# Kernels I

**Kernels** allow us to make complex, non-linear classifiers using Support Vector Machines.

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

To do this, we find the "similarity" of x and some landmark $l^{(i)}$:

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

This "similarity" function is called a **Gaussian Kernel**. It is a specific example of a kernel.

The similarity function can also be written as follows:

$$f_i = similarity(x, l^{(i)}) = \exp(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(i)})^2}{2\sigma^2})$$

There are a couple properties of the similarity function:

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

If x is far from $l^{(i)}$, then $f_i = \exp(-\frac{(large\ number)^2}{2\sigma^2}) \approx 0$

In other words, if x and the landmark are close, then the similarity will be close to 1, and if x and the landmark are far away from each other, the similarity will be close to 0.

Each landmark gives us the features in our hypothesis:

$$\begin{aligned} l^{(1)} &\rightarrow f_1 \\ l^{(2)} &\rightarrow f_2 \\ l^{(3)} &\rightarrow f_3 \\ &\dots \\ h_\Theta(x) &= \Theta_1 f_1 + \Theta_2 f_2 + \Theta_3 f_3 + \dots \end{aligned}$$

$\sigma^2$ is a parameter of the Gaussian Kernel, and it can be modified to increase or decrease the **drop-off** of our feature $f_i$. Combined with looking at the values inside Θ, we can choose these landmarks to get the general shape of the decision boundary.

# Kernels II

One way to get the landmarks is to put them in the **exact same** locations as all the training examples. This gives us m landmarks, with one landmark per training example.

Given example x:

$f_1 = similarity(x, l^{(1)}), f_2 = similarity(x, l^{(2)}), f_3 = similarity(x, l^{(3)})$, and so on.

This gives us a "feature vector," $f_{(i)}$ of all our features for example $x_{(i)}$. We may also set $f_0 = 1$ to correspond with $\Theta_0$. Thus given training example $x_{(i)}$:

$$x^{(i)} \rightarrow \begin{bmatrix} f_1^{(i)} = similarity(x^{(i)}, l^{(1)}) \\ f_2^{(i)} = similarity(x^{(i)}, l^{(2)}) \\ \vdots \\ f_m^{(i)} = similarity(x^{(i)}, l^{(m)}) \end{bmatrix}$$

Now to get the parameters Θ we can use the SVM minimization algorithm but with $f^{(i)}$ substituted in for $x^{(i)}$:

$$\min_\Theta C \sum_{i=1}^{m} y^{(i)} \text{cost}_1(\Theta^T f^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^{n} \Theta_j^2$$

Using kernels to generate f(i) is not exclusive to SVMs and may also be applied to logistic regression. However, because of computational optimizations on SVMs, kernels combined with SVMs is much faster than with other algorithms, so kernels are almost always found combined only with SVMs.

## Choosing SVM Parameters

Choosing C (recall that $C = \frac{1}{\lambda}$

- If C is large, then we get higher variance/lower bias
- If C is small, then we get lower variance/higher bias

The other parameter we must choose is $\sigma^2$ from the Gaussian Kernel function:

With a large $\sigma^2$, the features fi vary more smoothly, causing higher bias and lower variance.

With a small $\sigma^2$, the features fi vary less smoothly, causing lower bias and higher variance.

### Using An SVM

There are lots of good SVM libraries already written. A. Ng often uses 'liblinear' and 'libsvm'. In practical application, you should use one of these libraries rather than rewrite the functions.

In practical application, the choices you do need to make are:

- Choice of parameter C

- Choice of kernel (similarity function)
- No kernel ("linear" kernel) -- gives standard linear classifier
- Choose when n is large and when m is small
- Gaussian Kernel (above) -- need to choose $\sigma^2$
- Choose when n is small and m is large

The library may ask you to provide the kernel function.

**Note:** do perform feature scaling before using the Gaussian Kernel.

**Note:** not all similarity functions are valid kernels. They must satisfy "Mercer's Theorem" which guarantees that the SVM package's optimizations run correctly and do not diverge.

You want to train C and the parameters for the kernel function using the training and cross-validation datasets.

## Multi-class Classification

Many SVM libraries have multi-class classification built-in.

You can use the *one-vs-all* method just like we did for logistic regression, where $y \in 1, 2, 3, \dots, K$ with $\Theta^{(1)}, \Theta^{(2)}, \dots, \Theta(K)$. We pick class i with the largest $(\Theta^{(i)})^T x$.

## Logistic Regression vs. SVMs

If n is large (relative to m), then use logistic regression, or SVM without a kernel (the "linear kernel")

If n is small and m is intermediate, then use SVM with a Gaussian Kernel

If n is small and m is large, then manually create/add more features, then use logistic regression or SVM without a kernel.

In the first case, we don't have enough examples to need a complicated polynomial hypothesis. In the second example, we have enough examples that we may need a complex non-linear hypothesis. In the last case, we want to increase our features so that logistic regression becomes applicable.

**Note**: a neural network is likely to work well for any of these situations, but may be slower to train.