$

$$g_i(\mathbf{w}^*) \leq 0, \; i = 1 \ldots k \quad (4)$$

$$\alpha_i^* \geq 0, \; i = 1 \ldots k \quad (5)$$

Back to maximum margin perceptron

- We wanted to solve (rewritten slightly):

$$\begin{aligned} \text{min} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 \\ \text{w.r.t.} \quad & \mathbf{w}, w_0 \\ \text{s.t.} \quad & 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \leq 0 \end{aligned}$$

- The Lagrangian is:

$$L(\mathbf{w}, w_0, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 + \sum_i \alpha_i (1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0))$$

- The primal problem is: $\min_{\mathbf{w}, w_0} \max_{\alpha : \alpha_i \geq 0} L(\mathbf{w}, w_0, \alpha)$
- We will solve the dual problem: $\max_{\alpha : \alpha_i \geq 0} \min_{\mathbf{w}, w_0} L(\mathbf{w}, w_0, \alpha)$
- In this case, the optimal solutions coincide, because we have a quadratic objective and linear constraints (both of which are convex).

Solving the dual

- Lagrangian: $L(\mathbf{w}, w_0, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 + \sum_i \alpha_i(1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0))$
- From KKT (2), the derivatives of $L(\mathbf{w}, w_0, \alpha)$ wrt $\mathbf{w}, w_0$ should be 0:

$$\frac{\partial L}{\partial w_0} = -\sum_i \alpha_i y_i \quad \Rightarrow \quad \sum_i \alpha_i y_i = 0$$

$$\nabla_{\mathbf{w}} L = \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i \quad \Rightarrow \quad \mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

- Observe that $\alpha_i > 0$ implies that the corresponding constraint is tight, i.e. $y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) = 1$, from which we obtain

$$w_0 = y_i - \mathbf{w} \cdot \mathbf{x}_i \qquad \text{for all } i \text{ such that } \alpha_i > 0.$$

$\Rightarrow$ Just like for the perceptron with zero initial weights, the optimal solution for $\mathbf{w}$ is a linear combination of the $\mathbf{x}_i$.

- The output is

$$h_{\mathbf{w},w_0}(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{m} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) + w_0\right)$$

$\Rightarrow$ Output depends on weighted dot product of input vector with training examples

Solving the dual (cont'd)

- By plugging these back into the expression for L, we get:

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

  with constraints: $\alpha_i \geq 0$ and $\sum_i \alpha_i y_i = 0$

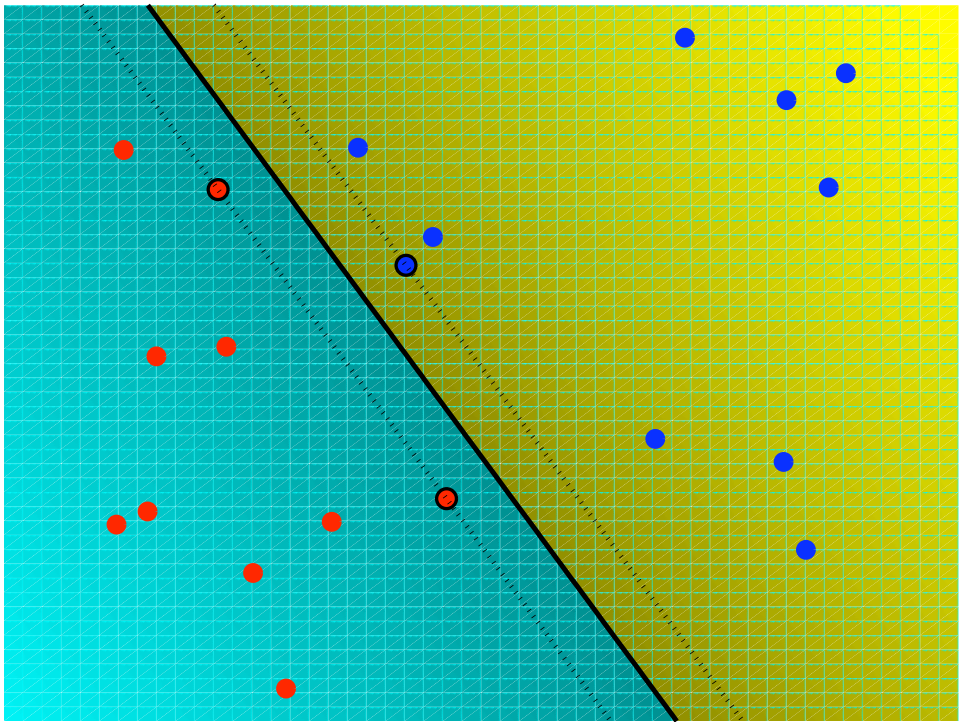
The support vectors

- Suppose we find optimal αs (e.g., using a standard QP package)
- The α_i will be > 0 only for the points for which $1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) = 0$
- These are the points lying on the edge of the margin, and they are called *support vectors*, because they define the decision boundary
- The output of the classifier for query point $\mathbf{x}$ is computed as:

$$\text{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) + w_0\right)$$

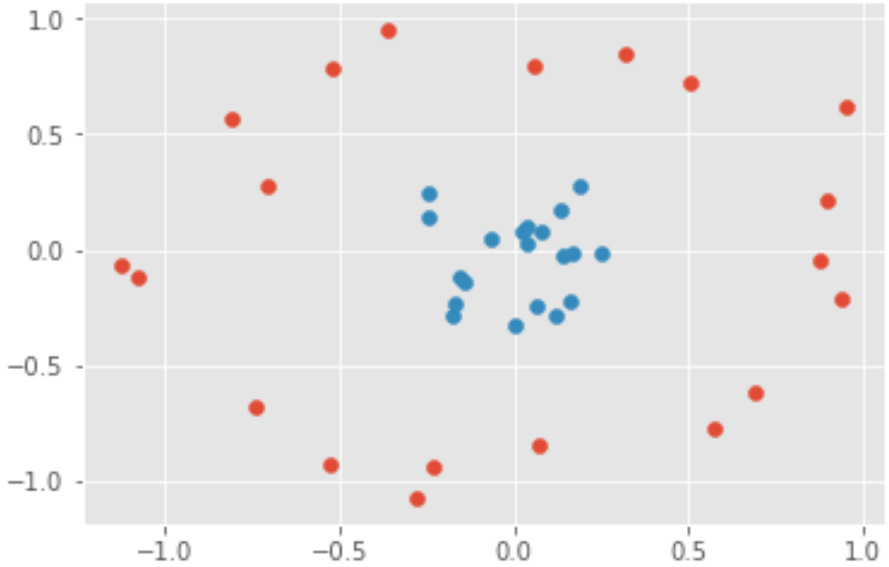
Hence, the output is determined by computing the *dot product of the point with the support vectors*!

Example



Support vectors are in bold

Non-linearly separable data



- A linear boundary might be too simple to capture the class structure.
- One way of getting a nonlinear decision boundary in the input space is to find a linear decision boundary in an expanded space (e.g., for polynomial regression.)
- Thus, $\mathbf{x}_i$ is replaced by $\phi(\mathbf{x}_i)$, where ϕ is called a *feature mapping*

Margin optimization in feature space

- Replacing $\mathbf{x}_i$ with $\phi(\mathbf{x}_i)$, the optimization problem to find $\mathbf{w}$ and w_0 becomes:

$$\begin{aligned} \text{min} \quad & \|\mathbf{w}\|^2 \\ \text{w.r.t.} \quad & \mathbf{w}, w_0 \\ \text{s.t.} \quad & y_i(\mathbf{w} \cdot \phi(\mathbf{x}_i) + w_0) \geq 1 \end{aligned}$$

- Dual form:

$$\begin{aligned} \text{max} \quad & \textstyle\sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y_i y_j \alpha_i \alpha_j \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) \\ \text{w.r.t.} \quad & \alpha_i \\ \text{s.t.} \quad & 0 \leq \alpha_i \\ & \textstyle\sum_{i=1}^m \alpha_i y_i = 0 \end{aligned}$$

$\Rightarrow$ We just need to compute $\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$ for each i, j to solve the dual problem, never need to compute the feature map explicitly...

Feature space solution

- The optimal weights, in the expanded feature space, are given by

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \phi(\mathbf{x}_i) \quad \text{and} \quad w_0 = y_j - \sum_{i=1}^{m} \alpha_i y_i \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$$

  where j is such that $\alpha_j > 0$.

- Classification of an input $\mathbf{x}$ is given by:

$$h_{\mathbf{w},w_0}(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{m} \alpha_i y_i \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) + w_0\right)$$

$\Rightarrow$ To solve the SVM optimization problem in dual form and to make a prediction, *we only need to compute dot-products of feature vectors*.

$\Rightarrow$ SVMs can straightforwardly be kernelized!

The "kernel trick"

- If we work with the dual, we do not actually have to ever compute the feature mapping ϕ. We just have to compute the similarity K.
- That is, we can solve the dual for the α_i:

$$\begin{aligned} \text{max} \quad & \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \\ \text{w.r.t.} \quad & \alpha_i \\ \text{s.t.} \quad & 0 \leq \alpha_i \\ & \sum_{i=1}^m \alpha_i y_i = 0 \end{aligned}$$

- The class of a new input $\mathbf{x}$ is computed as:

$$h_{\mathbf{w},w_0}(\mathbf{x}) = \text{sign}\left((\sum_{i=1}^m \alpha_i y_i \phi(\mathbf{x}_i)) \cdot \phi(\mathbf{x}) + w_0 \right) = \text{sign}\left(\sum_{i=1}^m \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + w_0 \right)$$

Regularization with SVMs

- Kernels are a powerful tool for allowing non-linear, complex functions
- But now the number of parameters can be as high as the number of instances!
- With a very specific, non-linear kernel, each data point may be in its own partition
- With linear and logistic regression, we used regularization to avoid overfitting
- We need a method for allowing regularization with SVMs as well.

Soft margin SVM

- Recall that in the linearly separable case, we compute the solution to the following optimization problem:

$$\begin{aligned} \text{min} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 \\ \text{w.r.t.} \quad & \mathbf{w}, w_0 \\ \text{s.t.} \quad & y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 \end{aligned}$$

- If we want to allow misclassifications, we can relax the constraints to:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 - \xi_i$$

- If $\xi_i \in (0, 1)$, the data point is within the margin
- If $\xi_i \geq 1$, then the data point is misclassified
- We define the *soft error* as $\sum_i \xi_i$
- We will have to change the criterion to reflect the soft errors

New problem formulation with soft errors

Instead of:

$$\begin{array}{rl} \text{min} & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{w.r.t.} & \mathbf{w}, w_0 \\ \text{s.t.} & y_i(\mathbf{w}\cdot\mathbf{x}_i + w_0) \geq 1 \end{array}$$

we want to solve:

$$\begin{array}{rl} \text{min} & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i \\ \text{w.r.t.} & \mathbf{w}, w_0, \xi_i \\ \text{s.t.} & y_i(\mathbf{w}\cdot\mathbf{x}_i + w_0) \geq 1 - \xi_i,\ \xi_i \geq 0 \end{array}$$

- Note that soft errors include points that are misclassified, as well as points within the margin
- There is a linear penalty for both categories
- The choice of the *constant C controls overfitting*

A built-in overfitting knob

$$\begin{array}{rl} \text{min} & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i \\ \text{w.r.t.} & \mathbf{w}, w_0, \xi_i \\ \text{s.t.} & y_i(\mathbf{w}\cdot\mathbf{x}_i + w_0) \geq 1 - \xi_i \\ & \xi_i \geq 0 \end{array}$$

- If C is 0, there is no penalty for soft errors, so the focus is on maximizing the margin, even if this means more mistakes
- If C is very large, the emphasis on the soft errors will cause decreasing the margin, if this helps to classify more examples correctly.
- Internal cross-validation is a good way to choose C appropriately

Lagrangian for the new problem

- Like before, we can write a Lagrangian for the problem and then use the dual formulation to find the optimal parameters:

$$\begin{aligned} L(\mathbf{w}, w_0, \alpha, \xi, \mu) &= \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i \\ &+ \sum_i \alpha_i \left(1 - \xi_i - y_i(\mathbf{w}_i \cdot \mathbf{x}_i + w_0)\right) + \sum_i \mu_i \xi_i \end{aligned}$$

- All the previously described machinery can be used to solve this problem
- Note that in addition to α_i we have coefficients μ_i, which ensure that the errors are positive, but do not participate in the decision boundary

Soft margin optimization with kernels

- Replacing $\mathbf{x}_i$ with $\phi(\mathbf{x}_i)$, the optimization problem to find $\mathbf{w}$ and w_0 becomes:

$$\begin{array}{rl} \text{min} & \|\mathbf{w}\|^2 + C\sum_i \xi_i \\ \text{w.r.t.} & \mathbf{w}, w_0, \xi \\ \text{s.t.} & y_i(\mathbf{w}\cdot\phi(\mathbf{x}_i) + w_0) \geq (1-\xi_i) \\ & \xi_i \geq 0 \end{array}$$

- Dual form and solution have similar forms to what we described last time, but in terms of kernels

Getting SVMs to work in practice

- Two important choices:
  - Kernel (and kernel parameters)
  - Regularization parameter C
- The parameters may interact!

  E.g. for Gaussian kernel, the larger the width of the kernel, the more biased the classifier, so low C is better
- Together, these control overfitting: always do an internal parameter search, using a validation set!
- Overfitting symptoms:
  - Low margin
  - Large fraction of instances are support vectors

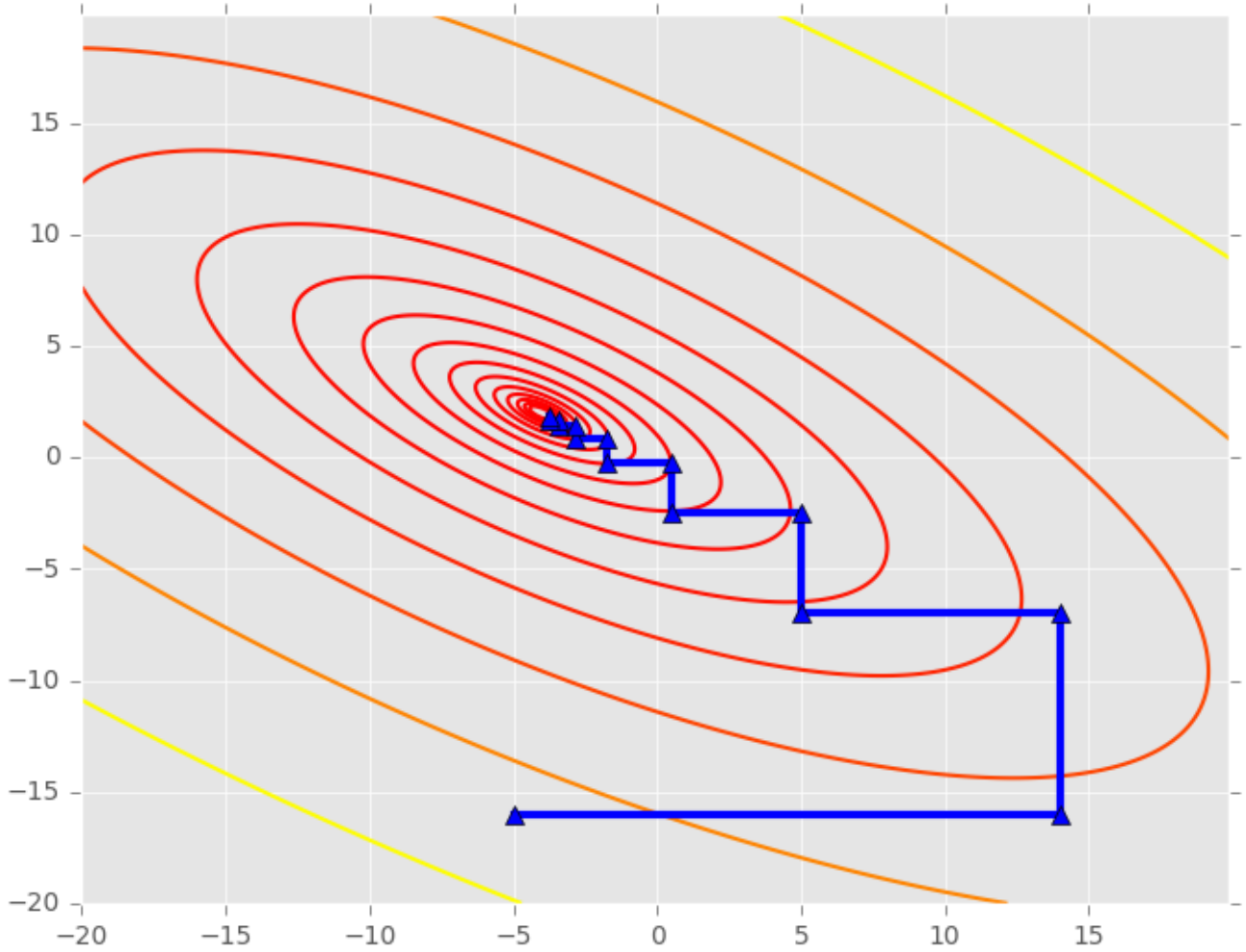
Solving the quadratic optimization problem

- Many approaches exist
- Because we have constraints, gradient descent does not apply directly (the optimum might be outside of the feasible region)
- Platt's algorithm is the fastest current approach, based on *coordinate ascent*

Coordinate ascent

- Suppose you want to find the maximum of some function $F(\alpha_1, \ldots \alpha_n)$ (the α_i could be scalars, vectors, matrices...)
- Coordinate ascent optimizes the function by repeatedly picking an α_i and optimizing it, while all other parameters are fixed:
  1. Initialize the α_i's
  2. Repeat until convergence (or stopping criterion):
     - Pick an i (round-robin, random, ...)
     - Solve $\alpha_i \leftarrow \arg\min_{\alpha_i} F(\alpha_1, \cdots, \alpha_n)$
- There are different ways of looping through the parameters:
  - Round-robin
  - Repeatedly pick a parameter at random
  - Choose next the variable expected to make the largest improvement
  - ...

Example



Our optimization problem (dual form)

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j (\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j))$$

with constraints: $0 \leq \alpha_i \leq C$ and $\sum_i \alpha_i y_i = 0$

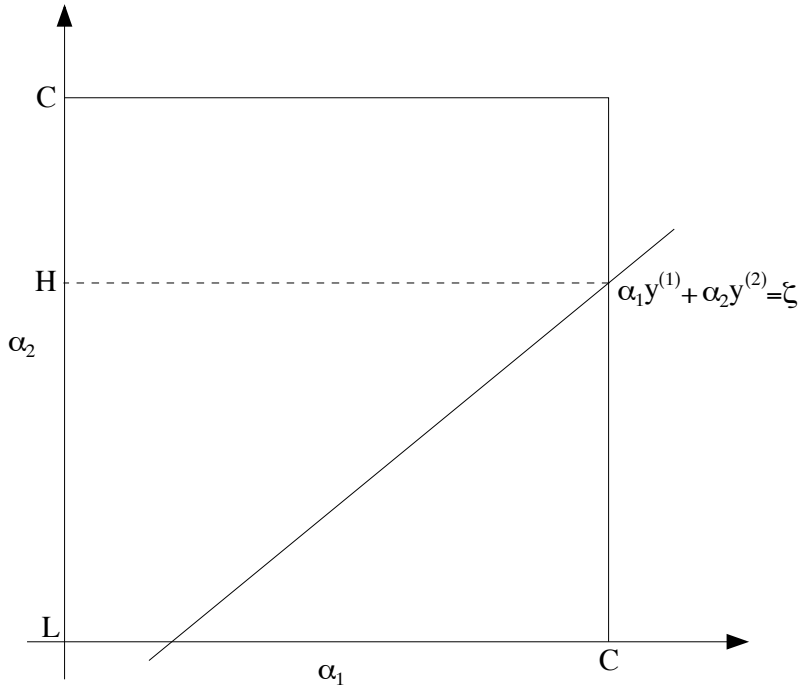
- Suppose we want to optimize for α_1 while $\alpha_2, \ldots \alpha_n$ are fixed
- We cannot do it because α_1 will be completely determined by the last constraint: $\alpha_1 = -y_1 \sum_{i=2}^{m} \alpha_i y_i$
- Instead, we have to optimize *pairs of parameters* α_i, α_j together

SMO

- Suppose that we want to optimize α_1 and α_2 together, while all other parameters are fixed.
- We know that $y_1\alpha_1 + y_2\alpha_2 = -\sum_{i=1}^{m} y_i\alpha_i = \xi$, where ξ is a constant
- So $\alpha_1 = y_1(\xi - y_2\alpha_2)$ (because y_1 is either $+1$ or -1 so $y_1^2 = 1$)
- This defines a line, and any pair α_1, α_2 which is a solution has to be on the line

SMO (II)

- We also know that $0 \leq \alpha_1 \leq C$ and $0 \leq \alpha_2 \leq C$, so the solution has to be on the line segment inside the rectangle below



SMO(III)

- By plugging α_1 back in the optimization criterion, we obtain a quadratic function of α_2, whose optimum we can find exactly
- If the optimum is inside the rectangle, we take it.
- If not, we pick the closest intersection point of the line and the rectangle
- This procedure is very fast because all these are simple computations.

Complexity

- Quadratic programming is expensive in the number of training examples
- Platt's SMO algorithm is quite fast though, and other fancy optimization approaches are available
- Best packages can handle $50,000+$ instances, but not more than $100,000$
- On the other hand, number of attributes can be very high (strength compared to neural nets)
- Evaluating a SVM is *slow if there are a lot of support vectors*.
- Dictionary methods attempt to select a subset of the data on which to train.

Support Vector Regression

- In regression problems, so far we have been trying to minimize mean-squared error:

$$\sum_i (y_i - (\mathbf{w} \cdot \mathbf{x_i} + w_0))^2$$

- In SVM regression, we will be interested instead in minimizing absolute error:

$$\sum_i |y_i - (\mathbf{w} \cdot \mathbf{x_i} + w_0)|$$

- This is more *robust to outliers* than the squared loss
- But we cannot require that all points be approximated correctly (overfitting!)

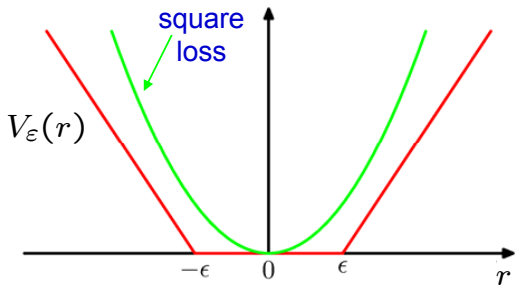
Loss function for support vector regression

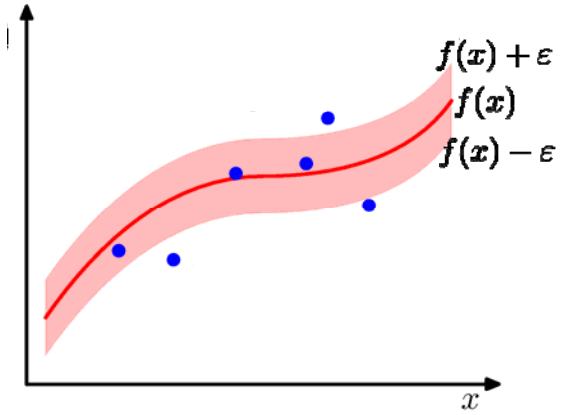
In order to allow for misclassifications in SVM regression (and have robustness to noise), we use the *ε-insensitive loss*:

$$J_\varepsilon = \sum_{i=1}^{m} J_\varepsilon(\mathbf{x_i}), \text{ where}$$

$$J_\varepsilon(\mathbf{x_i}) = \begin{cases} 0 & \text{if } |y_i - (\mathbf{w} \cdot \mathbf{x_i} + w_0)| \leq \varepsilon \\ |y_i - (\mathbf{w} \cdot \mathbf{x_i} + w_0)| - \varepsilon & \text{otherwise} \end{cases}$$

cost is zero inside epsilon "tube"





2

Solving SVM regression

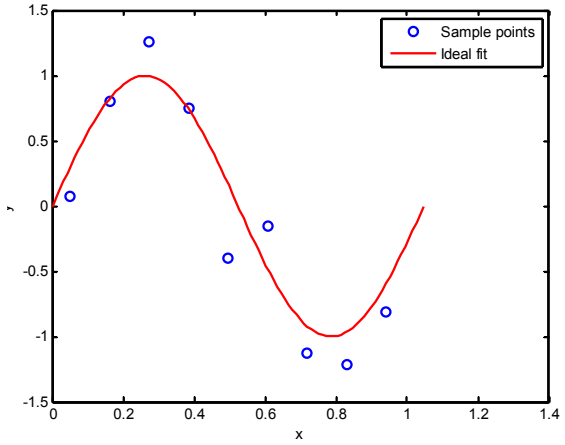
- We introduce slack variables, ξ_i^+, ξ_i^- to account for errors outside the tolerance area
- We need two kinds of variables to account for both positive and negative errors

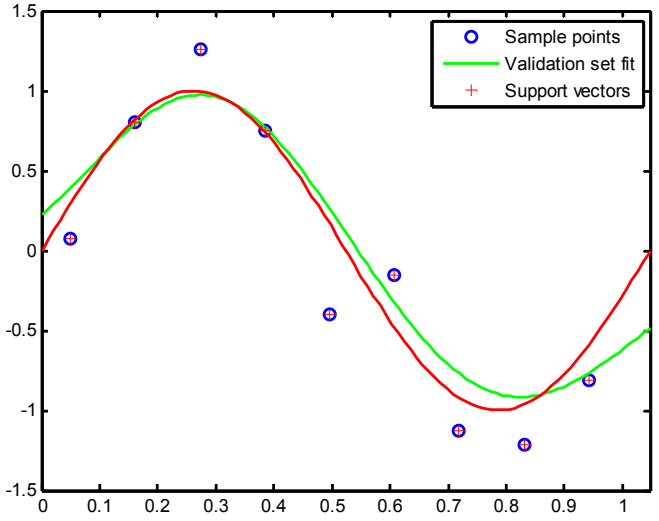
The optimization problem

$$
\begin{array}{rl}
\text{min} & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i(\xi_i^+ + \xi_i^-) \\
\text{w.r.t.} & \mathbf{w}, w_0, \xi_i^+, \xi_i^- \\
\text{s.t.} & y_i - (\mathbf{w}\cdot\mathbf{x}_i + w_0) \leq \varepsilon + \xi_i^+ \\
 & y_i - (\mathbf{w}\cdot\mathbf{x}_i + w_0) \geq -\varepsilon - \xi_i^- \\
 & \xi_i^+, \xi_i^- \geq 0
\end{array}
$$

- Like before, we can write the Lagrangian and solve the dual form of the problem
- Kernels can be used as before to get non-linear functions

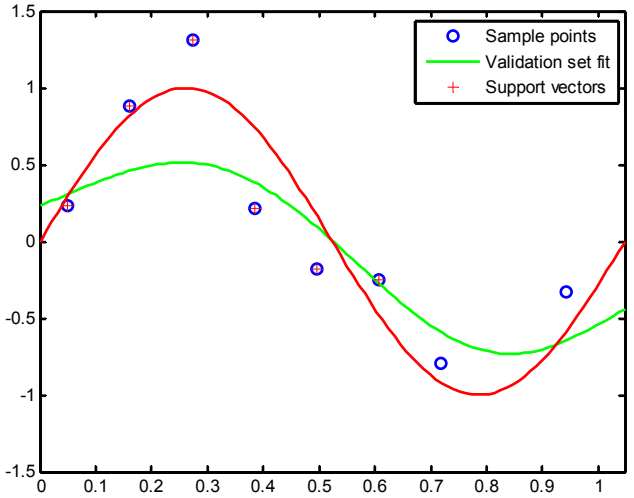
Effect of ε





epsilon = 0.01

epsilon = 0.8



[3]

- As ε increases, the function is allowed to move away from the data points, the number of support vectors decreases and the fit gets worse

[3] Zisserman course notes