Machine Learning
Week 7
Support Vector Machines

This week is about Support Vector Machines. SVMs are considered by many to be the most powerful 'black box' learning algorithm, and by posing a cleverly-chosen optimization objective, one of the most widely used learning algorithms today.

Large Margin Classification

Optimization Objective

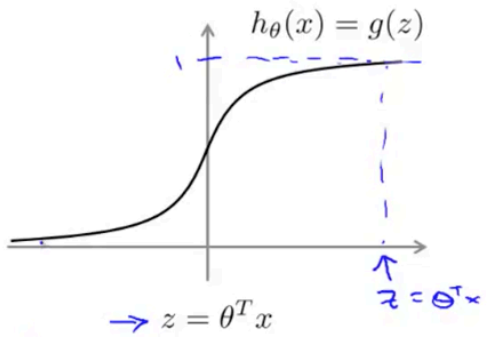
So far, for supervised learning algorithms, we have covered linear regression, logistic regression, and neural networks.

SVMs are also supervised learning algorithms.

SVM sometimes gives us a cleaner more powerful way of learning complex non-linear functions.

Before we start to understand SVM, lets first revisit logistic regression:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$



If $y = 1$, we want $h_\theta(x) \approx 1$, $\theta^T x \gg 0$

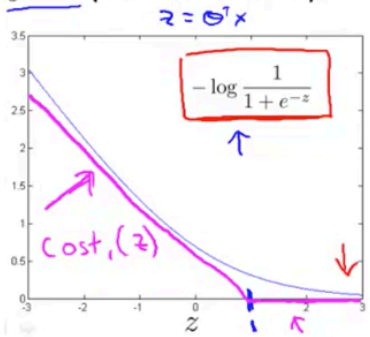
If $y = 0$, we want $h_\theta(x) \approx 0$, $\theta^T x \ll 0$

If we look at the cost function of logistic regression, each training example (x,y) contributes to our overall cost function.

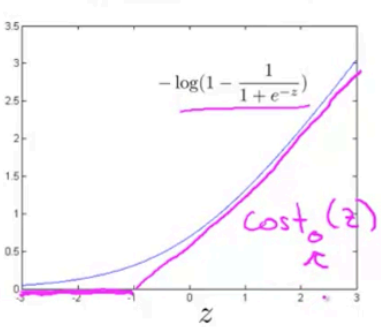
Cost of example: $-(y \log h_\theta(x) + (1-y)\log(1 - h_\theta(x)))$

$$= -y \log \frac{1}{1 + e^{-\theta^T x}} - (1-y)\log(1 - \frac{1}{1 + e^{-\theta^T x}})$$

If $y = 1$ (want $\theta^T x \gg 0$):



If $y = 0$ (want $\theta^T x \ll 0$):

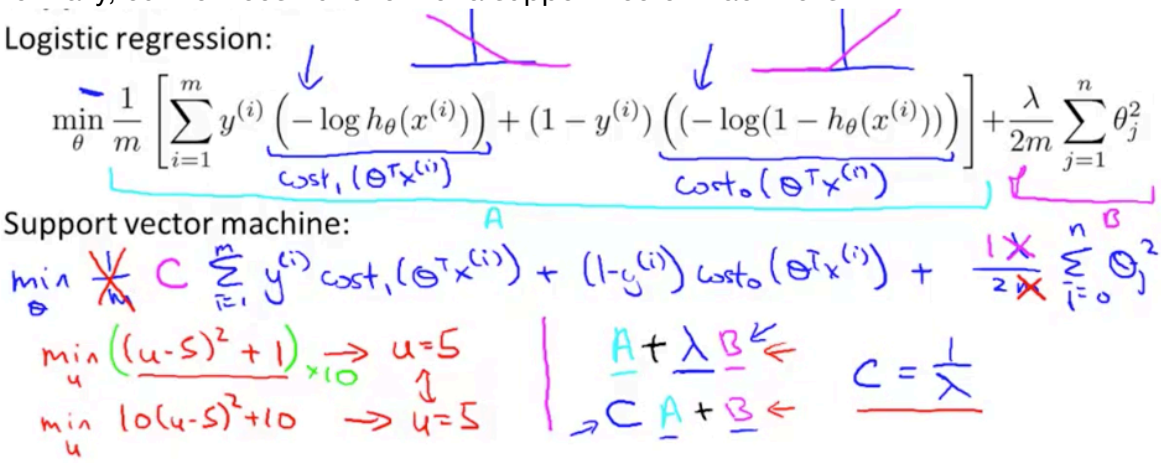


What we do for support vector machines is change the logistic regression cost function curve to be two straight line segments as shown in the magenta color.

- "cost_1_(z)"
- "cost_0_(z)"

This is to give us computational advantages that will allow for easier optimization later on.

Formally, our new cost function for a support vector machine is:



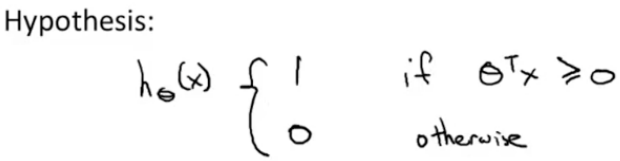
We cross out our 1/m term because multiplying the cost function by some constant does not change the optimization problem and as such we don't care about it / don't need it.

Another convention that we change for the SVM is changing from (A + lambda * B) to (C * A + B). This is just another way to control the way we optimize the first term instead of optimizing the second term.
Think of (C = 1/lambda) as it plays a similar role.

As such we have our overall cost function for a support vector machine which we want to minimize:

$$\min_\theta C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

Lastly, the support vector machine does not output a probability like logistic regression. What a support vector machine does is make a direct prediction of 1 or 0.

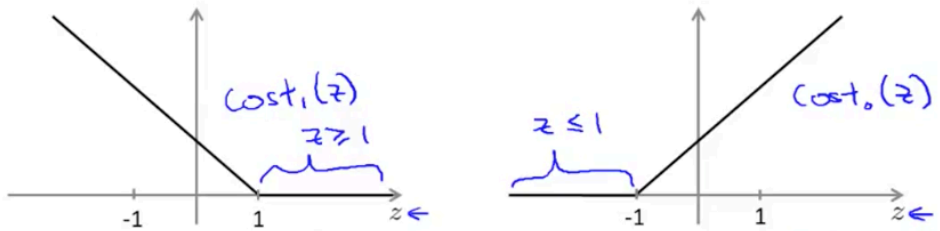


Large Margin Intuition

Sometimes the term 'Support Vector Machine' is called 'Large Margin Classifiers'.

Once again, here is our cost function for a SVM:

$$\min_\theta C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$



If $y = 1$, we want $\theta^T x \geq 1$ (not just ≥ 0)

If $y = 0$, we want $\theta^T x \leq -1$ (not just < 0)

What does it take to make our cost function small?

For positive examples, y=1, the cost is zero when z ≥ 1.
For negative examples, y=0, the cost is zero when z ≤ -1.

Because of this 'stricter' need SVM have a larger 'safety margin'.

...

What happens if we set our constant C to a very large value if we try to minimize the cost function?

SVM Decision Boundary

$$\min_\theta C\left[\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right]\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

Whenever $y^{(i)} = 1$:

$\theta^T x^{(i)} \geq 1$

Whenever $y^{(i)} = 0$:

$\theta^T x^{(i)} \leq -1$

$$\min_\theta \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

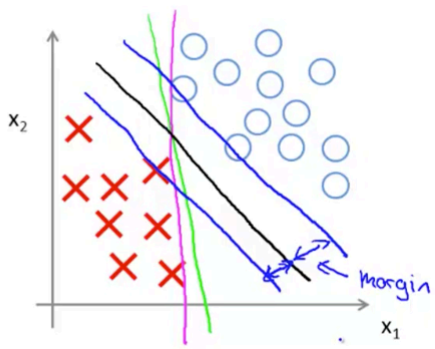
s.t. $\theta^T x^{(i)} \geq 1$ if $y^{(i)} = 1$

$\theta^T x^{(i)} \leq -1$ if $y^{(i)} = 0$

As you can see the middle term needs to go to zero to be minimize.

We then see a very interesting decision boundary:

SVM Decision Boundary: Linearly separable case



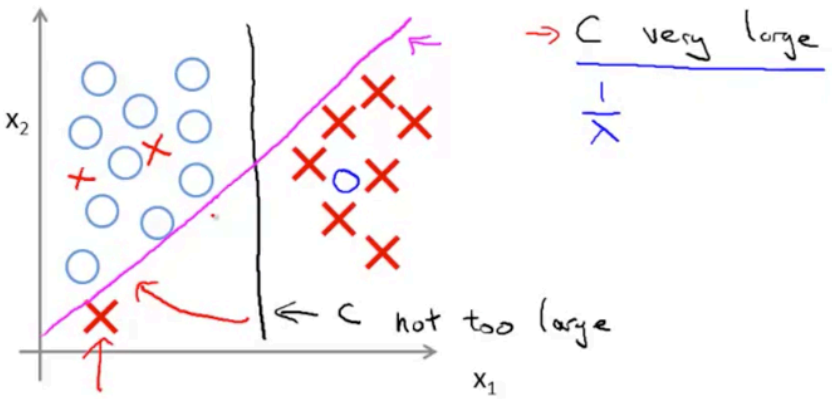
Large margin classifier

The SVM tries to separate the data with as much margin as possible.

Using Large Margin Classifiers can leave your learning algorithm vulnerable to outliers.

If C were very large we would see a decision boundary like the magenta line, but if left C not too large we would see something more like the black line even if your data isn't linearly separable like the graph above:

Large margin classifier in presence of outliers

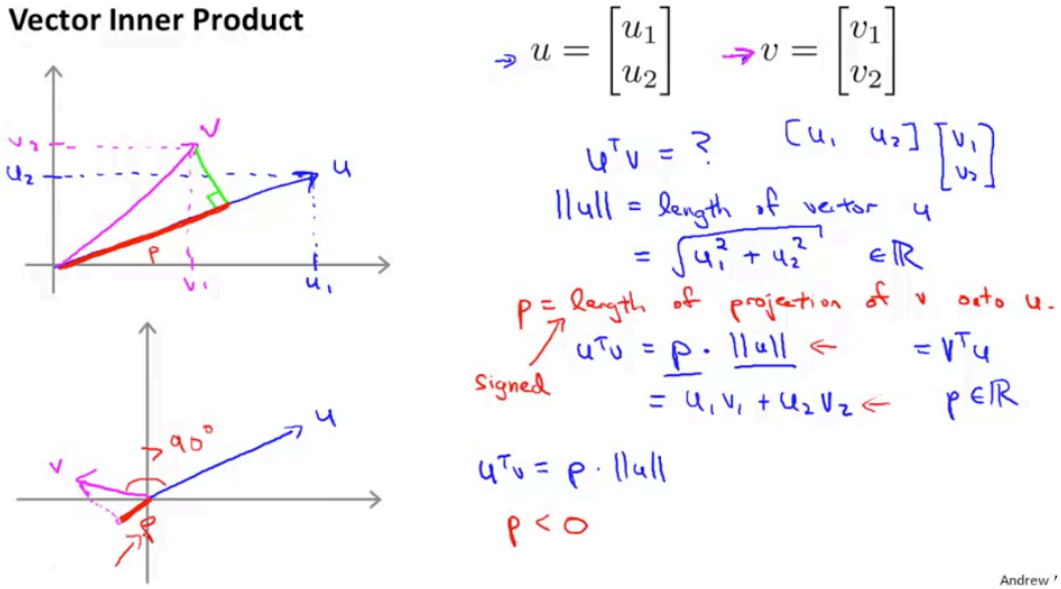


In the next video we will explore the steps to understand how we go from the optimization function to a large margin classifier.

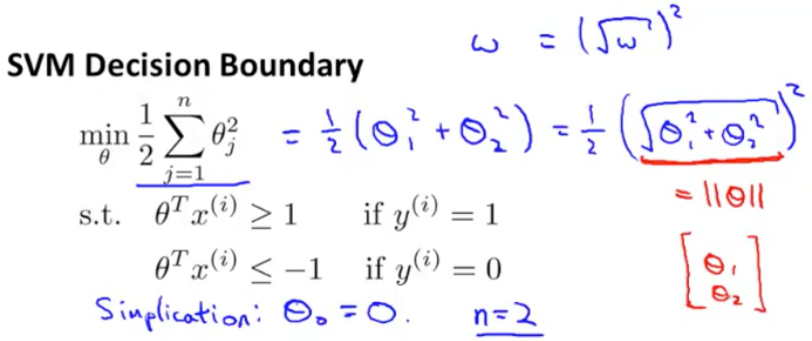
Mathematics Behind Large Margin Classification

This should provide better intuition about how the optimization of SVM leads to large classifiers.

Recall the Vector Inner Product:

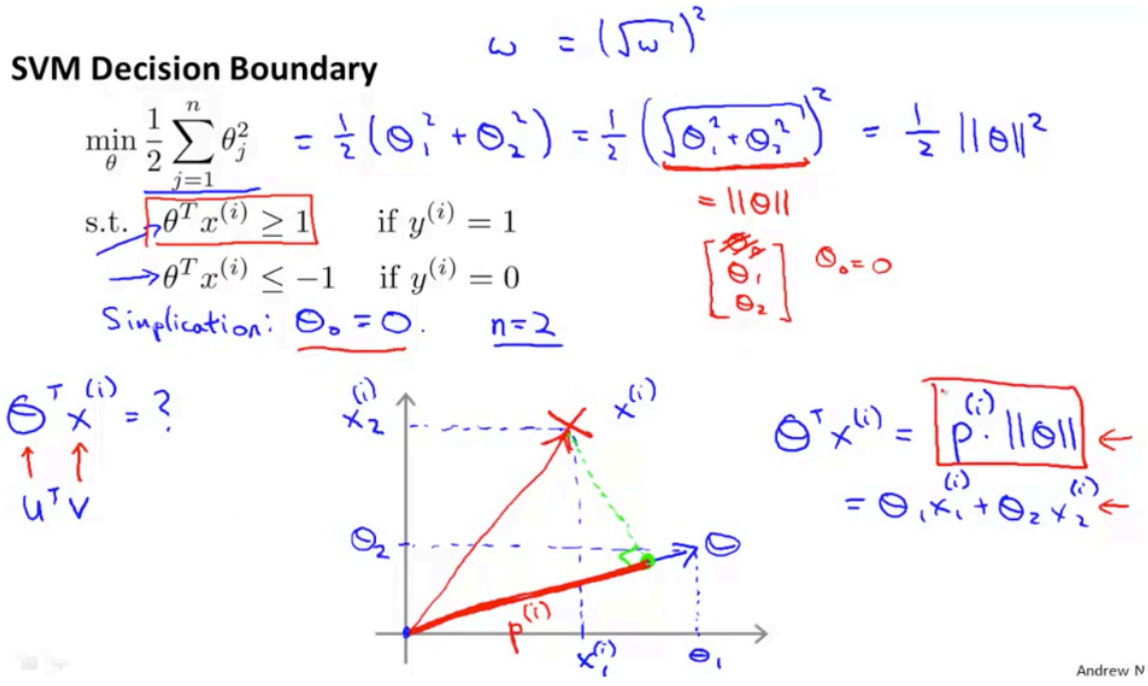


Now, let us look at the optimization objective of the SVM:



So essentially what the optimization object is doing is minimizing the squared norm of the squared length of the parameter vector theta.

Now lets look at (theta transpose x)



As such we can write (p I ||theta||) into our optimization object we get the following:

SVM Decision Boundary

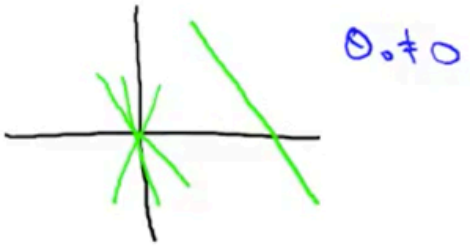
$$\min_\theta \frac{1}{2}\sum_{j=1}^{n}\theta_j^2 = \frac{1}{2}\|\theta\|^2$$

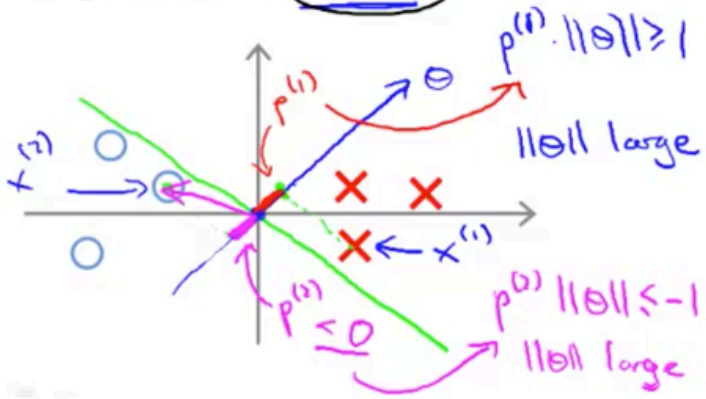
s.t. $p^{(i)} \cdot \|\theta\| \geq 1$ if $y^{(i)} = 1$

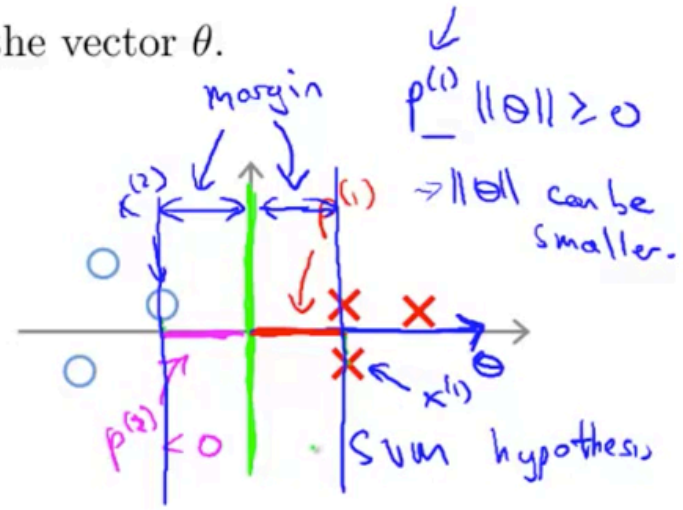
$p^{(i)} \cdot \|\theta\| \leq -1$ if $y^{(i)} = 1$

where $p^{(i)}$ is the projection of $x^{(i)}$ onto the vector θ.

Simplification: $\theta_0 = 0$







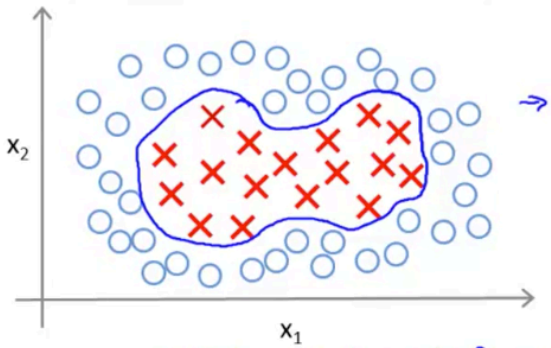
Kernels

Kernels 1

In this video we will start adapting SVMs in order to develop complex non-linear classifiers.

The main technique for doing this is called kernels.

Non-linear Decision Boundary



Predict $y = 1$ if

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \cdots \geq 0$$

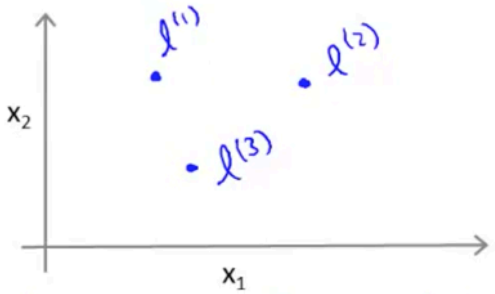
$h_\theta(x) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \cdots \geq 0 \\ 0 & \text{otherwise.} \end{cases}$

$\Rightarrow \theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \cdots$

$f_1 = x_1, \quad f_2 = x_2, \quad f_3 = x_1 x_2, \quad f_4 = x_1^2, \quad f_5 = x_2^2, \ldots$

Is there a different / better choice of the features $f_1, f_2, f_3, \ldots$?

Kernel



Given x, compute new feature depending on proximity to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$

Given x:

$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$

$f_2 = \text{similarity}(x, l^{(2)}) = \exp\left(-\frac{\|x - l^{(2)}\|^2}{2\sigma^2}\right)$

$f_3 = \text{similarity}(x, l^{(3)}) = \exp(\ldots)$

Kernel (Gaussian kernels) $k(x, l^{(i)})$

The 'similarity' function is what we call the kernel. This particular kernel is called a gaussian kernel.

Kernels and Similarity

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(1)})^2}{2\sigma^2}\right)$$

If $x \approx l^{(1)}$:

$f_1 \approx \exp\left(-\frac{0^2}{2\sigma^2}\right) \approx 1$

$l^{(1)} \to f_1$

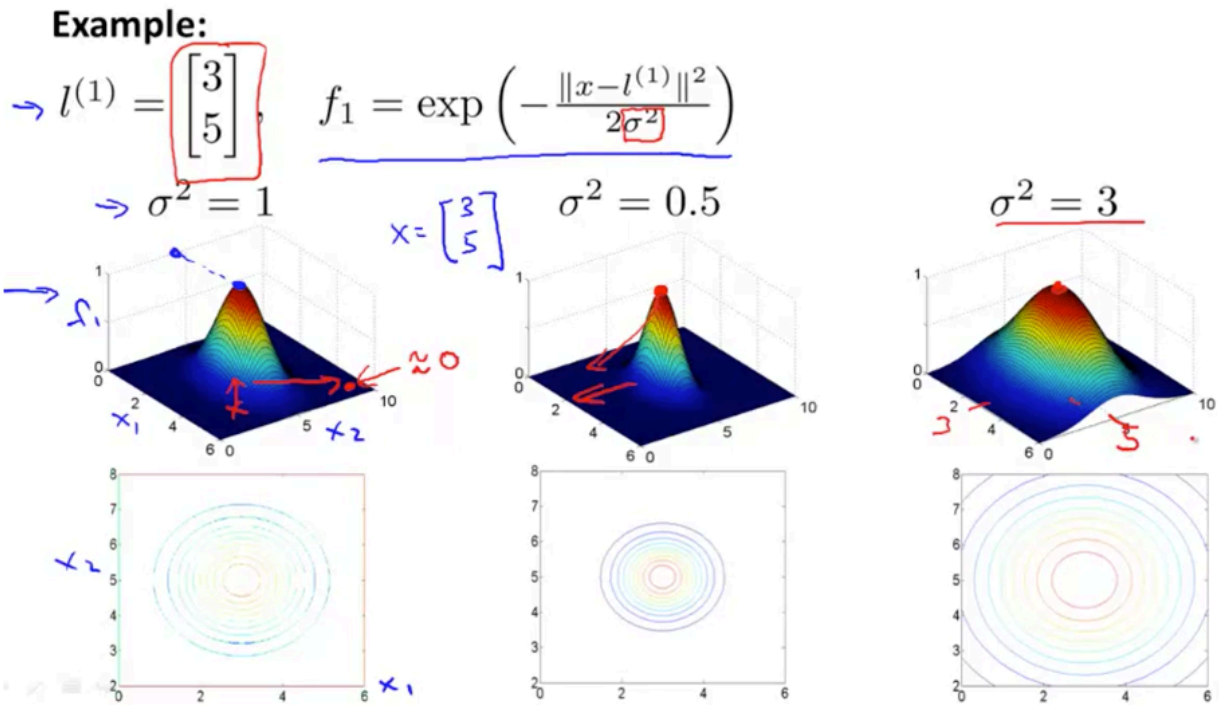
$l^{(2)} \to f_2$

$l^{(3)} \to f_3$.

If x if far from $l^{(1)}$:

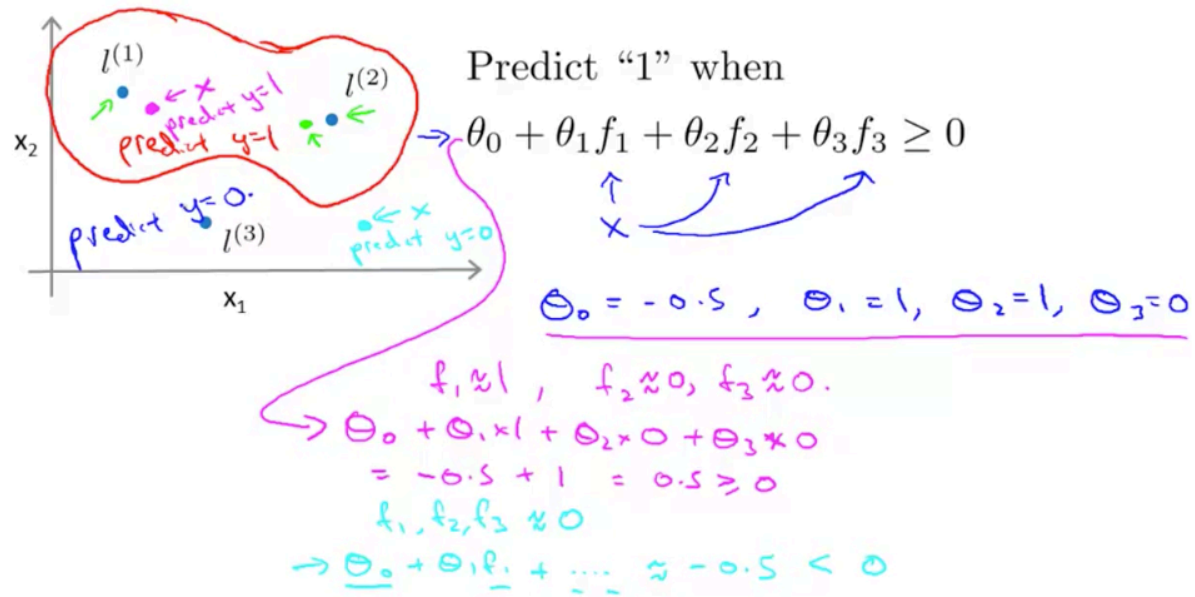
$f_1 = \exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \approx 0.$

Lets take a better look at the similarity function:



Given the following features, lets say we have f and see what the hypothesis will be. Say we get the following parameters:

What happens if we have a training example at the magenta dot, cyan dot, or green dot?

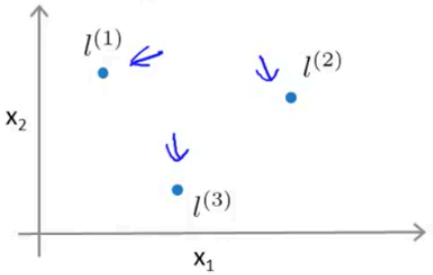


Next we will investigate how do we choose these landmarks and other similarity functions we can use.

Kernels 2

Let us explore how we choose the landmarks and other similarity functions as well as how to use these ideas in practice.

Choosing the landmarks



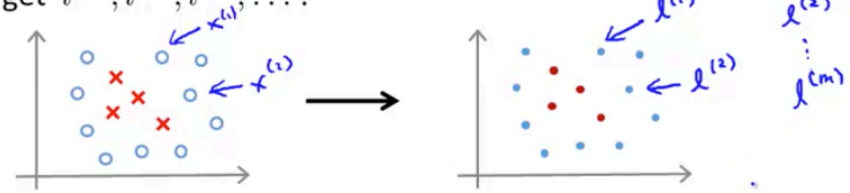
Given x:

$f_i = \text{similarity}(x, l^{(i)})$

$= \exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right)$

Predict $y = 1$ if $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

Where to get $l^{(1)}, l^{(2)}, l^{(3)}, \ldots$?



What we do is create a landmark in the exact location of every training example we have.

f == feature vector

SVM with Kernels

Given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$,

choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots, l^{(m)} = x^{(m)}$.

Given example x:

$f_1 = \text{similarity}(x, l^{(1)})$

$f_2 = \text{similarity}(x, l^{(2)})$

$\ldots$

$f = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix}$ $f_0 = 1$

For training example $(x^{(i)}, y^{(i)})$:

$f_1^{(i)} = \text{sim}(x^{(i)}, l^{(1)})$

$f_2^{(i)} = \text{sim}(x^{(i)}, l^{(2)})$

$f_i^{(i)} = \text{sim}(x^{(i)}, l^{(i)}) = \exp(-\frac{0}{2\sigma^2}) = 1$

$f_m^{(i)} = \text{sim}(x^{(i)}, l^{(m)})$

$x^{(i)} \in \mathbb{R}^{n+1}$ (or $\mathbb{R}^n$)

$f^{(i)} = \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix}$ $f_0^{(i)} = 1$

Andrew

If we want to make a prediction, we first compute a feature vector f for the new x:

SVM with Kernels

Hypothesis: Given x, compute features $f \in \mathbb{R}^{m+1}$

→ Predict "y=1" if $\theta^T f \geq 0$

$\theta_0 f_0 + \theta_1 f_1 + \cdots + \theta_m f_m$

$\theta \in \mathbb{R}^{m+1}$

$n = m$

Training:

$$\min_\theta C \sum_{i=1}^{m} y^{(i)} cost_1(\theta^T f^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^{m} \theta_j^2$$

$\theta^T x^{(i)}$ $\theta^T f^{(i)}$

$\rightarrow \theta_0$

$\sum_j \theta_j^2 = \theta^T \theta$ ← $\theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$ (ignore θ_0)

$\|\theta\|^2$

$\theta^T M \theta$ ←

$M = 10{,}000$

Let's look at the bias and variance trade offs when using a SVM:

SVM parameters:

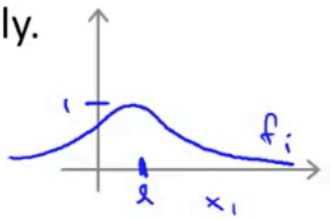
C ($= \frac{1}{\lambda}$). → Large C: Lower bias, high variance. (small λ)

→ Small C: Higher bias, low variance. (large λ)

σ^2 Large σ^2: Features f_i vary more smoothly.

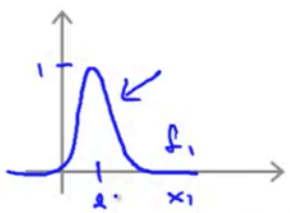
→ Higher bias, lower variance.

$\exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$



Small σ^2: Features f_i vary less smoothly.

Lower bias, higher variance.



Using an SVM

It is not a good idea to write your own software to compute the SVM optimization problem, or any other linear algebra functions. You should use one of the highly optimized libraries for these tasks.

Some SVM software packages are liblinear, libsvm, ... to solve for parameters theta.

The tasks we are responsible for is:
- Choice of parameter C
- Choice of kernel (similarity function)
- Choice of sigma

No kernel (linear kernel) is a valid option.

E.g. No kernel ("linear kernel")
Predict "y = 1" if $\theta^T x \geq 0$

$\theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n \geq 0$

→ n large, m small

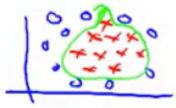
$x \in \mathbb{R}^{n+1}$

Gaussian kernel:

$f_i = \exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right)$, where $l^{(i)} = x^{(i)}$.

Need to choose σ^2.

$x \in \mathbb{R}^n$, n small and/or m large



If you do decide to use a gaussian kernel, here is what you need to do:

Kernel (similarity) functions:

$x^{(i)}$, $l^{(j)} = x^{(j)}$

```
function f = kernel(x1,x2)
    f = exp( -|| x1 - x2 ||^2 / (2σ^2) )
return
```

$x \rightarrow \begin{matrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{matrix}$

→ Note: Do perform feature scaling before using the Gaussian kernel.

$x \in \mathbb{R}^n$

$||x - l||^2$

$v = x - l$

$||v||^2 = v_1^2 + v_2^2 + \cdots + v_n^2$

$= (x_1 - l_1)^2 + (x_2 - l_2)^2 + \cdots + (x_n - l_n)^2$

1000 feet^2 1-5 bedrooms

Other choices of kernel

Note: Not all similarity functions $\text{similarity}(x, l)$ make valid kernels.
(Need to satisfy technical condition called "Mercer's Theorem" to make sure SVM packages' optimizations run correctly, and do not diverge).

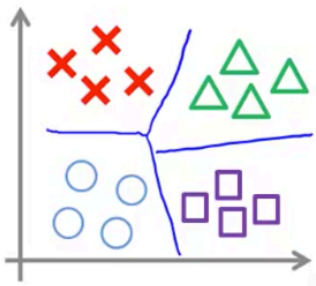
Many off-the-shelf kernels available:

- Polynomial kernel: $k(x, l) = (x^T l)^2$, $(x^T l)^3$, $(x^T l + 1)^3$, $(x^T l + 5)^4$, $(x^T l + \text{constant})^{\text{degree}}$
- More esoteric: String kernel, chi-square kernel, histogram intersection kernel, ... $sim(x, l)$

Usually the polynomial kernel performs worse than the gaussian kernel but it is something you may see.

How do you get an SVM to output decision boundaries for multi-class classification problems?

Multi-class classification



$$y \in \{1, 2, 3, \ldots, K\}$$

Many SVM packages already have built-in multi-class classification functionality.

Otherwise, use one-vs.-all method. (Train K SVMs, one to distinguish $y = i$ from the rest, for $i = 1, 2, \ldots, K$), get $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$

Pick class i with largest $(\theta^{(i)})^T x$ ($y=1$, $y=2$, ..., $\theta=K$)

When to use Logistic Regression, or SVM?

Logistic regression vs. SVMs

n = number of features ($x \in \mathbb{R}^{n+1}$), m = number of training examples

- If n is large (relative to m): (e.g. $n \geq m$, $n = 10{,}000$, $m = 10 \cdots 1000$)
  - Use logistic regression, or SVM without a kernel ("linear kernel")
- If n is small, m is intermediate: ($n = 1$-1000, $m = 10$-$10{,}000$)
  - Use SVM with Gaussian kernel
- If n is small, m is large: ($n = 1$-1000, $m = 50{,}000+$)
  - Create/add more features, then use logistic regression or SVM without a kernel
- Neural network likely to work well for most of these settings, but may be slower to train.

** Note SVMs have a convex optimization problem and won't get stuck in local optima