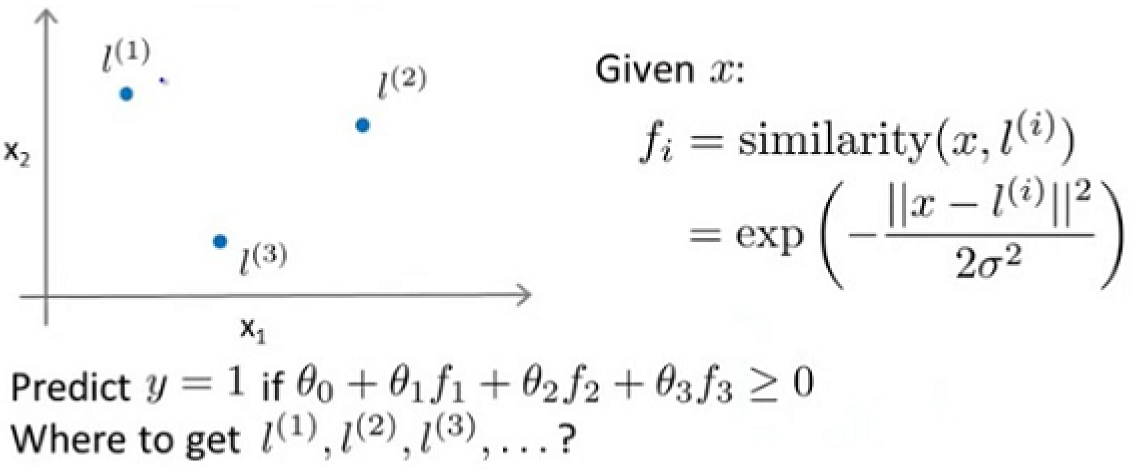
Kernel II

Previously, we started to talk about the kernels idea and how it can be used to define new features for the support vector machine, in this lesson we're going to filling in missing detail and practical implications regarding kernels (e.g. the bias-variance tradeoff)

- We spoke about picking landmarks manually, defining the kernel, and building a hypothesis function:
    - Where do we get the landmarks from?
    - For complex problems we probably want lots of them

Choosing the landmarks



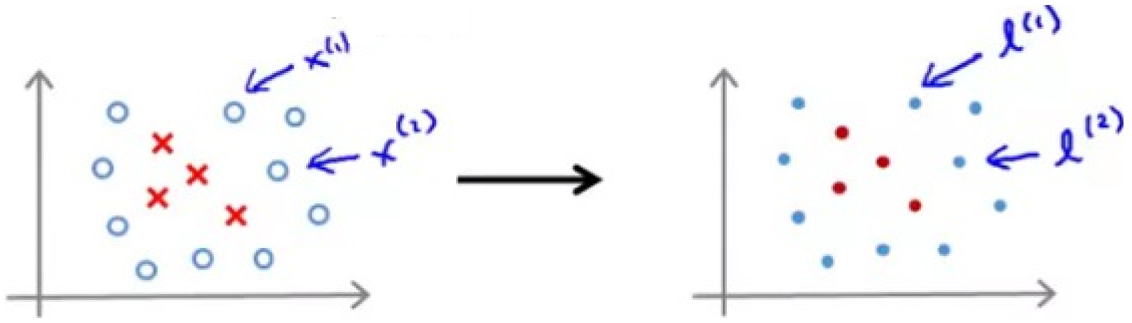
Given x:

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

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

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

So in practice this is how the landmarks are chosen which is that given the machine learning problem. We have some data set of some positive and negative examples. So, this is the idea here which is that we're gonna take the examples and for every training example that we have, we're just going to put landmarks as exactly the same locations as the training examples.



What we're going to end up with using this method is, we're going to end up with m landmarks of $l^{(1)}$, $l^{(2)}$, $\ldots$, $l^{(m)}$ (one landmark per location per training example). This is nice because it is saying that our features are basically going to measure how close an example is to one of the things we saw in our training set (means our features measure how close to a training set example something is).

So, just to write this outline a little more concretely:

- 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 (this example x can be something in the training set, it can be something in the cross validation set, or it can be something in the test set, we're going to compute the following features:

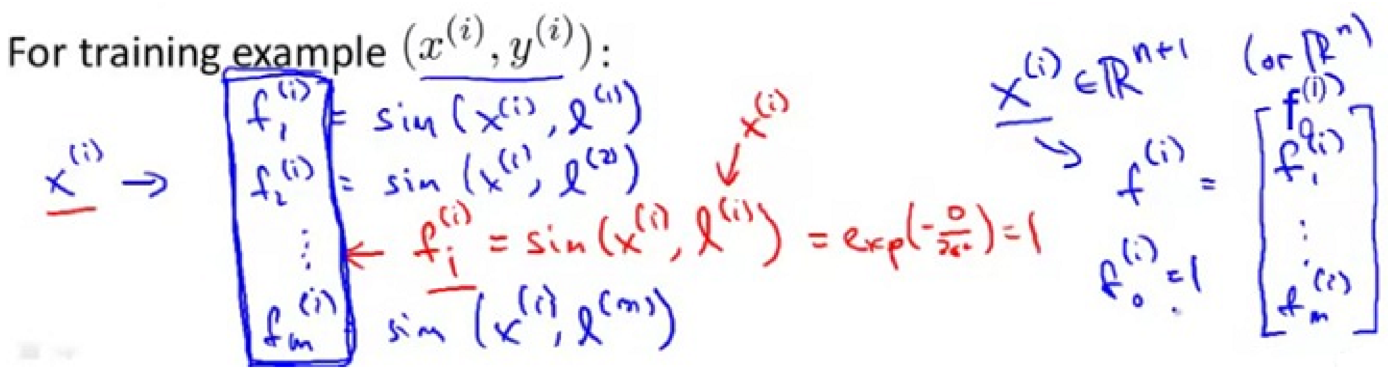
- $f_1 = \text{similarity}(x, l^{(1)})$
- $f_2 = \text{similarity}(x, l^{(2)})$
- $\ldots$

Where $l^{(1)}$ is actually $x^{(1)}$), and so on and these features then give us a feature vector:

$f = \begin{bmatrix} f_0 \\ f_1 \\ \dots \\ f_m \end{bmatrix}$ where f_0 is always equals to 1, this plays a role similar to what we had previously (where f_0 was our interceptor).

A more detailed look at generating the f vector:

- If we had a training example - features we compute would be using $(x^{(i)}, y^{(i)})$
- So we just cycle through each landmark, calculating how close to that landmark actually $x^{(i)}$ is:



For training example $(x^{(i)}, y^{(i)})$:
$x^{(i)} \rightarrow$
$f_1^{(i)} = sim(x^{(i)}, l^{(1)})$
$f_2^{(i)} = sim(x^{(i)}, l^{(2)})$
$f_i^{(i)} = sim(x^{(i)}, l^{(i)}) = exp(-\frac{0}{2\sigma^2}) = 1$
$x^{(i)}$
$f_m^{(i)}$ $sim(x^{(i)}, l^{(m)})$
$x^{(i)} \in \mathbb{R}^{n+1}$ (or $\mathbb{R}^n$)
$f^{(i)} =$
$f_0^{(i)} = 1$
$f_0^{(i)}$ $f_1^{(i)}$ $f_m^{(i)}$

So somewhere in the list we compare x to itself... (i.e. when we're at $f_i^{(i)}$)

- $f_i^{(i)} = sim(x^{(i)}, l^{(i)}) = \exp(-\frac{0}{2\sigma^2}) = 1$
- So because we're using the Gaussian Kernel this evalues to 1
- So one of our features for this training example is going to be equal to 1.

So instead of representing our training example, using $x^{(i)}$ which is $n + 1$ dimensional vector, we can now instead represent our training example using this feature vector $f^{(i)}$, and $f^{(i)}$ is the f feature vector for the i^{th} example.

Given these kernels, how do we use a support vector machine?

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

- Predict "$y = 1$" if $\theta^T f \geq 0$
  - Where $\theta^T f = \theta_0 f_0 + \theta_1 f_1 + \dots + \theta_m f_m$
  - $\theta \in \mathbb{R}^{m+1}$ dimensional vector (we have m here because the number of landmarks is equal to the training set size)

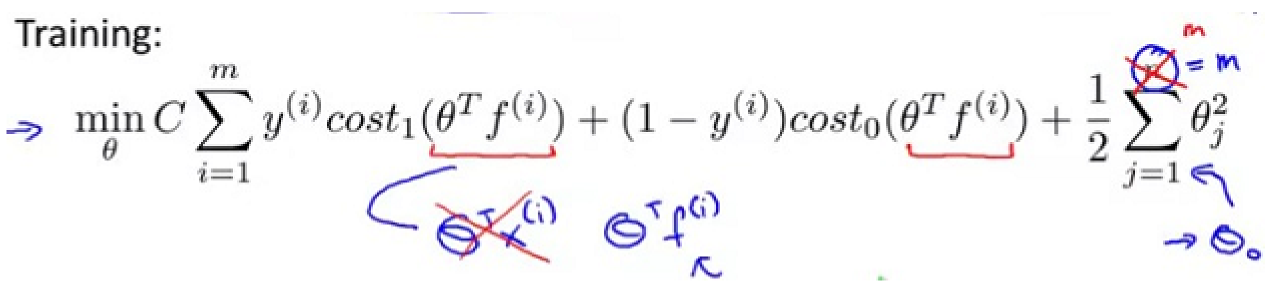
If we already have a learning set of parameters θ, then if we given a value of x and we want to make a prediction, what we do is we compute the features f, which is $\mathbb{R}^{m+1}$ dimensional vector (m because we have m training examples and thus m landmarks). So we predict "$y = 1$" if $\theta^T f \geq 0$, this is how we make a prediction assuming we already have θ.

SVM training with kernels

Using the Support Vector Machine learning algorithm:

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

- Now, we minimize using $f^{(i)}$ as the feature vector instead of $x^{(i)}$
- It's by solving this minimization problem that we get the parameters for our Support Vector Machine.
- In this setup, $m = n$
  - Because number of features is the number of training data examples we have



Training:

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

$\cancel{\theta^T x^{(i)}}$ $\theta^T f^{(i)}$ $\rightarrow \theta_0$

And so this gives us a slightly different distance metric. We'll use a slightly different measure instead of minimizing exactly the norm of theta squared means that minimize something slightly similar to it. That's like a rescale version of the parameter vector theta that depends on the kernel. But this is kind of a mathematical detail. That allows the support vector machine software to run much more efficiently.

One final mathematic detail (not crucial to understand)

- If we ignore θ_0 then the following is true:
  - $\sum_{j=1}^{n} \theta_j^2 = \theta^T \theta$
- What many implementations do is: $\theta^T M \theta$, where the matrix M depends on the kernel we use
- M gives a slightly different measure instead of minimizing exactly the $||\theta||^2$ means that minimize something slightly similar to it. That's like a rescale version of the parameter vector θ that depends on the kernel.
  - Allows more efficient computation, and scale to much bigger training sets
  - If we have a training set with 10,000 values, means we get 10,000 features, solving for all these parameters can become expensive, so by adding this M we avoid a for loop and use a matrix multiplication algorithm instead
- We can apply kernels to other learning algorithms, but they tend to be very computationally expensive (support vector machine and kernels tend go particularly well together. Whereas, logistic regression and kernels, we can do it, but this would run very slowly)
- The SVM is far more efficient - so more practical algorithm

SVM parameters (Bias and variance trade off using SVM):

One other thing that is worth knowing about is when we're applying support vector machine, how do we choose the parameters of the support vector machine?

Choosing C, where C plays a role similar to $1/\lambda$ (where λ is the regularization parameter):

- Large C (small λ): **Lower bias**, **high variance** -> Overfitting
- Small C (large λ): **Higher bias**, **low variance** -> Underfitting

SVM parameters (σ^2)

So the parameter C is one of the parameters we need to choose. The other one is the parameter σ^2, which appeared in the Gaussian kernel.

- Large σ^2: Features f_i vary more smoothly.
  - Higher bias, lower variance.
- Small σ^2: Features f_i vary less smoothly.
  - Lower bias, higher variance.

Lower bias, higher variance.

Video Question: Suppose you train an SVM and find it overfits your training data. Which of these would be a reasonable next step? Check all that apply.

- Increase C
- [✓ Decrease C]
- [✓ Increase σ^2]
- Decrease σ^2