Support Vector Machines

Jeremy Irvin and
Daniel Spokoyny

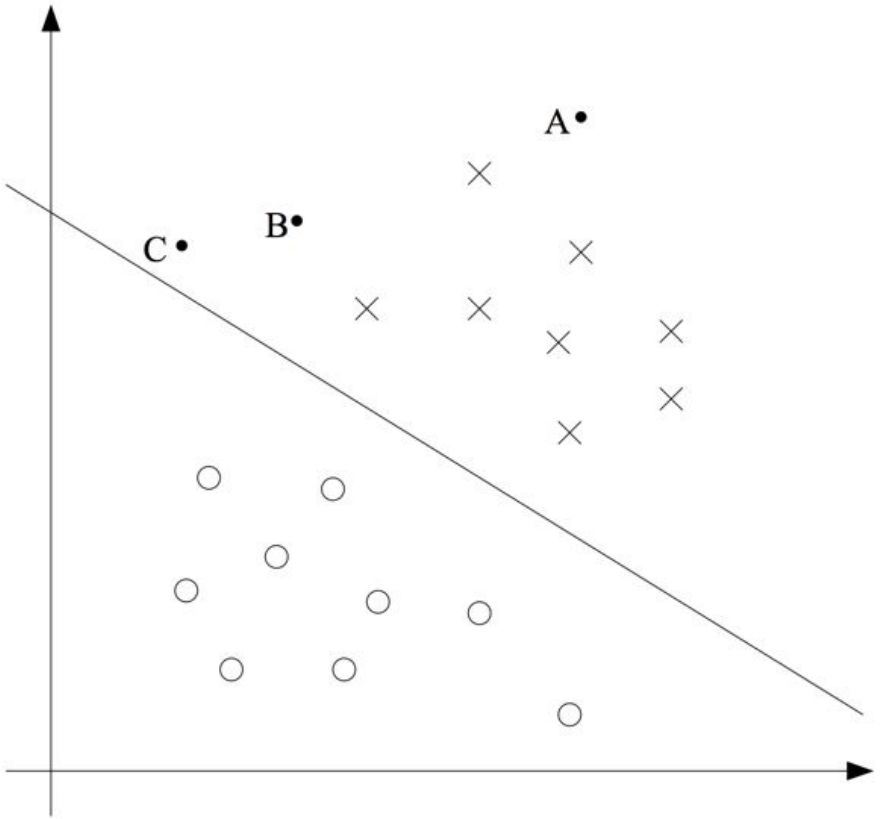
Created from Andrew Ng's
Stanford CS229 Notes

Functional Margin Intuition

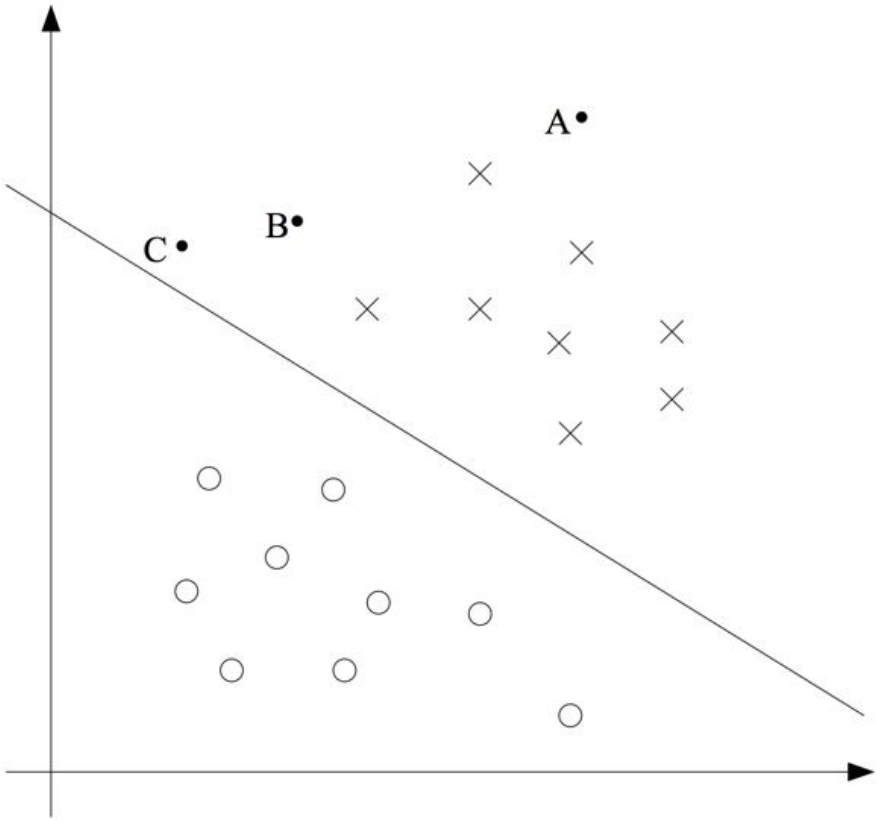
- Remember logistic regression, where $p(y = 1|x; \theta)$ is modeled by $h_\theta(x) = g(\theta^T x)$.
- We predict '1' if and only if $h_\theta(x) \geq 0.5$, or equivalently, $\theta^T x \geq 0$.
- So for a positive training example, the larger $\theta^T x$, the more 'confident' we are that the label is 1.
- Intuitively, our model is a good fit if we have learned a θ such that $\theta^T x^{(i)} \gg 0$ when $y^{(i)} = 1$ and $\theta^T x^{(i)} \ll 0$ when $y^{(i)} = 0$.

Geometric Margin Intuition

- Below, **x**'s represent positive training examples and **o**'s represent negative training examples, and the line is $\theta^T x = 0$ (the decision boundary, or separating hyperplane):



Geometric Margin Intuition



- Notice A is very far from the decision boundary - we should be very confident about our prediction for this point.
- However, C is very close - a small change to the boundary could change the prediction here.
- So we should be confident about our predictions for points far from the decision boundary.
- Intuitively, our model is a good fit the data is far from the learned decision boundary (high confidence).

Change of Notation

- We will be discussing a binary classification problem (now with $y \in \{-1, 1\}$) using a linear classifier with parameters w, b:

$$h_{w,b}(x) = g(w^T x + b)$$

- The g above is not the sigmoid function from before - it is any function satisfying

$$g(z) = \begin{cases} 1 & z \geq 0 \\ -1 & z < 0 \end{cases}$$

- Notice that this classifier will directly predict either 1 or -1, unlike logistic regression which estimated to probability of y being 1.

Functional Margin

- Given a training example $(x^{(i)}, y^{(i)})$, the functional margin of (w, b) is

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x^{(i)} + b)$$

- If $y^{(i)} = 1$, then the functional margin is large when $w^T x^{(i)} + b$ is a large positive number.

- If $y^{(i)} = -1$, then the functional margin is large when $w^T x^{(i)} + b$ is a large negative number.

- Also, our prediction is correct if $y^{(i)}(w^T x^{(i)} + b) > 0$.

- So a large functional margin means a confident and correct prediction.

Functional Margin

- There is one big problem with the functional margin that makes it a poor measure of confidence!

- For any choice of g, if we replace **w** with **2w** and **b** with **2b**, then

$$g(w^T x + b) = g(2w^T + 2b)$$

and thus our prediction $h_{w,b}(x)$ would not change at all.

- But this means we can make the functional margin arbitrarily large!

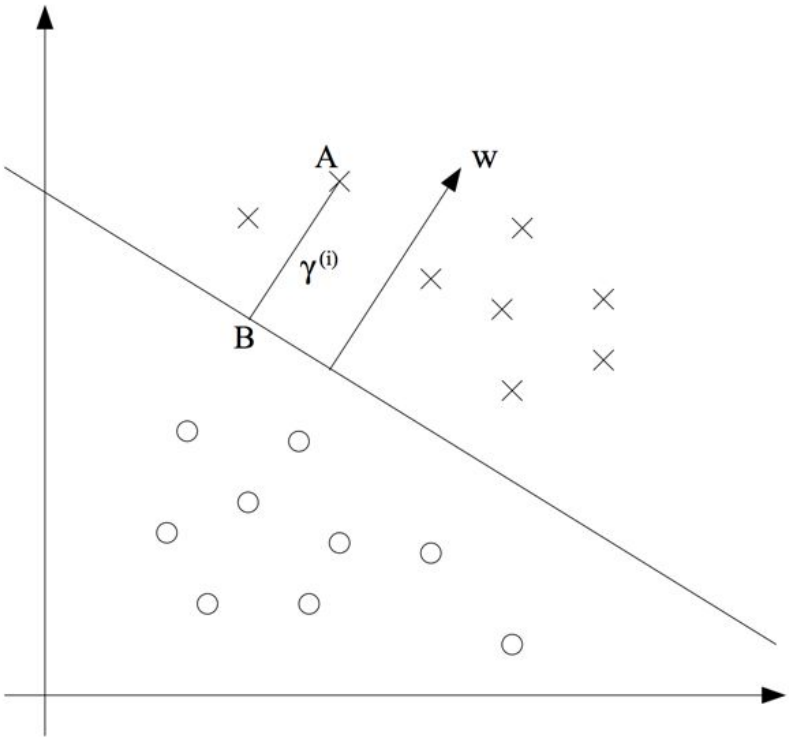
Functional Margin

- Maybe we should impose some normalization condition, like $||w||_2 = 1$ (replace (w, b) with $(w/||w||_2, b/||w||_2)$ when computing the functional margin).

- Given a training set $S = \{(x^{(i)}, y^{(i)}) : i = 1, ..., m\}$, we define the functional margin of the training set to be the smallest functional margin of each individual training example, ie,

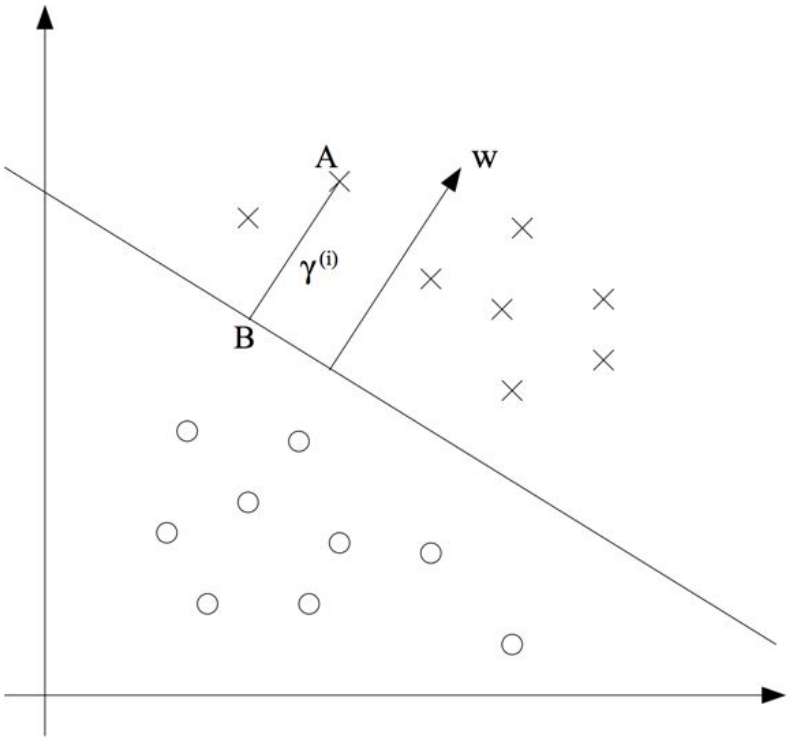
$$\hat{\gamma} = \min_{i=1,...,m} \hat{\gamma}^{(i)}$$

Geometric Margin



- The separating hyperplane corresponding to (w,b) is shown.
- The vector w is shown as well - it is orthogonal to the decision boundary - coincidence?
- A represents the input $x^{(i)}$ with label $y^{(i)} = 1$.
- The distance $\gamma^{(i)}$ to the decision boundary is the length of the line segment AB.
- How can we find $\gamma^{(i)}$?

Geometric Margin



- First note that

$$B = A - \gamma^{(i)} \frac{w}{||w||} = x^{(i)} - \gamma^{(i)} \frac{w}{||w||}$$

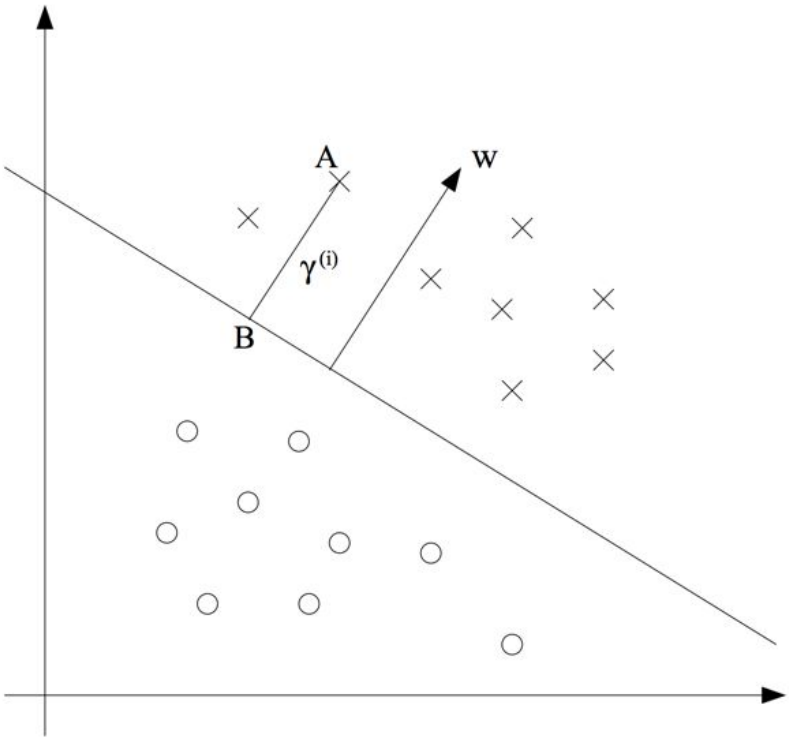
- But B lies on the decision boundary, so it satisfies $w^T B + b = 0$. Therefore

$$w^T \left(x^{(i)} - \gamma^{(i)} \frac{w}{||w||} \right) + b = 0$$

and solving for $\gamma^{(i)}$ yields

$$\gamma^{(i)} = \frac{w^T x^{(i)} + b}{||w||} = \left(\frac{w}{||w||} \right)^T x^{(i)} + \frac{b}{||w||}$$

Geometric Margin



- We can do the same thing for negative training examples to find the more general definition

$$\gamma^{(i)} = y^{(i)} \left(\left(\frac{w}{||w||} \right)^T x^{(i)} + \frac{b}{||w||} \right)$$

- If $||w|| = 1$ then this is the same as the functional margin!
- Notice that this margin is invariant to scaling of the parameters.

Geometric Margin

- This means we can choose any scaling constraint without changing the value of the margin!
- Given a training set $S = \{(x^{(i)}, y^{(i)}) : i = 1, ..., m\}$, we define the functional margin of the training set to be the smallest functional margin of each individual training example, ie,

$$\hat{\gamma} = \min_{i=1,...,m} \hat{\gamma}^{(i)}$$

Maximal Margin Classifier

- Given a training set, we hope to find a decision boundary which maximizes the (geometric) margin, since this would imply a confident set of predictions and thus a good fit to the data.

- Suppose that our training set is linearly separable (we are able to separate the positive and negative examples using a hyperplane).

- How do we find the separating hyperplane which maximizes the geometric margin?

Maximal Margin Classifier

- Formally, the problem formulation becomes

$$\begin{aligned} \max_{\gamma,w,b} \quad & \gamma \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq \gamma, \quad i = 1, \ldots, m \\ & ||w|| = 1. \end{aligned}$$

- We want to maximize γ subject to every training example having functional margin greater than or equal to γ.

$$||w|| = 1$$

- Notice that ensures that the functional margin equals the geometric margin, so this optimization problem results in parameters **(w,b)** which maximize the geometric margin of the training set.

Maximal Margin Classifier

- But solving this problem is difficult due to the non-convex $||w|| = 1$ constraint - we cannot use any standard optimization software to solve the problem in its current form.

- We can reformulate this problem as

$$\max_{\hat{\gamma},w,b} \quad \frac{\hat{\gamma}}{||w||}$$
$$\text{s.t.} \quad y^{(i)}(w^T x^{(i)} + b) \geq \hat{\gamma}, \quad i = 1, \ldots, m$$

- We want to maximize $\hat{\gamma}/||w||$ subject to every training example having functional margin greater than $\hat{\gamma}$.

- Since geometric and functional margins are related by $\gamma = \hat{\gamma}/||w||$, this yields the same result.

Maximal Margin Classifier

- But again solving this problem is difficult due to the non-convex objective function - still not standard software can solve the optimization problem in this form.

- Remember that we can add any constraint on **w** and **b** without changing the geometric margin!

- We will introduce the scaling constraint that the functional margin of **w,b** of the training set must be 1, ie:

$$\hat{\gamma} = 1$$

- Because multiplying **w** and **b** by some constant results in the functional margin multiplied by the same constant, this is just a scaling constraint - it can be satisfied by rescaling **w,b**.

Maximal Margin Classifier

- Plugging this into the reformulated problem above and noticing that maximizing $\hat{\gamma}/||w|| = 1/||w||$ is the same thing as minimizing $\frac{1}{2}||w||^2$, we have the optimization problem:

$$\begin{aligned} \min_{\gamma,w,b} \quad & \frac{1}{2}||w||^2 \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1, \ldots, m \end{aligned}$$

- This has a convex quadratic objective function and linear constraints - it can be efficiently solved using quadratic programming software!

- The solution yields the optimal (maximal) margin classifier.

Lagrange Duality

- We will not discuss the details of Lagrange duality - but it essentially allows us to reformulate this optimization problem in its dual form.

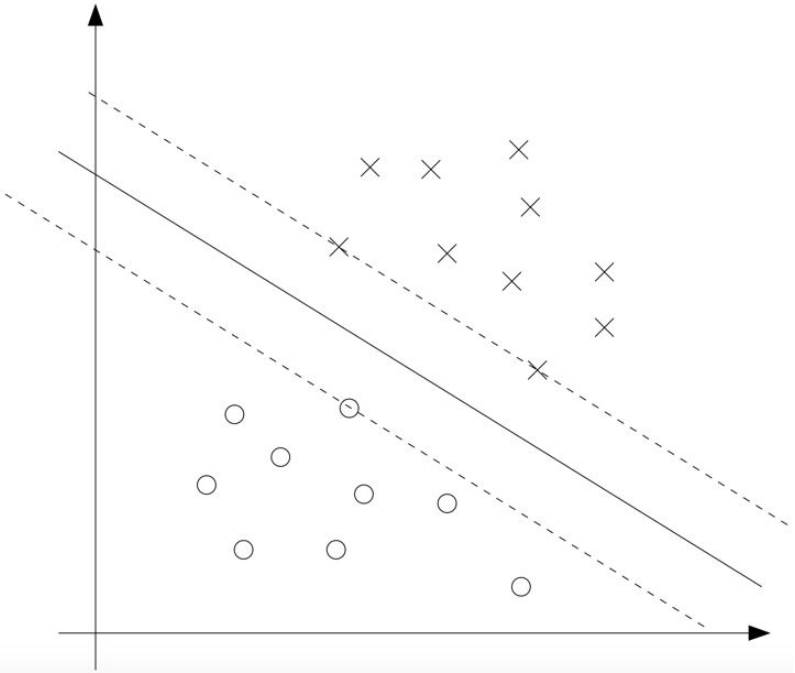
- Doing so will allow us to use kernels for efficiency in high dimensional spaces, as well as efficiency in general - much better than generic quadratic programming software.

- If you would like to learn more, the details can be found in Andrew Ng's CS229 notes!

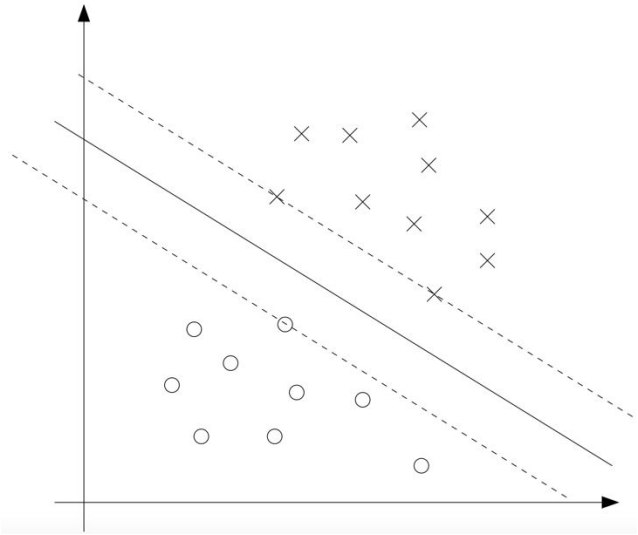
SVM Intuition

- Using Lagrange Duality, we can once again reformulate our optimization problem.

- Consider the figure below. The solid line is the maximum margin separating hyperplane:



SVM Intuition



- The three points closest to the decision boundary (on the dashed lines) are called the support vectors.
- Using Lagrange Duality, you can show that the number of support vectors can be much smaller than the training set.

SVM Intuition

- Again using Lagrange Duality, the solution to the following problem

$$\begin{aligned} \max_\alpha \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle. \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0, \end{aligned}$$

  can be used to solve the original problem.

- Moreover, this problem is written only in terms of inner products between input feature vectors.

- We can exploit this property to apply kernels to the classification problem. The resulting algorithm, called support vector machines, allows for efficient learning in very high dimensional spaces.

Kernels

- Recall the initial linear regression problem.
- We had a few attributes given, like the number of ants x, the size of the house y, etc, and we're trying to make a prediction about the house.
- We could have used slightly different variations of our features instead - $x^2, x^3, \sin x, \ldots$ - and learned a much more complex function using least squares as before.

Kernels

- The original (raw) input is called the attributes, and the attributes mapped to a new set of quantities passed to the learning algorithm are called the features.

- Let ϕ denote the feature mapping, which maps from the attributes to the features.

- For example,

$$\phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \\ \sin(x) \end{bmatrix}$$

Kernels

- In any learning algorithm, rather than directly inputting our input attributes, we may want to instead learn using the features.

- We can do this easily by replacing the attributes x everywhere with $\phi(x)$.

- Since the SVM algorithm can be written entirely in terms of inner products, we can replace all of the inner product of attribute vectors with inner products of feature vectors.

Kernels

- We define the Kernel to be

$$K(x,z) = \langle \phi(x), \phi(z) \rangle = \phi(x)^T \phi(z)$$

- So we could replace $\langle x, z \rangle$ with $K(x,z)$ and the algorithm would learn using the features rather than the attributes.

- So given ϕ, we can compute $K(x,z)$ by finding $\phi(x)$ and $\phi(z)$ and taking their inner product.

- However, it is often very inexpensive to calculate $K(x,z)$ directly from the attributes when $\phi(x)$ may be very expensive to calculate (high-dim).

Kernels

- In these situations, by using an efficient way to calculate the kernel $K(x, z)$, the algorithm (SVM) can learn in high dimensional feature space (the range of ϕ), without ever explicitly finding or representing the vectors $\phi(x)$.

- Example: $x, z \in \mathbb{R}^n, \quad K(x, z) = (x^T z)^2$

$$
\begin{aligned}
K(x,z) &= \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_i z_i\right) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j z_i z_j \\
&= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j)
\end{aligned}
$$

Kernels

$$K(x,z) = \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j)$$

- Thus we can write $K(x,z) = \phi(x)^T \phi(z)$ where ϕ is defined by (for n=3):

$$\phi(x) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_1 x_3 \\ x_2 x_1 \\ x_2 x_2 \\ x_2 x_3 \\ x_3 x_1 \\ x_3 x_2 \\ x_3 x_3 \end{bmatrix}$$

Kernels

- Intuitively, if $\phi(x)$ and $\phi(z)$ are close together, we might expect $K(x,z) = \phi(x)^T \phi(z)$ to be large.

- If they are ‘far” apart (say orthogonal) then $K(x,z) = \phi(x)^T \phi(z)$ will be small.

- So we can think of $K(x,z)$ as a similarity measure of $\phi(x)$ and $\phi(z)$ (or of x and z).

- Suppose we find some function $K(x,z)$ that we think is a good measure of similarity of x and z.

Kernels

- Maybe we choose

$$K(x,z) = e^{-\frac{||x-z||^2}{2\sigma^2}}$$

- This measure is close to *1* when *x* and *z* are close, and close to *0* when they are far apart.

- Can we use this *K* as the kernel in an SVM?

- Yes! It is called the Gaussian kernel, and corresponds to an infinite dimensional feature mapping ϕ.

Kernels

- How do we tell is a function *K* is a valid kernel (ie, that there exists some feature mapping ϕ such that $K(x,z) = \phi(x)^T \phi(z)$)?

- Suppose for now that *K* is a valid kernel, and consider some finite set of *m* points (not necessarily the training set) $\{x^{(1)}, ..., x^{(m)}\}$.

- Define a square *m*-by-*m* matrix *K* with $K_{ij} = K(x^{(i)}, x^{(j)})$. This is called the Kernel matrix.

Kernels

- If K is a valid kernel, then FIX

$$K_{ij} = K(x^{(i)}, x^{(j)}) = \langle x^{(i)}, x^{(j)} \rangle = \langle x^{(j)}, x^{(i)} \rangle = K(x^{(j)}, x^{(i)}) = K_{ji}$$

so K is symmetric.

- It can be easily shown that in fact K is positive semidefinite.

- In fact, this is also a sufficient condition:

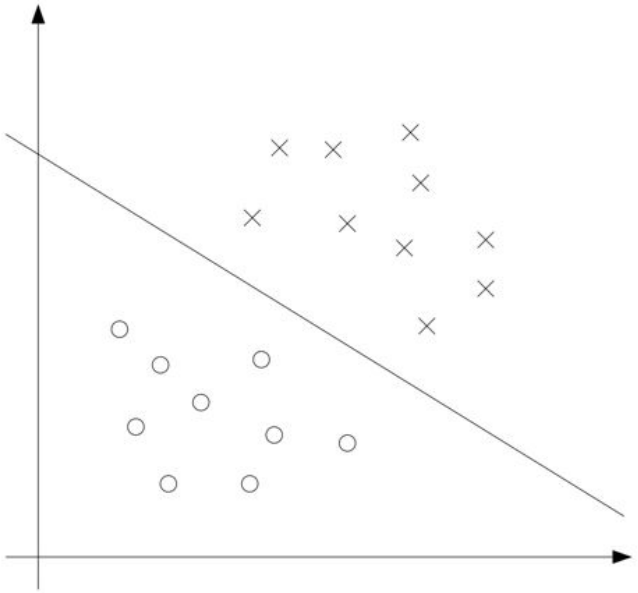
Theorem (Mercer). Let $K : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ be given. Then for K to be a valid (Mercer) kernel, it is necessary and sufficient that for any $\{x^{(1)}, \ldots, x^{(m)}\}$, $(m < \infty)$, the corresponding kernel matrix is symmetric positive semi-definite.

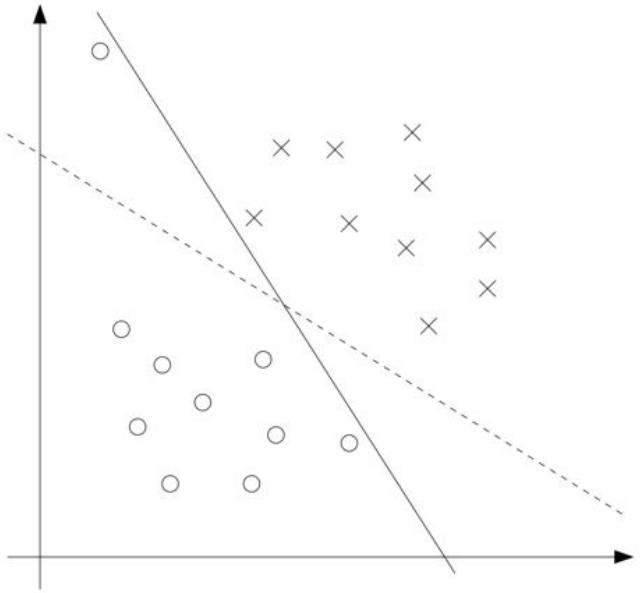
Regularization

- The SVM algorithm has thus far assumed the data is linearly separable.
- Mapping data to a high dimensional feature space via ϕ increases the likelihood the data is separable, but this is not always the case.
- In some cases it is not clear whether finding a separating hyperplane is what we want to do, since it may be susceptible to outliers.

Regularization

- For example,





- Here, the outlier causes the decision boundary to make a large rotation, causing the classifier to have a much smaller margin.

Regularization

- So to make the algorithm work for non-linearly separated datasets and simultaneously be less sensitive to outliers, we reformulate our optimization as:

$$\begin{aligned} \min_{\gamma,w,b} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m \\ & \xi_i \geq 0, \quad i = 1, \ldots, m. \end{aligned}$$

Regularization

$$\min_{\gamma,w,b} \quad \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i$$

$$\text{s.t.} \quad y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m$$

$$\xi_i \geq 0, \quad i = 1, \ldots, m.$$

- The Csum term is called ℓ_1-regularization.

- Now, the training examples can have functional margin less than 1, and if one has functional margin $1 - \xi^i, \xi^i > 0$, then we pay the cost of the objective function increased by $C\xi_i$.

- *C* controls the weighting between making the $||w||^2$ term small and ensuring the examples have functional margin at least *1*.

Regularization

- Once again we can use Lagrange duality to reformulate the problem in terms of only inner products:

$$\begin{aligned}
\max_\alpha \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\
\text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m \\
& \sum_{i=1}^{m} \alpha_i y^{(i)} = 0,
\end{aligned}$$

Coordinate Ascent

- Consider solving the *unconstrained* optimization problem:

$$\max_{\alpha} W(\alpha_1, \ldots, \alpha_m)$$

- We've seen gradient ascent as one optimization algorithm.

- Let's consider coordinate ascent:

Loop until convergence: {

For $i = 1, \ldots, m$, {

$\alpha_i := \arg\max_{\hat{\alpha}_i} W(\alpha_1, \ldots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \ldots, \alpha_m)$

}

}

Coordinate Ascent

Loop until convergence: {

For $i = 1, \ldots, m$, {

$\alpha_i := \arg\max_{\hat{\alpha}_i} W(\alpha_1, \ldots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \ldots, \alpha_m)$

}

}

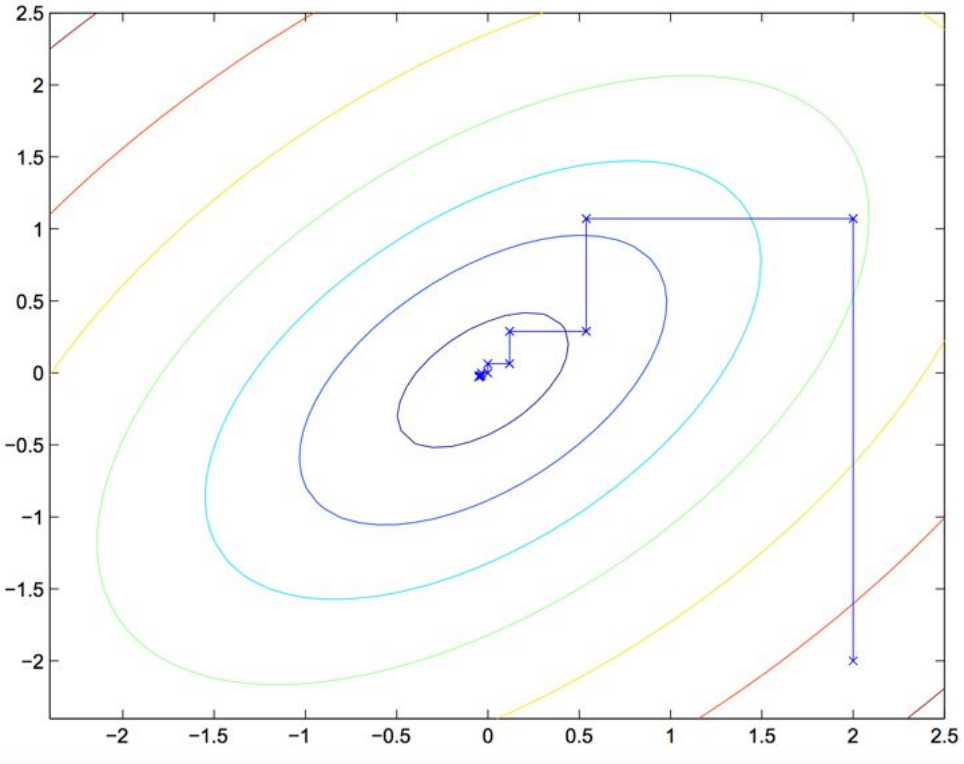
- In the innermost loop, we hold all variables except for one fixed, and reoptimize *W* with respect to that one variable.

- This algorithm reoptimizes the variables in order, but a more sophisticated algorithm may choose other orderings such as updating the variable which makes the largest increase in *W*.

Coordinate Ascent

- Coordinate ascent is fairly efficient when W is in a form that the 'argmax' in the inner loop can be performed efficiently.

- Example of coordinate ascent:



SMO Algorithm

- Recall the reformulation:

$$\begin{aligned} \max_\alpha \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0, \end{aligned}$$

- Suppose we hold all but one variable fixed and reoptimize with respect to that one variable. Can we make progress?

- No! The last variables must be fixed as well:

$$\alpha_1 y^{(1)} = -\sum_{i=2}^{m} \alpha_i y^{(i)}$$

SMO Algorithm

- Thus to update some of the variables, we must update two simultaneously. This motivates the SMO algorithm:

Repeat till convergence {

1. Select some pair α_i and α_j to update next (using a heuristic that tries to pick the two that will allow us to make the biggest progress towards the global maximum).
2. Reoptimize $W(\alpha)$ with respect to α_i and α_j, while holding all the other α_k's $(k \neq i, j)$ fixed.

}

- This is a very efficient algorithm because the update to the pair of variables can be computed very efficiently. See the notes.

What Just Happened?

- Maximal Margin Classifiers
- Kernels
- Regularization
- Coordinate Ascent and the SMO Algorithm