**The optimization objectives of SVM:**

minθ C \* ∑i=1m[y\_i \* cost1(θTx\_i) + (1-y\_i) \* cost0(θTx\_i)] + ½ \* ∑j=1nθ2j (1)

Cost1(z) = -log(1/(1 + e-z)); Cost0(z) = -log(1 – 1/(1 + e-z))

The constant C is used to trade-off the bias and variance like the parameter lambda in logistic regression.

Just like lambda in logistic regression, when lambda set to large, it will weight the regularization more than cost function. When C is set to be very small value that correspond to giving the regularization term a much larger rate than the first part.

The hθ(x) = 1 if θTx >= 0

0 otherwise

SVM:

According to the graph of Cost1(z) and Cost0(z);

Cost1(z) :

**1 z**

Cost0(z):

**-1 z**

if y = 1, we want θTx >= 1, that makes the first term of (1) will equal to 0.

if y = 0, we want θTx <= -1, that makes the first term of (1) also will be 0.

when the first part of (1) is 0, and we will get the new optimization objectives:

minθ  ½ \* ∑j=1nθ2j

s.t. θTx\_i >= 1 if y\_i = 1;

θTx\_i <= -1 if y\_i = 0;

**Kernel:**

Given x, compute new feature depending on proximity to landmarks l1, l2, l3.

f1 = similarity(x, l1),

f2 = similarity(x, l2),

f3 = similarity(x, l3)

the similarity() is the kernel function.

e.g. Gaussian kernel = e^(- ||x – l\_i||^2 / (2 \* б^2))

if x close to l1, then ||x – l\_i|| close to 0. so the f1 close to 1.

if x is far from l1, then ||x – l\_i|| will be very large. So the f1 close to 0.

This measure is measure how x close to f1.

The landmark l\_i create some new features f\_i given the training example x.

**Choosing the landmark l\_i**

By Andrew Ng,

Given (x1, y1), (x2, y2), (x3, y3), … , (xm, ym),

choose l1 = x1, l2 = x2, l3 = x3, … , lm = xm.

Given example x:

f1 = similarity(x, l1)

f2 = similarity(x, l2)

…...

fm = similarity(x, lm)

f = [f0, f1, f2, …, fm](f0 = 1)

Predict “y = 1” if θTf >= 0;

For training example (xi, yi):

xi => [f0\_i, f1\_i, f2\_i, f3\_i, …, fm\_i]( fj\_i = sim(xi, lj), j from 1 up to m), f0\_i = 1;

finally, for training phrase:

minθ C \* ∑i=1m[y\_i \* cost1(θTf\_i) + (1-y\_i) \* cost0(θTf\_i)] + ½ \* ∑j=1nθ2j

this n actual equals to m, and this is also similar to other algorithm that the regularization not include the theta0.

it's by solving this minimization problem, you then get the parameters vector theta for your SVM.

Some trick for implementation

∑j=1nθ2j =θTθ in the previous algorithm. But it's slightly different in SVM.

∑j=1nθ2j =θTMθ in the SVM, and the M matrix is depending on the specific kernel you use.

SVM parameters:

C( = 1/lambda). Large C: Lower bias, high variance.

Small C: Higher bias, low variance.

If we use the Gaussian kernel, we also should choose the parameter б^2(sigma squared).

б^2 Large б^2: Features fi vary more smoothly.

High bias, lower variance.

Small б^2: Features fi vary less smoothly.

Lower bias, higher variance.

**SVM in Practice:**

Need to specify:

Choice of parameter C.

Choice of kernel.

E.g. No kernel(“linear kernel”)

predict “y = 1” if θTx >= 0;

if the features n is very large whereas the training examples m is small, you could use the linear model to separate the training examples instead of using a very complicated nonlinear model.

The second choice is

Gaussian Kernel:

fi = e^(- ||x – l\_i||^2 / (2 \* б^2)), where l\_i = xi. For this kernel, we need to choose the б^2.

If your feature n is relatively small and training examples are pretty large, you may use the Gaussian kernel. The Gaussian kernel probably the most common kernel.

If your features take on very different ranges of value, what we should do essentially that **doing perform feature scaling before using the Gaussian Kernel.**

**Many off-the-shelf kernels available:**

- Polynomial kernel: k(x,l) = (xTl)^2, (xTl)^3, (xTl + 1)^3, more general form: (xTl + a)^b

- String kernel: if your input data is text strings or other types of strings.

- chi-square kernel, histogram kernel, intersection kernel

Multi-class classification:

Using one-vs-all method.(Train K SVMs, one to distinguish y = i from the rest, for i = 1,2,…, K), get θ\_1, θ\_2, …, θ\_K, pick class I with largest θ\_iTx;

Choice of Algorithm

n = number of features(x є Rn+1), m = number of training examples.

If n is large(relative to m):

Use logistic regression, or SVM without a kernel(“linear kernel”).

If n is small, m is intermediate:

Use Gaussian kernel

if n is small, m is large:

Create/add more features, then use logistic regression or SVM without a kernel

A well designed Neural network likely to work well for most of these settings, but may be slower to train.

The SVM optimization problem is a convex optimization problem and so the good SVM optimization software packages will always find the global minimum or something close to it.