→ Each landmark gives us the features in our hypothesis :

$$L^1 \rightarrow f_1$$
$$L^2 \rightarrow f_2$$
$$L^3 \rightarrow f_3$$

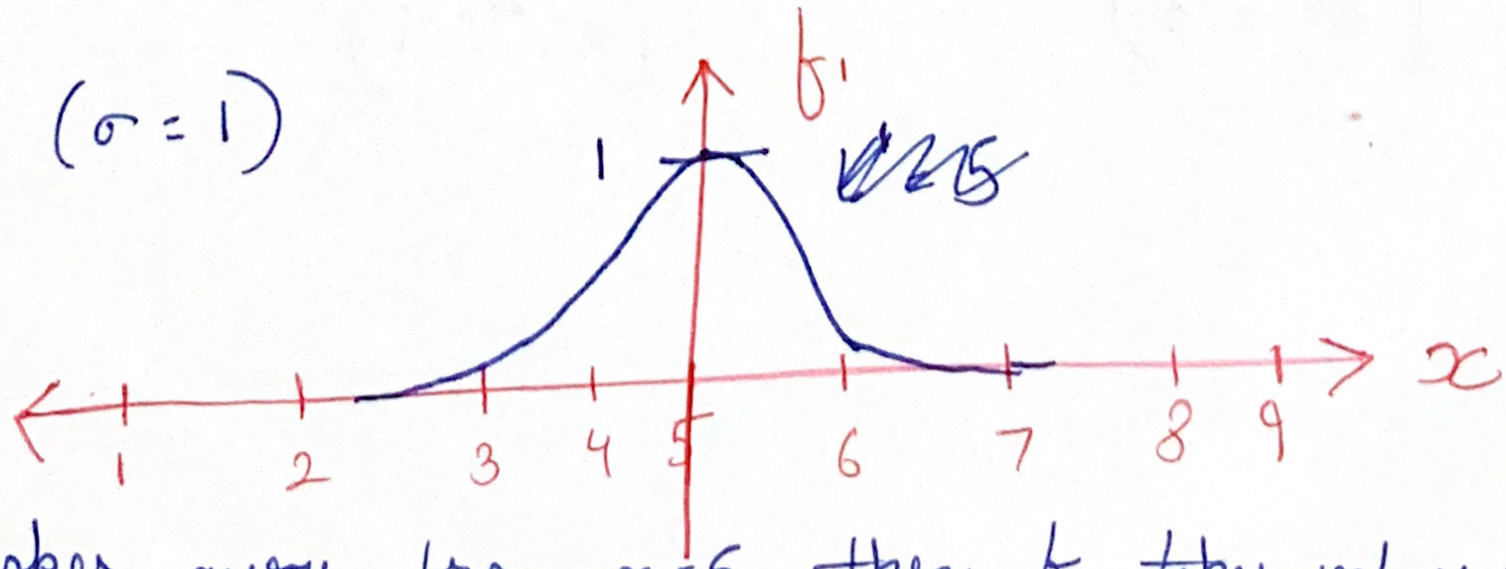
→ If we plot f_i vs the input vector, we'll observe that as x moves away from L^i, then the feature takes on values close to 0.

So, this measures how close x is to this landmark.

eg : Suppose L^1 is a 1-D vector, i.e. $L^1 = [5]$.

$$\therefore f_1 = \exp\left(-\frac{(x-5)^2}{2\sigma^2}\right)$$

→ Sketch : ($\sigma = 1$)



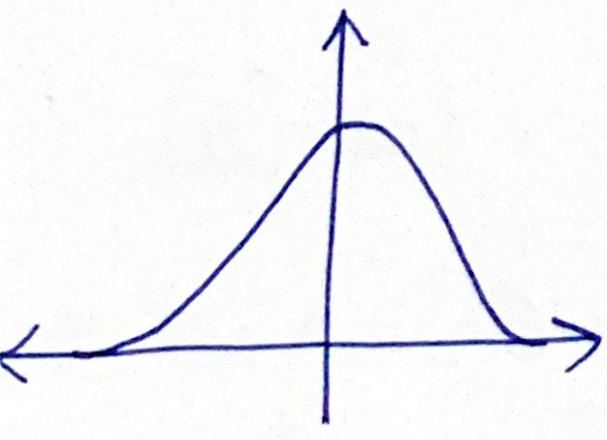
As x moves away from 5, then f_1 takes values close to 0.

⇒ What does σ^2 do ?

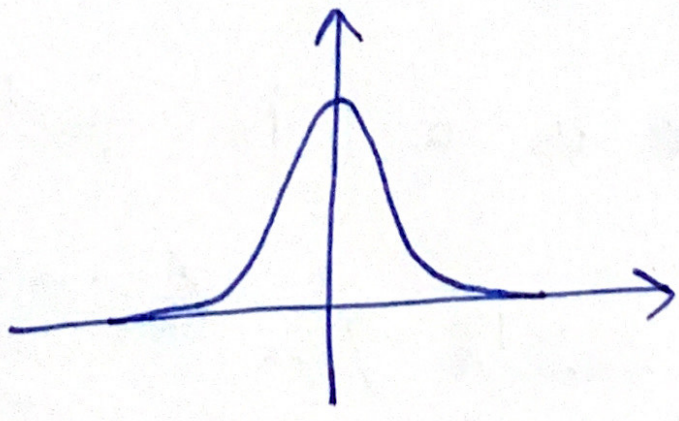
→ σ^2 = Variance → Measurement of the spread between numbers in a dataset, i.e. it measures how far each number in the set from the mean, & it also therefore from every other number in the set.

→ Variance also defines the steepness of the rise around the landmark. Low variance ⇒ more steepness

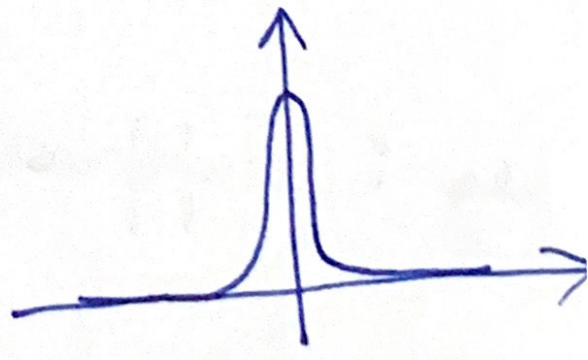
eg: for the example on the prev. page,



($\sigma^2 = 3$)



($\sigma^2 = 1$)

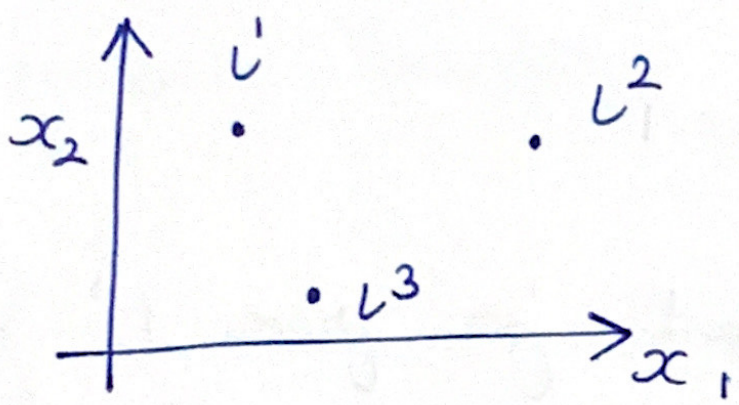


($\sigma^2 = 0.5$)

⇒ Given this definition, what kind of hypotheses can we learn ?

→ Suppose we have some training examples & we predict $y = 1$ when $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

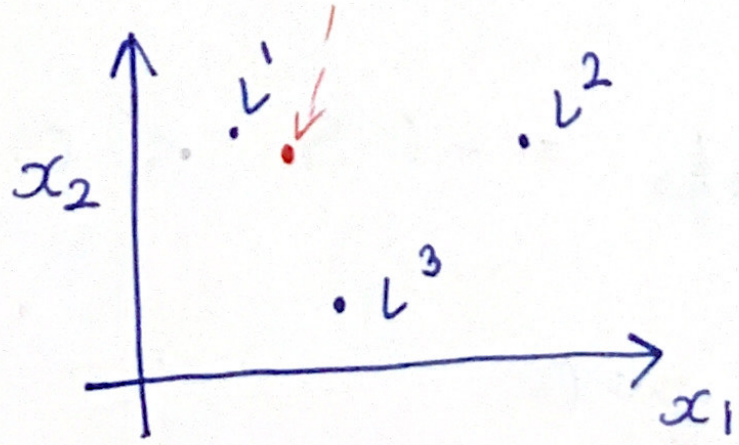
& suppose $x_0 = 0$



* Suppose we've magically obtained the θ vector & for our example, we obtain :

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

→ Given our placement of our landmarks, what happens if we have a training example like the one denoted by the red dot.



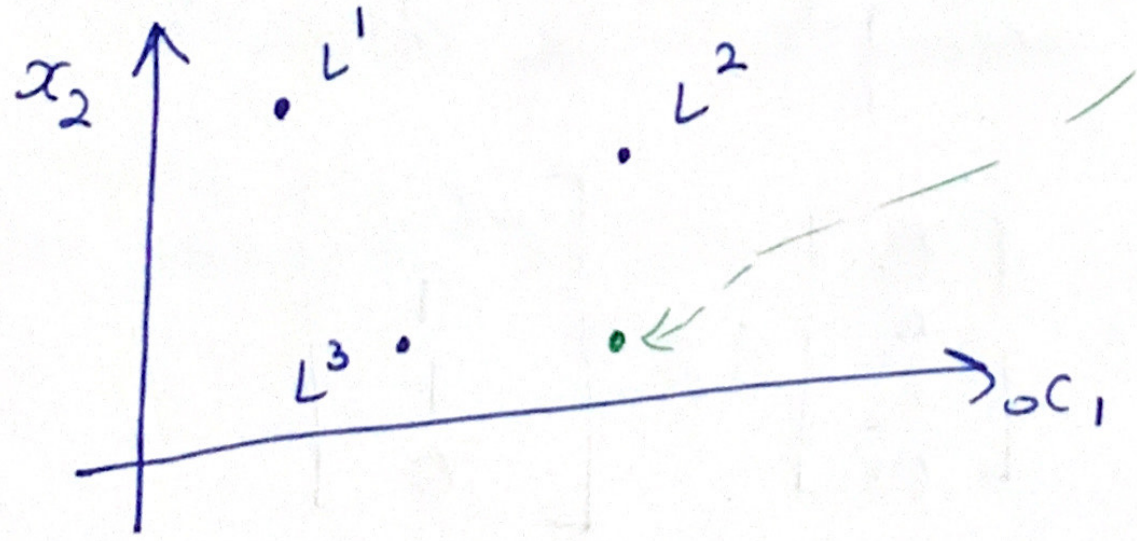
We know $f_1 \approx 1$ as point is near L^1, but f_2 & f_3 will be close to 0.

Coming to our hypothesis

$$\theta_0 f_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 = -0.5 + 1 + 0 + 0 = 0.5 \geq 0$$

$$\Rightarrow y = 1$$

→ If we had a point for from L^1 & L^2, but near L^3, we get: (denoted by green dot)



Conclusion Using sim. arguments, $f_1 = f_2 \approx 0$; $f_3 \approx 1$

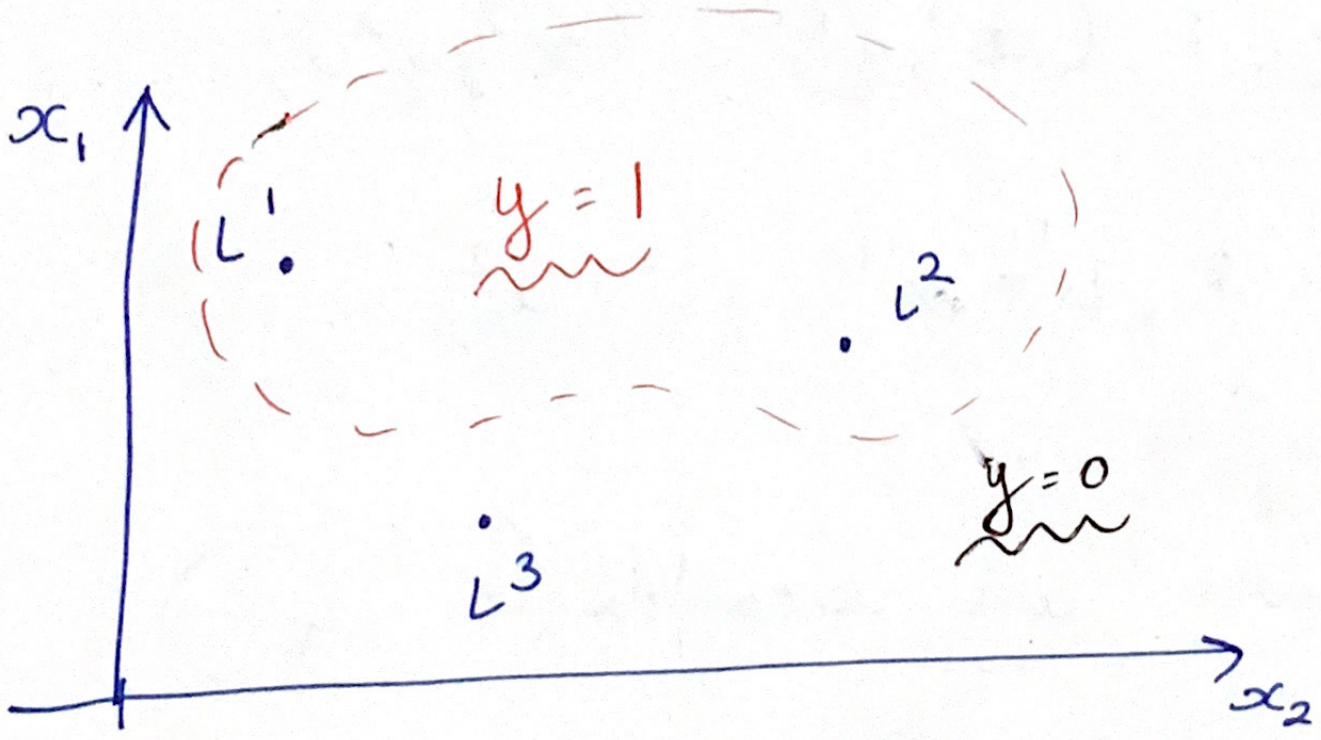
Computing hypothesis,

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 = -0.5 < 0$$

$$\Rightarrow y = 0$$

→ We can see that for points near l^1 & l^2, our hypothesis predicts $y=1$, & it predicts $y=0$ for points near l^3.

Which means we've created a non linear decision boundary which looks something like this :



→ This was cool !! But how do we choose the landmarks ??

⇒ Chossing the landmarks

One way to get the landmarks is to put them in the exact same locations as all the training examples.

This gave

→ Concretely,

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

Chosse : $l^1 = x^1, \; l^2 = x^2, \; \ldots, \; l^m = x^m$

* Given an example (x^i, y^i),

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

$$f_1^i = k(x^i, l^1)$$

$$f_2^i = k(x^i, l^2)$$

$$\vdots$$

$$f_i^i = k(x^i, l^i) = \exp(0) = 1$$

$$\vdots$$

$$f_m^i = k(x^i, l^m)$$

* Using this feature vector f instead of x makes using a SVM much more easier.

⇒ SVM with kernels

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

Training examples

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

→ Training :

$$\min_\theta \left\{ C \sum_{y=1}^{m} y^i \, cost_1(\theta^T f^i) + (1 - y^i) \, Cost_0(\theta^T f^i) + \frac{1}{2} \sum_{j=1}^{n=m} \theta_j^2 \right\}$$

As we've created ^ no. of features = no. of training examples

$\therefore \quad n = m$

→ One particular subtlety : If we ignore θ_0 then the fol. is true $\sum_{j=1}^{n} \theta_j^2 \equiv \theta^T \theta$. Most many implementations rewrite this as $\theta^T M \theta$.

* [illegible] ⟹ depends on the number of training example.

* This makes it efficient for large training example.

→ Using kernels to generate f_i is not exclusive to SVMs. It may also be applied to algorithms like logistic regression.

However, it is seen that kernels combined with SVMs is much faster than with other algorithms.

⇒ SVM parameters

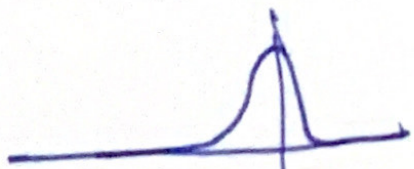
→ $C = 1/\lambda$:

* Large C ⇒ Small λ : Lower bias & higher variance

* Small C ⇒ Large λ : Higher bias & lower variance

→ σ^2 : * Large σ^2 ⇒ features vary more smoothly ⇒ Higher bias, lower variance

* Small σ^2 ⇒ feature vary less smoothly



⇒ Using a SVM

→ Use SVM software package (eg: liblinear, libsvm) to solve for parameters θ. 😊

→ Need to specify:

* Choice of parameter C
* Choice of kernel (similarity function)
  - No kernel or "Linear" kernel:
    - Predict $y = 1$ if $\theta^T x \geq 0 \rightarrow \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n \geq 0$
    - Used when n is large, m is small; As you have a really small & a high dimensional dataset, using a complex model might result in overfitting. So, using a simple linear model makes more sense in this case.
  - Gaussian Kernel: Predict $y = 1$ if $\theta^T f \geq 0$
    - $f_i = \exp\left(-\frac{\|x - l^i\|^2}{2\sigma^2}\right)$, where $l^i = x^i$
    - Need to choose σ^2
    - Used when n is small & m is large.

⇒ Points to remember if using Gaussian Kernel

→ Some SVM packages will expect you to define the kernel function explicitly.

eg: For Gaussian kernel,

x ↓ , i ↓

function f = kernel (x_1, x_2)

$$f = \exp\left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)$$

return

→ Make sure you perform feature scaling before using a Gaussian kernel.

If you don't, then features with large value will dominate the f value.

eg: Let $v = x - l^{(i)}$

Then $\|x - l\|^2 = \|v\|^2 = v_1^2 + v_2^2 + \cdots + v_n^2$

$= (x_1 - l^{(i)})^2 + (x_2 - l^{(i)})^2 + \cdots + (x_n - l^{(i)})^2$

If $x_1 \in [1000, 2000]$ & $x_2 \in [2, 10]$, then $(x_1 - l^{(i)})^2$ will result in large values

⇒ Other choices of kernel

→ Note: Not all similarity functions are valid Kernels. They must satisfy "Mercer's Theorem" in order to make sure that the SVM packages' optimizations run correctly, & do not diverge.

→ Polynomial kernel :

Similarity $(x, l) \equiv k(x, l) = (x^T l + k)^D$

; $k \rightarrow$ Constant ; $D \rightarrow$ constant degree of polynomial

eg: $k(x, l) = (x^T l)^2$, $(x^T l + 1)^2$, $(x^T l)^3$

* Uses the idea that if x & l are similar, then their inner product $(x^T l)$ will be large.
* Usually used when x & l are non-negative
* Not used that often & usually performs worse than Gaussian kernel.

→ More esoteric examples :

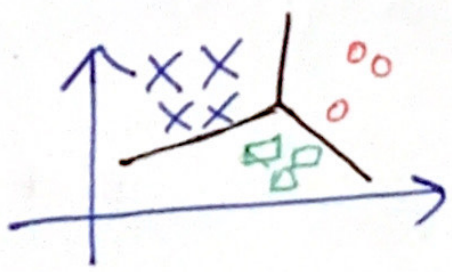
* String kernel → Used if input data consists of texts or strings.

* chi-square kernel

* histogram intersection kernel

→ Note : While chossing b/w the diff. types of kernel, and chossing parameters such as C, σ^2, etc. choose whatever performs best on the cross validation data.

⇒ Multi class classification



→ Many SVM libraries have this feature built-in.

→ Otherwise, use one vs all method, discussed in logistic regression.

* Train K SVMs, one to distinguish $y=i$ from the rest $\forall\, i = [1, 2, \ldots K]$ & get $\theta^1, \theta^2, \ldots, \theta^K$
(θ^1 : $y=1$, θ^2 : $y=2$, θ^K : $y=K$)

* Pick class i with largest value of $(\theta^i)^T x$.

⇒ Logistic regression vs SVM

→ Let n denote no. of features s.t. $x \in \mathbb{R}^{n+1}$ (x_0 included)

→ m = No. of training examples

→ If n is large relative to m eg: $n \geq m$, $n = 10000$
$m \in [10, 1000]$

*Use ~~Logis~~ logistic regression, or SVM without a kernel (linear kernel).

*Reason: You probably don't have enough ~~data~~ examples to ~~fit~~ need a very complicated non linear hypothesis function.

→ If n is small, m is intermediate eg: $n \in [1, 1000]$
$m \in [10, 10000]$

* Use SVM with Gaussian Kernel, as we have enough examples that we may need a complex non-linear hypothesis

→ If n is small, m is large: eg: $n \in [1, 1000]$
$m \in [50000, \infty)$

* Create/add more features & use logistic regression or SVM without kernel.

→ A neural net is likely to work well for any of these situations but may be slower to train.