

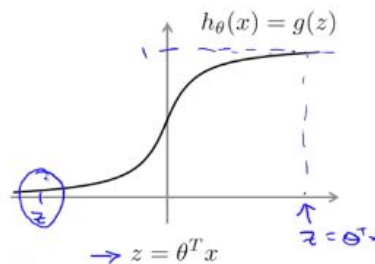
## Week 7: Support Vector Machines

### Optimization Objective

- We will look at Support Vector Machines now, the last Supervised Learning algorithm we will spend significant time on. Let's start with remembering our logistic regression algorithm where  $h_{\theta}(x) = 1/(1 + e^{-\theta^T x})$ . We refer to  $\theta^T x$  as  $z$ . Looking at our sigmoid function chart we can see that  $h_{\theta}(x) \approx 1$  when  $z$  is big and, conversely, we can see that  $h_{\theta}(x) \approx 0$  when  $z$  is very small. If  $y=1$  we want  $h_{\theta}(x) \approx 1$  and we want  $z \gg 0$ . If  $y=0$  we want  $h_{\theta}(x) \approx 0$  and therefore we want  $z \ll 0$

### Alternative view of logistic regression

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



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

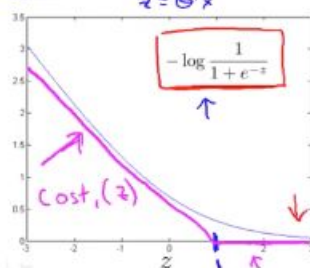
- Now let's see how we can construct the Support Vector Machine. We take the logistic regression cost function and graph below how each term contributes to the cost function when  $y=1$  (left chart) and when  $y=0$  (right chart).

### Alternative view of logistic regression

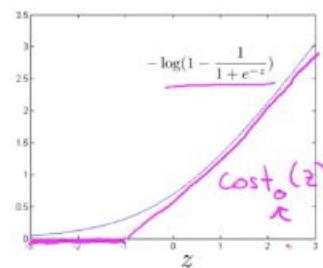
Cost of example:  $-(y \log h_{\theta}(x) + (1 - y) \log(1 - h_{\theta}(x)))$

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

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



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



What we do next is we take the original curve and create 2 lines: one from 1 to the end of the chart and the other one from 1 to the other end of the chart. We do the same for the other case and we end up with 2 costs:  $Cost_1(z)$  and  $Cost_0(z)$ . With this new information we can now build the cost function for SVM

- We start with the original logistic regression function and introduce some new conventions
  - $Cost_1(z) = -\log(h_\theta(x^{(i)}))$
  - $Cost_0(z) = -\log(1 - h_\theta(x^{(i)}))$
  - The whole first term is now  $A$ , the regularization term is now  $B$
  - We remove  $\lambda$  and introduce  $C = 1/\lambda$

My SVM cost function is  $= C \sum_{i=1}^m [(y^{(i)} * cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) * cost_0(\theta^T x^{(i)})] + 1/2 (\sum_{j=1}^n \theta_j^2)$

### Support vector machine

Logistic regression:

$$\min_{\theta} \frac{1}{m} \left[ \sum_{i=1}^m y^{(i)} \underbrace{(-\log h_\theta(x^{(i)}))}_{\text{cost}_1(\theta^T x^{(i)})} + (1 - y^{(i)}) \underbrace{(-\log(1 - h_\theta(x^{(i)})))}_{\text{cost}_0(\theta^T x^{(i)})} \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

Support vector machine:

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

$\min_u (u-5)^2 + 1 \rightarrow u=5$   
 $\min_u 10(u-5)^2 + 10 \rightarrow u=5$

$A + \lambda B \leftarrow$   
 $C A + B \leftarrow$   
 $C = \frac{1}{\lambda}$

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

Andrew B

- The hypothesis here is different because it does not output a probability. It just says that  $h_\theta(x) = 1$  if  $\theta^T x \geq 0$  or  $h_\theta(x) = 0$  otherwise

### SVM hypothesis

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

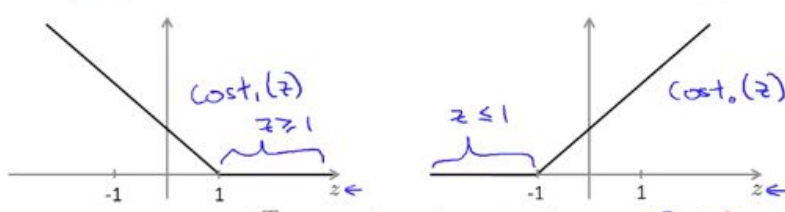
Hypothesis:

$$h_\theta(x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

## Large Margin Intuition

- Sometimes people talk about SVMs as large margin classifiers. We are going to talk about what that means. Taking our SVM cost function imagine we want to make each term small or zero. For the first term (left chart below),  $\text{cost}_1(z) = 0$  if  $z \geq 1$  and for the second term (right chart below)  $\text{cost}_0(z) = 0$  if  $z \leq -1$ . We are then saying we want  $\theta^T x \geq 1$  in one case and we want  $\theta^T x \leq -1$  in the other case.

### Support Vector Machine

$$\rightarrow \min_{\theta} C \sum_{i=1}^m \left[ y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$


$\rightarrow$  If  $y = 1$ , we want  $\theta^T x \geq 1$  (not just  $\geq 0$ )  $\theta^T x \geq 1$   
 $\rightarrow$  If  $y = 0$ , we want  $\theta^T x \leq -1$  (not just  $< 0$ )  $\theta^T x \leq -1$   
 $C = 100,000$

Now let's take a look at what happens if we set a huge value for our constant  $C$

- If we set a huge value for  $C$  we are highly motivated to make the multiplying term as close to zero since we are trying to minimize  $\theta$ . So now we need to think what it would take to make the term in the blue box = 0. We remember that whenever  $y^{(i)} = 1$   $\theta^T x \geq 1$  and whenever  $y^{(i)} = 0$   $\theta^T x \leq -1$ . We then end up with the following:  $\min (1/2) \sum_{j=1}^n \theta_j^2$  subject to  $\theta^T x \geq 1$  if  $y^{(i)} = 1$  and  $\theta^T x \leq -1$  if  $y^{(i)} = 0$

### SVM Decision Boundary

$$\min_{\theta} C \sum_{i=1}^m \left[ y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

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

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

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

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

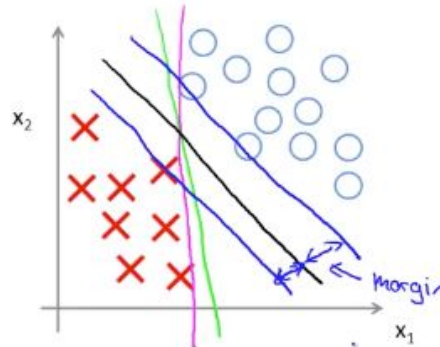
$$\min_{\theta} C + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

$\text{s.t. } \theta^T x^{(i)} \geq 1 \text{ if } y^{(i)} = 1$   
 $\theta^T x^{(i)} \leq -1 \text{ if } y^{(i)} = 0.$

- Let's see what kind of decision boundary we get with SVM. In the example above we have a case where the boundary can be linear and there are many solutions. SVM will

give us the black line below where we have a large margin between the training examples and the decision boundary. This fact lends a robustness to the algorithm

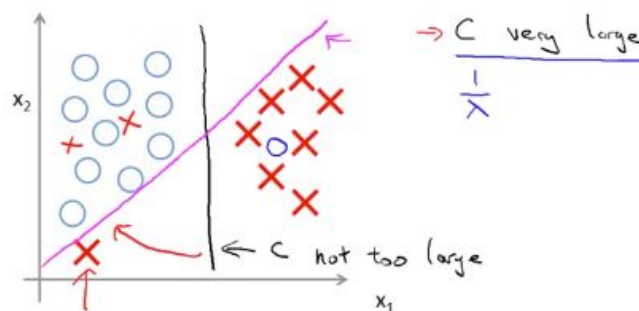
### SVM Decision Boundary: Linearly separable case



Large margin classifier

- How does SVM behave in the presence of outliers? If we have an outlier and  $C$  is very large then SVM will change its decision boundary to the magenta line below. However, changing our boundary in the presence of one outlier does not seem like a very good idea. If  $C$  is not too large we will stay with the black decision boundary below. In practice, when  $C$  is not very very large SVM will do a good job ignoring the outlier values and will do the 'right thing', i.e. keeping my original decision boundary instead of changing it. It will also do reasonable things if the data is not linearly separable.

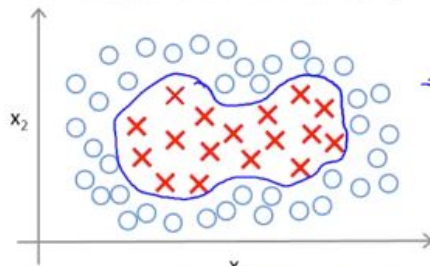
### Large margin classifier in presence of outliers



## Kernels I

- The main technique for adapting SVMs to develop complex linear classifiers is Kernels. We'll define those now. We start with the example below and we try to find a function that will create a decision boundary to fit it. For SVM, our hypothesis would be:  
 $y = 1$  if  $\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \dots \geq 0$  and 0 otherwise. We introduce a new term,  $f$ ,  
 $f_1 = x_1, f_2 = x_2, f_3 = x_1 x_2$  and so on.

## Non-linear Decision Boundary



Predict  $y = 1$  if

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

$$h_0(x) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \dots \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \dots$$

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

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

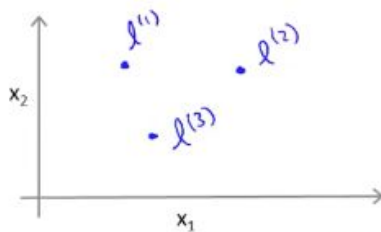
Andrew B

One question to ask is if we have a better choice of features than  $f_1, f_2, f_3$  since those are high order polynomials and are computationally expensive

- What would be a better choice for these features? Given  $x$ , we compute a new feature depending on its proximity to a few points that I choose. In the example below we select 3 points:  $\gamma_1, \gamma_2, \gamma_3$  which we call landmarks. Then we use the following function:

given  $x$ ,  $f_1 = \text{similarity}(x, \gamma_1) = \exp(-(\|x - l^{(1)}\|^2 / 2\sigma^2))$  and use it for the other landmarks.

## Kernel



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

$$\begin{aligned} \text{Given } x: \quad f_1 &= \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right) \\ f_2 &= \text{similarity}(x, l^{(2)}) = \exp\left(-\frac{\|x - l^{(2)}\|^2}{2\sigma^2}\right) \\ f_3 &= \text{similarity}(x, l^{(3)}) = \exp(\dots) \end{aligned}$$

$\nwarrow$   $\nearrow$   
 kernel (Gaussian kernels)  $k(x, l^{(i)})$

Andrew

This similarity is mathematically called a kernel and in this example we are using Gaussian Kernels. Kernels are also denoted as  $k(x, \gamma^{(i)})$

- Now let's take a look at what the kernels do. We start with the first landmark which is computed as shown below. If  $x \approx l^{(1)}$  we have  $\exp(0^2 / 2\sigma^2) \approx 1$ .  
If  $x$  far from  $l^{(i)}$  we have  $\exp(\text{large number} / 2\sigma^2) \approx 0$

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

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

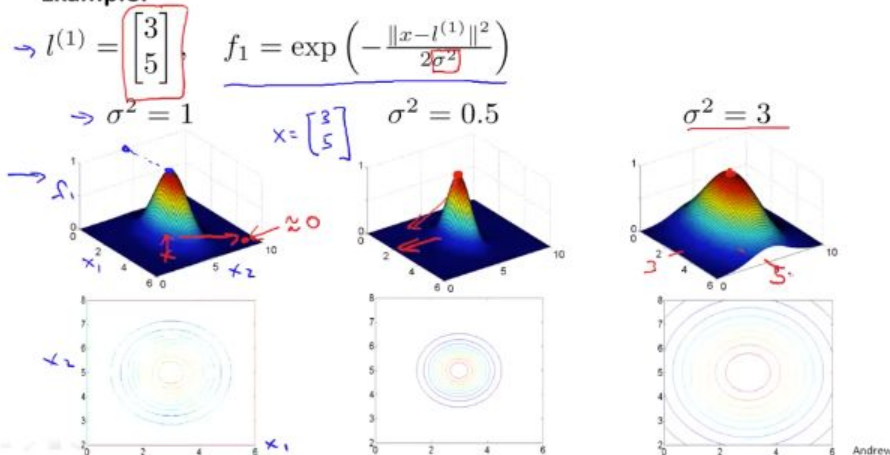
$$\begin{aligned} l^{(1)} &\rightarrow f_1 \\ l^{(2)} &\rightarrow f_2 \\ l^{(3)} &\rightarrow f_3 \end{aligned}$$

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

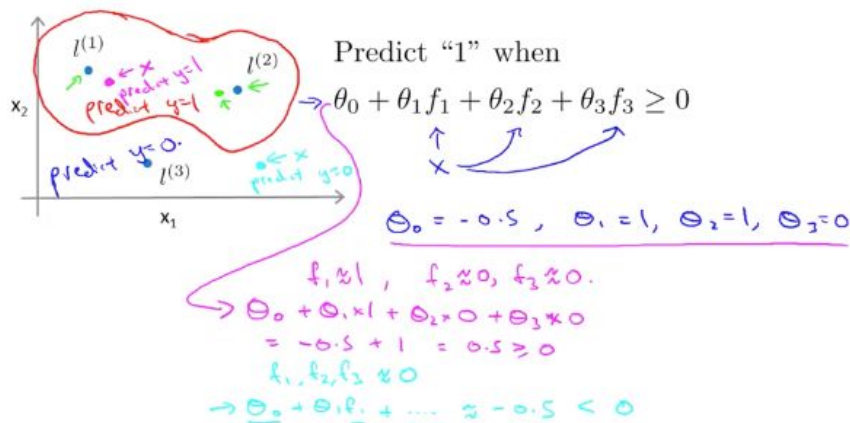
$$f_1 = \exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \approx 0.$$

- Now let's plot this similarity function to see how it looks like. Our  $l^{(1)}$  in this example is  $[3 \ 5]$ . If  $\sigma^2 = 1$  we get the first plot. At 3,5 we are at the top of the 'cone' and as we move away from  $l^{(1)}$  it 'travels down' the cone until it reaches zero. We can also see how this 'cone' changes if we vary the value of  $\sigma^2$ . If  $\sigma^2 = 0.5$  the cone becomes narrower and getting to zero is faster. If  $\sigma^2 = 3$  the cone becomes 'fatter' and getting to zero is slower

Example:



- Suppose we have the function below.



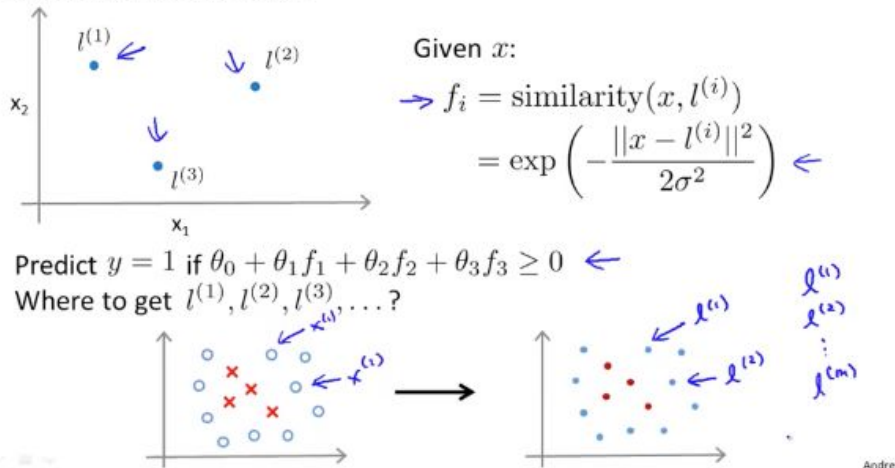


Also suppose we have the  $\theta$  values shown above. We first apply what we learned to a training example close to  $l^{(1)}$ . Using our equation we find that we end up with 0.5 which is bigger than 0 so  $h_\theta(x) = 1$ . We then pick the training example in cyan and end up with a value of -0.5 so  $h_\theta(x) = 0$ . If we do the same with other examples we find out that our decision boundary takes the shape shown above. Inside we predict 1, outside we predict 0.

## Kernels II

- Now we look at how do we choose the landmarks we talked about before. What we do is given a training set like the one below and create a landmark, as shown in the right chart, at right the same place where the training examples are. We then end up with  $m$  landmarks where  $m = \text{number of training samples}$ . This is nice because my features are going to measure how close an example is to one of the things I see on my training set.

### Choosing the landmarks



- Given we have  $x^{(m)}$  samples we have  $l^{(m)}$  landmarks. We can then build an  $f$  vector that contains all the similarity functions where  $f_1 = \text{similarity}(x, l^{(1)})$ ,  $f_2 = \text{similarity}(x, l^{(2)})$ .

### SVM with Kernels

- Given  $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$ ,
- choose  $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \dots, l^{(m)} = x^{(m)}$ .

Given example  $x$ :

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

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

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

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

$$x^{(i)} \in \mathbb{R}^{n+1} \text{ (or } \mathbb{R}^n) \rightarrow f^{(i)} = \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ f_2^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix}$$

$f_0^{(i)} = 1$

Our  $f$  vector then contains  $[f_0, f_1, f_2, \dots, f_m]$ . As always,  $f_0 = 0$

- Given that, we then have a new hypothesis that uses  $f$  and is written as  $h_\theta(x) = 1$  if  $\theta^T f \geq 0$  where  $\theta^T f = \theta_0 f_0 + \theta_1 f_1 + \dots + \theta_m f_m$ . How do we find the thetas? By solving the cost function we saw earlier and is shown below. The difference is that cost is now written as  $cost_1(\theta^T f^{(i)})$ .

### SVM with Kernels

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

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

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

Training:

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

$$-\sum_j \theta_j^2 = \theta^T \theta \leftarrow \theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_m \end{bmatrix} \quad (\text{ignoring } \theta_0)$$

$$-\theta^T M \theta \leftarrow \| \theta \|^2 \quad M = 10,000$$

A final implementation details is that for computational efficiency we change the definition of the regularization term to  $\theta^T M \theta$

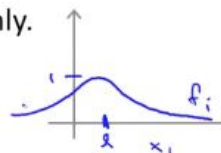
- How do you choose the parameters of the Support Vector Machine? When using SVM we need to choose  $C$  which is  $1/\lambda$ . If we use a large  $C$  it is equivalent to using a small  $\lambda$  which means we are not using much regularization and this translates in a hypothesis with high variance (overfit) and lower bias. Conversely, a small  $C$  means we are using a large  $\lambda$  and have a hypothesis with high bias (underfit) and low variance.

### SVM parameters:

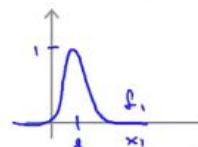
$C (= \frac{1}{\lambda})$ . → Large  $C$ : Lower bias, high variance. (small  $\lambda$ )  
→ Small  $C$ : Higher bias, low variance. (large  $\lambda$ )

$\sigma^2$  Large  $\sigma^2$ : Features  $f_i$  vary more smoothly.  
→ Higher bias, lower variance.

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



Small  $\sigma^2$ : Features  $f_i$  vary less smoothly.  
Lower bias, higher variance.



The other parameter we have to choose is  $\sigma^2$ . If we chose a large  $\sigma^2$  features  $f_i$  vary smoothly and we have a hypothesis with high bias and low variance. If we chose a small  $\sigma^2$ , features  $f_i$  vary less smoothly and we have a hypothesis with low bias and high variance



## Using an SVM

- Use software, don't implement it yourself, there are many libraries that already implement SVMs. When using one you have to specify:
  - C
  - Kernel
    - Linear Kernel = No Kernel: OK to use when n is large and m (number of training samples is small)
    - Gaussian Kernel: OK to use when m is large and n is small. In this case we also need to choose  $\sigma^2$ .

Use SVM software package (e.g. liblinear, libsvm, ...) to solve for parameters  $\theta$ .

Need to specify:

→ Choice of parameter C.

Choice of kernel (similarity function):

E.g. No kernel ("linear kernel")

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

$$\theta_0 + \theta_1 x_1 + \dots + \theta_n x_n \geq 0 \quad \rightarrow \quad \underline{n \text{ large}}, \quad \underline{m \text{ small}} \quad x \in \mathbb{R}^{n+1}$$

Gaussian kernel:

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

Need to choose  $\underline{\sigma^2}$ .

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

- Some languages will ask you to create the similarity function and others will provide it. When using a Gaussian Kernel we need to remember that we might have to use feature scaling to avoid having one feature dominate the hypothesis. In the example shown below we have  $x_1$  as the size of the house and therefore could be in a scale of thousands, we also have  $x_2$  which is the number of bedrooms which is in a scale of 1-10. To avoid the domination of  $x_1$  we use feature scaling.

**Kernel (similarity) functions:**

```

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

$x \rightarrow \begin{matrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{matrix}$

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

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

$$\|x - l\|^2 = (x_1 - l_1)^2 + (x_2 - l_2)^2 + \dots + (x_n - l_n)^2$$

$\underbrace{(x_1 - l_1)^2}_{1000 \text{ feet}^2} + \underbrace{(x_2 - l_2)^2}_{1-5 \text{ bedrooms}} + \dots$

- The 2 most common choices of kernel are the linear kernel and the gaussian kernel. There are other choices but all choices have to satisfy Mercer's theorem that makes sure all implementations run correctly. Some of the other kernel choices are
  - Polynomial Kernel: standard formula is  $(X^T l + \text{constant})^{\text{degree}}$ . Below are some examples of polynomial kernels.
  - Others are the string kernel, the chi-square kernel, histogram intersection kernel but they are rarely used. Professor has used each maybe once in his lifetime.

### 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:

$$k(x, l) = (x^T l + \text{constant})^{\text{degree}}$$

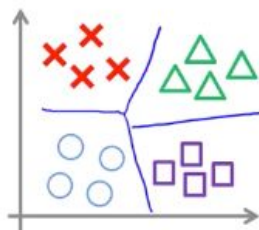
Handwritten examples:  $(x^T l)^2$ ,  $(x^T l + 1)^2$ ,  $(x^T l + 5)^2$

- More esoteric: String kernel, chi-square kernel, histogram intersection kernel, ...

$$\text{sim}(x, l)$$

- Choose the kernel that performs best in the cross validation data
- Multiclass Classification: Most SVMs already have multiclass classification built in. If not we then used the one-vs-all method we reviewed earlier in the course and we pick the class  $i$  with the largest  $(\theta^{(i)})^T x$ .

### Multi-class classification



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

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  $i = 1, 2, \dots, K$ ), get  $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(K)}$

Pick class  $i$  with largest  $(\theta^{(i)})^T x$

$$\begin{matrix} \uparrow & \uparrow & \dots & \uparrow \\ y=1 & y=2 & & y=K \end{matrix}$$

- When should you use logistic regression and when should you use SVM?
  - If  $n$  (number of features) is large relative to  $m$  (number of examples) use logistic regression or SVM without a kernel

- If  $n$  is small and  $m$  is intermediate (say 10-10,000) then use SVM with Gaussian kernel
- If  $n$  is small and  $m$  is large use logistic regression or SVM with no kernel

### 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.  $n \geq m$ ,  $n = 10,000$ ,  $m = 10 \dots 1000$ )

→ Use logistic regression, or SVM without a kernel ("linear kernel")

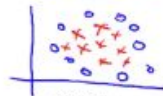
→ If  $n$  is small,  $m$  is intermediate: ( $n = 1-1000$ ,  $m = 10-10,000$ ) ←

→ 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.



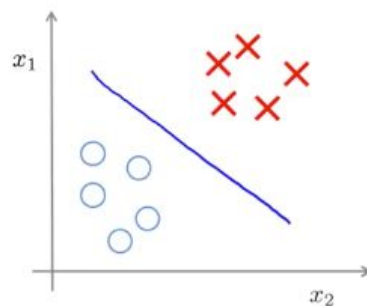
Neural Networks work well in these settings but will be slower to train. Finally, logistic regression and SVM with a linear kernel will give you very similar results.

## Week 7: Unsupervised Learning

### Introduction: Clustering

- This is the first time we start talking about Unsupervised Learning. In supervised learning problems we are given a labeled dataset and we have to find the decision boundary that separates positive label examples from negative label examples. In other words, given a set of labels we have to find a hypothesis that fits it.

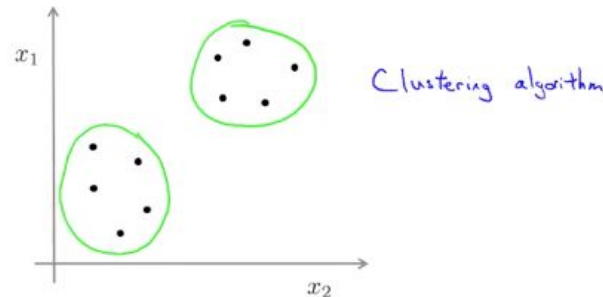
### Supervised learning



Training set:  $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \dots, (x^{(m)}, y^{(m)})\}$  ←

- In unsupervised learning, we are given data with no labels. The data looks like the one below. Here are a set of points  $x^{(1)}, x^{(2)}, x^{(3)} \dots x^{(m)}$  with no y labels. In this case we give this unlabeled data to the algorithm and ask it to find some structure or pattern in the data. Given the data set below we might have an algorithm that separate this dataset into two clusters. We are going to spend some time on clustering algorithms.

## Unsupervised learning



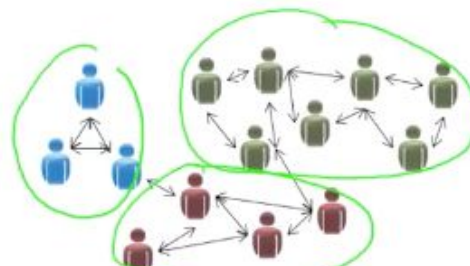
Training set:  $\{x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(m)}\}$  ←

- What is clustering good for? Various applications, some of them are shown below: market segmentation, social network analysis, organizing computing clusters and astronomical data analysis

## Applications of clustering



→ Market segmentation



→ Social network analysis



→ Organize computing clusters



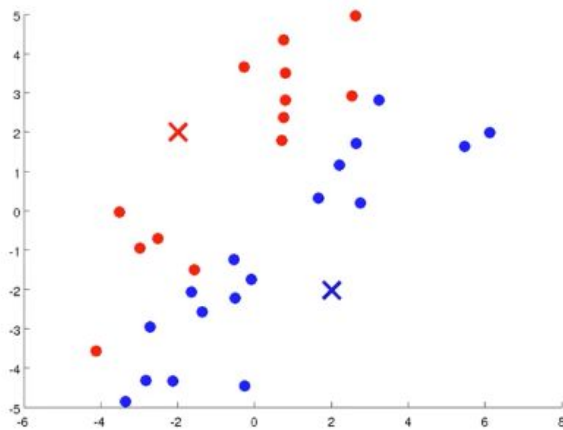
→ Astronomical data analysis

## K-Means Algorithm

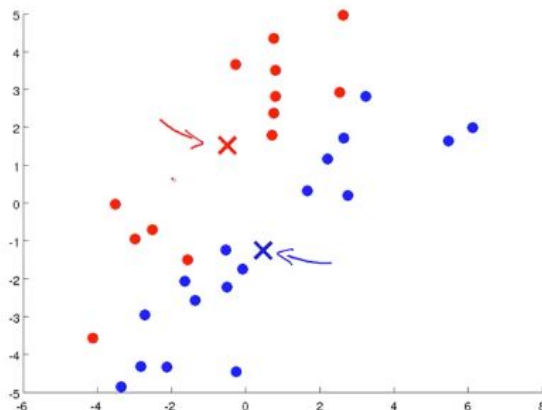
- K-Means is by far the most used clustering algorithm so we are going to talk about it now. Let's say I have an unlabeled data set like the one below and I want to group it into

two clusters. First step is to randomly initialize two points which we will call cluster centroids. After that we perform two steps:

- A cluster assignment step: the algorithm goes through all the dots/examples and depending on whether they are closer to the red or blue centroid assigns them to one of the two cluster centroids. In the example below that means coloring the examples from the original green color to either red or blue
- A move centroid step: In this step we take the two centroids and move them to the average of the points of the same color. So we take the mean of the location of all the red dots and move the red centroid there. We then do the same for the blue centroid

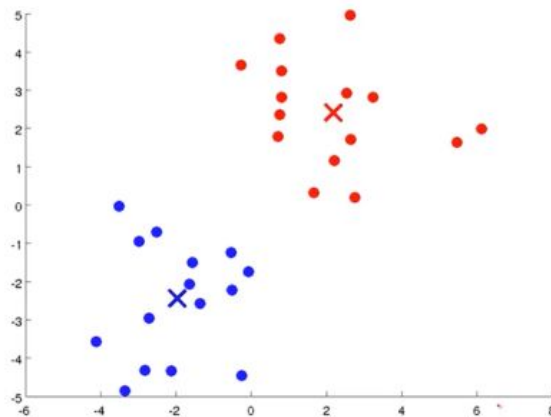


The new location of the centroids is shown below. Now they are in the new means.



- Then we do another cluster assignment step and recolor the dots based on their closeness to a centroid and the new chart is shown below. Some dots changed color. We then do another move centroid step. We do the whole sequence again and we end up with the final chart below. We can do another K-Means round but the clusters and centroids will not change





- Now we can define K-Means more formally: K-Means takes two inputs:
  - K: the number of clusters
  - A training set  $\{x^{(1)}, x^{(2)}, x^{(3)} \dots x^{(m)}\}$  and there is no  $x^{(0)}$  so the size of the training set is  $n$  and not  $n+1$

### K-means algorithm

Input:

- $K$  (number of clusters)  $\leftarrow$
- Training set  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$   $\leftarrow$

$$x^{(i)} \in \mathbb{R}^n \text{ (drop } x_0 = 1 \text{ convention)}$$

- More formal definitions: we randomly initialize  $K$  cluster centroids  $\mu_1, \mu_2, \dots, \mu_K$ . We then do a repeat loop of the 2 steps:
  - First we do the cluster assignment step where we are trying to minimize  $\|x^{(i)} - \mu_k\|$  and the result is  $C^{(i)}$  which tells us the cluster assigned to example  $i$

### K-means algorithm

Randomly initialize  $K$  cluster centroids  $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$

Repeat {

Cluster assignment step {

for  $i = 1$  to  $m$

$C^{(i)} :=$  index (from 1 to  $K$ ) of cluster centroid closest to  $x^{(i)}$

for  $k = 1$  to  $K$

$\mu_k :=$  average (mean) of points assigned to cluster  $k$

Move centroid

}  $\mu_2 = \frac{1}{4} [x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)}] \in \mathbb{R}^n$

$\mu_1$   $\mu_2$

$\min_k \|x^{(i)} - \mu_k\|^2$

$\rightarrow C^{(1)}=2, C^{(5)}=2, C^{(6)}=2, C^{(10)}=2$

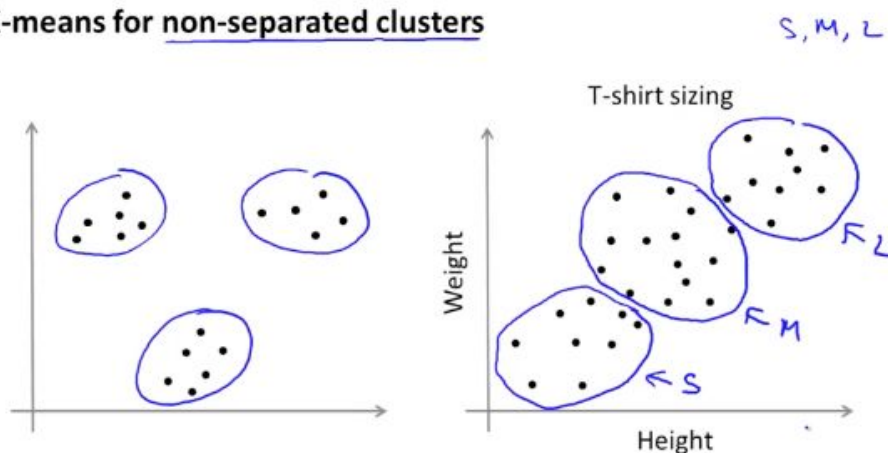
- Then we do the move centroid step where we average the mean of the points assigned to cluster  $k$ . For example, if we have examples

$x^{(1)}, x^{(5)}, x^{(6)}$  and  $x^{(10)} \Rightarrow c^{(1)} = 2, c^{(5)} = 2, c^{(6)} = 2, c^{(10)} = 2$  (meaning all those examples are assigned to cluster 2) the average  $\mu_2 = 1/4 (x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)})$

We could end up with a cluster centroid with no points assigned to it. In that case we just eliminate that cluster and we end up with  $k-1$  clusters

- One last thing: we can also apply K-Means to non-separable clusters as shown below on the right. We have been applying K-Means to separable clusters but the t-shirt example on the right is different. In this case a t-shirt manufacturer is trying to decide the t shirt sizes to make and has some data from the population on weight and height. K-Means might create the clusters shown below

### K-means for non-separated clusters



### Optimization Objective

- K-Means also has an optimization objective and a cost function to minimize. We will talk about it now.  $c^{(i)}$  = index of the cluster to which example  $x^{(i)}$  is currently assigned,  $\mu_k$  is the cluster centroid  $k$  and  $\mu_{c^{(i)}}$  is the cluster centroid of cluster to which  $x^{(i)}$  is assigned

#### K-means optimization objective

$\rightarrow c^{(i)}$  = index of cluster  $(1, 2, \dots, K)$  to which example  $x^{(i)}$  is currently assigned

$\rightarrow \mu_k$  = cluster centroid  $k$  ( $\mu_k \in \mathbb{R}^n$ )

$\mu_{c^{(i)}}$  = cluster centroid of cluster to which example  $x^{(i)}$  has been assigned

$x^{(i)} \rightarrow \underline{5} \quad c^{(i)} = \underline{5} \quad \mu_{c^{(i)}} = \mu_5$

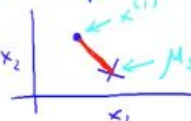
Optimization objective:

$$\rightarrow J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^m \boxed{\|x^{(i)} - \mu_{c^{(i)}}\|^2}$$

$$\rightarrow \min_{c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K} J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$$

$$\rightarrow \mu_1, \dots, \mu_K$$

Distortion



Andrew 1

Our cost function is defined as  $J(c^{(1)} \dots c^{(m)}, \mu_1, \dots, \mu_k) = 1/m \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$ . It is also sometimes called the distortion function

- Taking our algorithm definition we can mathematically prove that
  - The cluster assignment step is minimizing  $J(\dots)$  with regards to  $(c^{(1)} \dots c^{(m)})$  while holding  $\mu_1, \dots, \mu_k$  fixed
  - Similarly, the move centroid step is minimizing  $J(\dots)$  with regards to  $\mu_1, \dots, \mu_k$

### K-means algorithm

Randomly initialize  $K$  cluster centroids  $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$

Repeat {

Cluster assignment step  
Minimize  $J(\dots)$  w.r.t  $c^{(1)}, c^{(2)}, \dots, c^{(m)}$   
(holding  $\mu_1, \dots, \mu_k$  fixed)

for  $i = 1$  to  $m$   
 $c^{(i)} := \text{index (from 1 to } K \text{) of cluster centroid closest to } x^{(i)}$

Move centroid  
for  $k = 1$  to  $K$   
 $\mu_k := \text{average (mean) of points assigned to cluster } k$

} Minimize  $J(\dots)$  w.r.t  $\mu_1, \dots, \mu_k$

### Random Initialization

- We are going to talk about to initialize K-Means. There are many ways of initializing the cluster centroid but there is one recommended method. This method is:
  - $K$  should be less than  $m$
  - Randomly pick  $K$  training examples. In the example below  $K = 2$
  - Set  $\mu_1, \dots, \mu_k$  equal to these  $K$  examples

### Random initialization

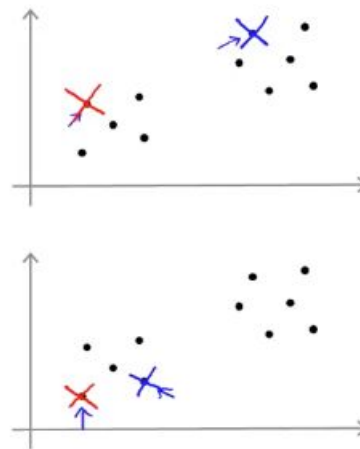
Should have  $K < m$

Randomly pick  $K$  training examples.

Set  $\mu_1, \dots, \mu_K$  equal to these  $K$  examples.

$$\mu_1 = x^{(i)}$$

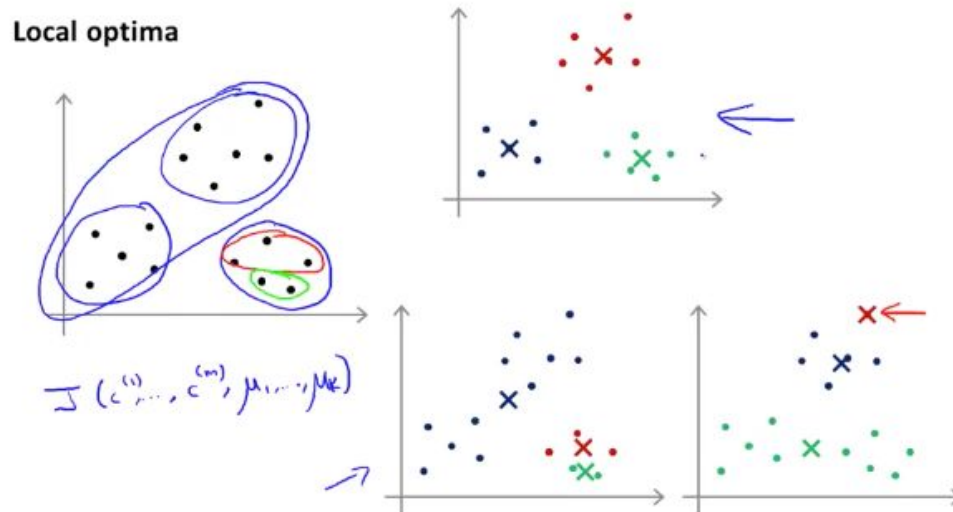
$$\mu_2 = x^{(j)}$$



The first chart above is a good example but we might not be so lucky all the times. The second chart is an example of where we start with not a very good initialization point.

These charts also hint at an issue: K-means can end up with different solutions and, furthermore, can end up with a local (not a global) optimum

- Below are some examples: the top one is the global optimum, the 2 low ones are examples of K-means getting stuck in a local optima. To minimize the possibility of K-means getting stuck in a local optimum what we do is we try multiple random initializations and we run it lots of times



- How do we do this? Below is an example. We do 100 initializations (common range is 50-1000) where we randomly initialize K-means, run it and then compute the distortion function. We end up with 100 cost functions and we pick the one with the lowest cost. This works very well for a small  $k$  (say  $k$  from 2 to 10) but does not work so well with larger  $k$  values.

### Random initialization

For  $i = 1$  to 100 { 50 - 1000

- Randomly initialize K-means.
- Run K-means. Get  $c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K$ .
- Compute cost function (distortion)
- $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$

}

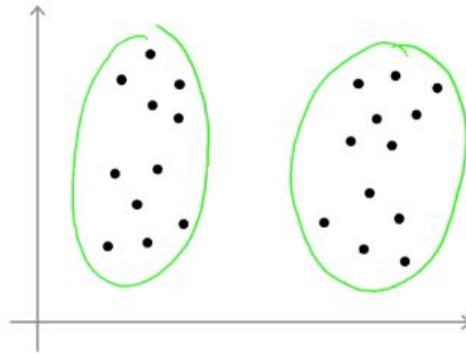
Pick clustering that gave lowest cost  $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$

$k = 2 - 10$

Choosing the number of clusters

- What is the right value of K? Most common way is to choose it by hand but we are going to look at other ways. A large part of the problem is because the number of cluster is ambiguous as shown below. Some people might see 2 clusters, some people might see 4 clusters.

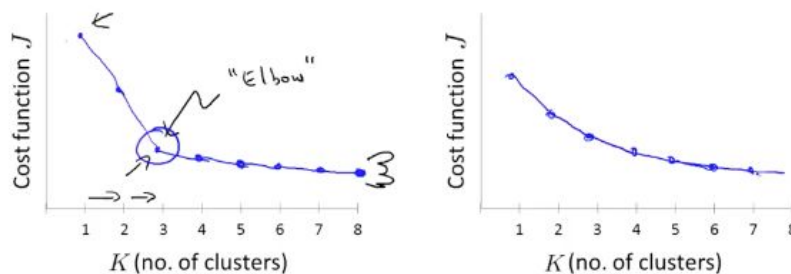
**What is the right value of K?**



- One way to choose the value of K is to use the Elbow method where we graph the cost function against different values of K. We might get a curve like the one on the left where it is clear where there is an 'elbow' so we can pick that elbow as our K value (3 in this case). However, this method is not very used in practice because sometimes we end up with a curve on the right. In that curve it is not clear if there is an elbow so K=3, K=4, K=5 might be good choices. Elbow is worth a shot but we should not have high expectations

**Choosing the value of K**

Elbow method:

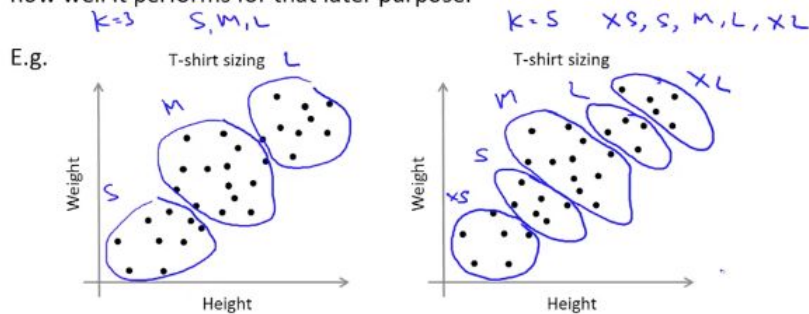


- We can also evaluate K based on how it's going to be used for some downstream purpose like t shirt sizing. In the example below we end up with 3 or 5 t-shirt sizes and a good questions to ask here is what would be better for the t-shirt business, 3 or 5 sizes?



### Choosing the value of K

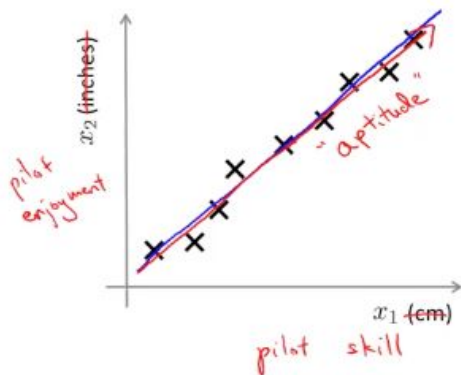
Sometimes, you're running K-means to get clusters to use for some later/downstream purpose. Evaluate K-means based on a metric for how well it performs for that later purpose.



### Dimensionality Reduction Motivation I: Data Compression

- We start talking about a second type of unsupervised learning problem, dimensionality reduction. To illustrate it let's take the case of the chart below where we have overlapping features (perhaps because one of them was delivered by one engineering tema and the other one from another engineering time). In this case it is the same feature but in one side measured in inches and on the other side measured in centimeters. A better example is plotting pilot skill against pilot enjoyment to come up with pilot 'aptitude'

### Data Compression

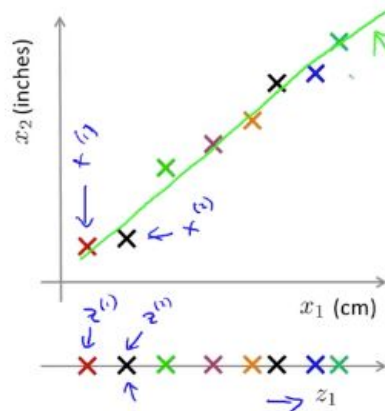


Reduce data from  
2D to 1D

- Reducing the dimension means creating the line shown below and projecting all the training examples on that line so we end up creating a new feature, feature  $z$  and all my training examples are on the  $z$  line. So  $x^{(1)}, x^{(2)} \dots x^{(m)}$  have a corresponding  $z^{(1)}, z^{(2)} \dots z^{(m)}$  but here just have one number (where as before we had two numbers). Summarizing, by projecting all the examples on that line now I just need one real number

which specifies where feature  $z$  lines on that line. This is great because it not only reduces memory requirements but also makes our learning algorithm runs faster

## Data Compression



Reduce data from  
2D to 1D

$$x^{(1)} \in \mathbb{R}^2 \rightarrow z^{(1)} \in \mathbb{R}$$

$$x^{(2)} \in \mathbb{R}^2 \rightarrow z^{(2)} \in \mathbb{R}$$

$\vdots$

$$x^{(m)} \rightarrow z^{(m)}$$

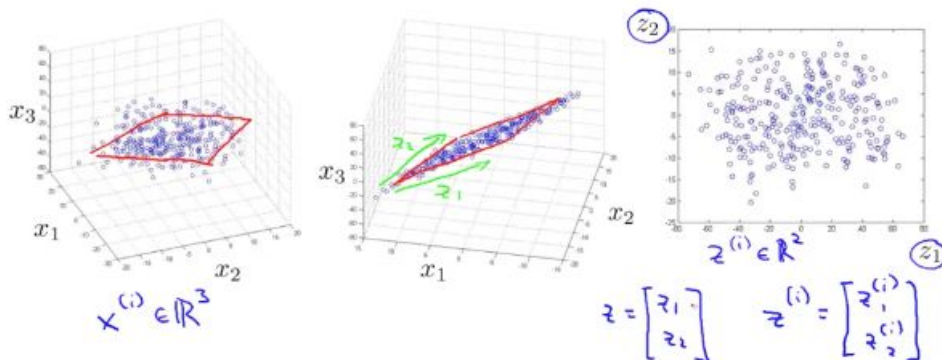
- The previous slide showed an example of reducing from 2 dimensions to 1 dimension. The chart below shows an example of reducing from 3 dimensions to 2 dimensions. The original training set is shown on the left side. We then project it into a single line, shown in the chart in the middle and from there we can eliminate one dimension and plot our training examples in two dimensions:  $z_1$  and  $z_2$ .  $z^{(i)}$  ends being a two dimensional vector:  

$$\begin{bmatrix} z_1^{(i)} & z_2^{(i)} \end{bmatrix}$$

## Data Compression

10000  $\rightarrow$  1000

Reduce data from 3D to 2D



## Dimensionality Reduction Motivation I: Visualization

- Another application of dimensionality reduction is to visualize the data, we will talk about it now. Here's an example: supposed we have a list of countries with a number of statistics for each one, about 50 numbers per country. It is hard to visualize 50 dimensions so we look for a way to reduce the number.

**Data Visualization**

$x \in \mathbb{R}^{50}$        $x^{(i)} \in \mathbb{R}^{50}$

	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	
Country	GDP (trillions of US\$)	Per capita GDP (thousands of intl. \$)	Human Develop- ment Index	Life expectancy	Poverty Index (Gini as percentage)	Mean household income (thousands of US\$)	...
→ Canada	1.577	39.17	0.908	80.7	32.6	67.293	...
China	5.878	7.54	0.687	73	46.9	10.22	...
India	1.632	3.41	0.547	64.7	36.8	0.735	...
Russia	1.48	19.84	0.755	65.5	39.9	0.72	...
Singapore	0.223	56.69	0.866	80	42.5	67.1	...
USA	14.527	46.86	0.91	78.3	40.8	84.3	...
...	...	...	...	...	...	...	...

[resources from en.wikipedia.org]

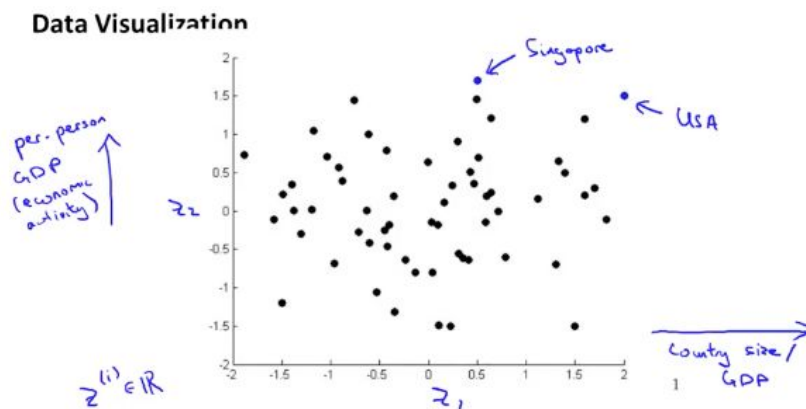
- What we did is then reduce from 50 dimensions to 2 dimensions for the table shown below. We can then plot the table below in two dimensions which is easier to visualize

### Data Visualization

Country	$z_1$	$z_2$
Canada	1.6	1.2
China	1.7	0.3
India	1.6	0.2
Russia	1.4	0.5
Singapore	0.5	1.7
USA	2	1.5
...	...	...

$z^{(i)} \in \mathbb{R}^2$   
Reduce data  
from 50D  
to 2D

- Plotting the data we end up with the chart below. We have axes  $z_1$  and  $z_2$  therefore  $z^{(i)}$  has two dimensions

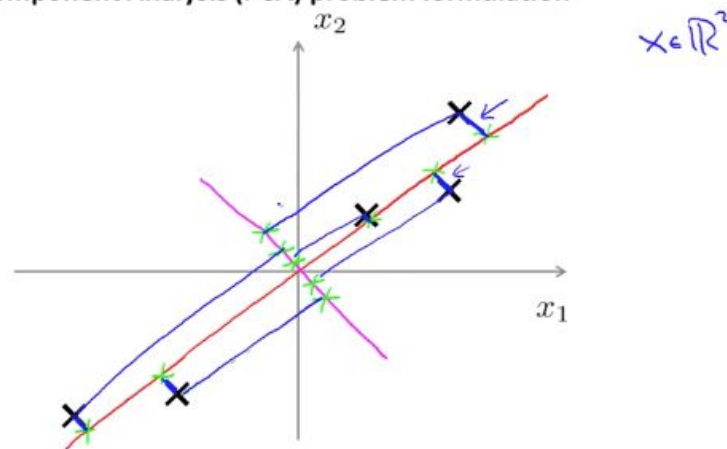


Having two dimensions allows to more succinctly capture what are the two main dimensions of the variations amongst countries

### Principal Component Analysis: Problem Formulation

- For dimensionality reduction the most common algorithm is Principal Component Analysis. Let's illustrate it with the example below. We have a set of training examples below and we want to come up with a line where we project those examples. The red line seems like a good choice because the difference between the original location and the new location of the examples (the blue lines) is short. In contrast, we can create another line, the magenta line, and if we project our examples there we can see that the blue lines are huge and, therefore, the errors (called projection errors) are huge. The red line is a better choice then.

#### Principal Component Analysis (PCA) problem formulation

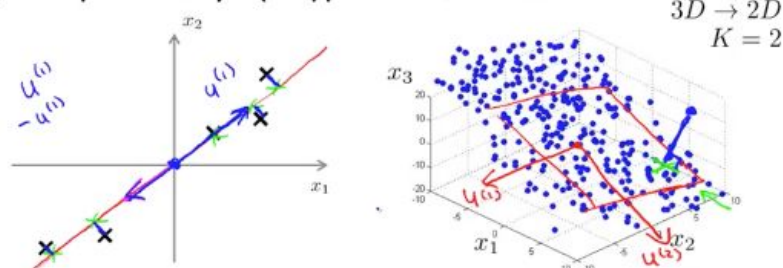


Andre

Before doing PCA we need to do feature scaling and normalization

- A more formal definition of PCA is shown below

#### Principal Component Analysis (PCA) problem formulation



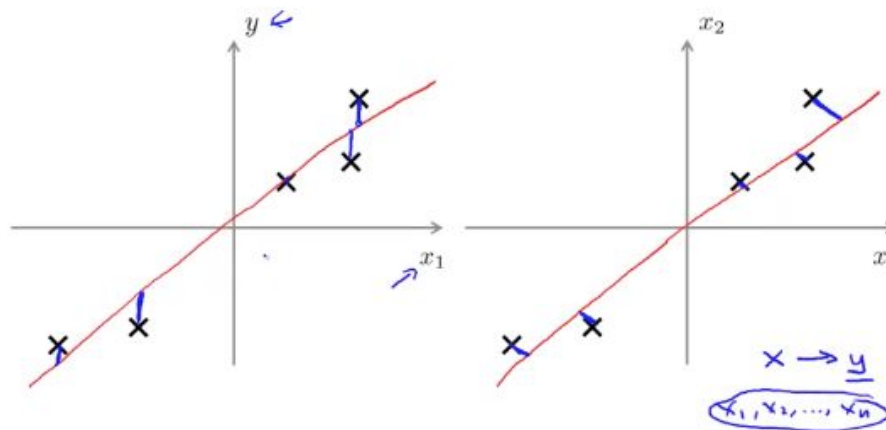
Reduce from 2-dimension to 1-dimension: Find a direction (a vector  $u^{(1)} \in \mathbb{R}^n$ ) onto which to project the data so as to minimize the projection error.

Reduce from n-dimension to k-dimension: Find  $k$  vectors  $u^{(1)}, u^{(2)}, \dots, u^{(k)}$  onto which to project the data, so as to minimize the projection error.

PCA tries to find a direction (a vector) onto which to project the data so the projection error is minimized. Above we have examples of reduction from 2 dimensions to 1 dimension and from 3 dimensions to 2 dimensions

- Sometimes people ask if PCA is similar to linear regression. The answer is no for two reasons:
  - First in linear regression we are trying to minimize the squared magnitude of the blue vertical lines (left chart) whereas PCA is trying to minimize the magnitude of the lines which are drawn at an angle (right chart)
  - The second reason is that in linear regression we have an x value and we are trying to predict the y value (y is then special). In PCA we don't have a y, we just have a vector  $x_1, x_2 \dots x_n$ , and all x features are treated the same

### PCA is not linear regression



- Summarizing, PCA is trying to find a lower dimensional surface in which to project the data so as to minimize the squared projection error.

### Principal Component Analysis: Algorithm

- We will now learn how to apply PCA. Before applying it we have a preprocessing step that involves feature scaling and mean normalization. The first step is to replace each  $x_j^{(i)}$  with  $x_j - \mu_j$  and the second is scale the features if they have different scales, i.e. one has number of rooms and the other one has house size. We scale it by replacing  $x_j$  with  $(x_j - \mu_j)/s_j$



## Data preprocessing

Training set:  $x^{(1)}, x^{(2)}, \dots, x^{(m)}$  ←

Preprocessing (feature scaling/mean normalization):

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}$$

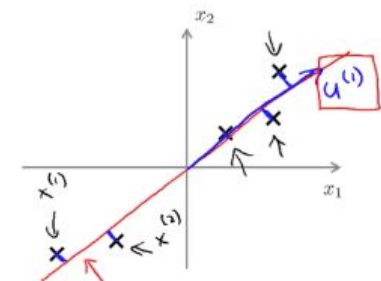
Replace each  $x_j^{(i)}$  with  $x_j^{(i)} - \mu_j$ .

If different features on different scales (e.g.,  $x_1$  = size of house,  $x_2$  = number of bedrooms), scale features to have comparable range of values.

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$$

- To refresh our memory, PCA aims to reduce dimensionality. In the examples below we are reducing dimensions from 2 to 1 (left chart) and 3 to 2 (right chart). What we do is we project the original training examples onto the red line below and now we only need one vector,  $u^{(1)}$ , and one number,  $z_1$ , to specify the position of an example. For 3 to 2 dimension reduction we need two vectors,  $u^{(1)}, u^{(2)}$  and  $z = [z_1 \ z_2]$

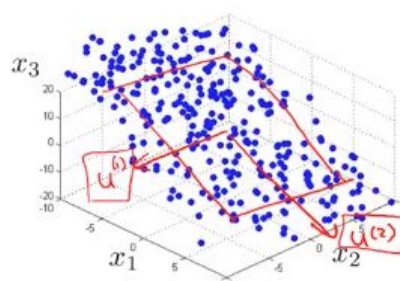
## Principal Component Analysis (PCA) algorithm



Reduce data from 2D to 1D

$$x^{(i)} \in \mathbb{R}^2 \rightarrow z^{(i)} \in \mathbb{R}$$

Below the equation, a 1D plot shows the projected points  $z^{(1)}, \dots, z^{(m)}$  along a horizontal axis labeled  $z_1$ .



Reduce data from 3D to 2D

$$x^{(i)} \in \mathbb{R}^3 \rightarrow z^{(i)} \in \mathbb{R}^2$$

$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

Given this, PCA needs to compute the vectors ( $u^{(1)}, u^{(2)}$ ) and the  $z$  vector. How do we compute them?

- To reduced from  $n$ -dimensions to  $k$ -dimensions we have to compute the covariance matrix and the eigenvectors of matrix  $\Sigma$

## Principal Component Analysis (PCA) algorithm

Reduce data from  $n$ -dimensions to  $k$ -dimensions

Compute "covariance matrix":

$$\Sigma = \frac{1}{m} \sum_{i=1}^m (x^{(i)})(x^{(i)})^T$$

Handwritten notes: "Sigma" and " $n \times n$ ".

Compute "eigenvectors" of matrix  $\Sigma$ :

$$[U, S, V] = \text{svd}(\text{Sigma});$$

Handwritten notes: "Singular value decomposition" and "eig(Sigma)".

$$U = \begin{bmatrix} | & | & | & | \\ u^{(1)} & u^{(2)} & u^{(3)} & \dots & u^{(k)} \\ | & | & | & | \end{bmatrix}$$

Handwritten notes: " $U \in \mathbb{R}^{n \times n}$ " and " $u^{(1)}, \dots, u^{(k)}$ ".

The covariance matrix *sigma* is defined as  $\Sigma = (1/m) \sum_{i=1}^n (x^{(i)})(x^{(i)})^T$  and we compute the eigenvectors using the octave command  $[U, S, V] = \text{svd}(\text{Sigma})$ ; Of the values returned we will get a matrix *u* with *n* vectors and we just take the first *k* vectors to reduce dimensions from *n* to *k*

- Once we have the *U* matrix we take the first *k* vectors and we end up with a matrix of dimensions *n* x *k*. *Z* is then computed as follows:  $z = u_{\text{reduce}}^T * x$ . By transposing  $u_{\text{reduce}}$  we end up with a matrix of dimensions *k* x *n* multiplied by example *x* which is of dimensions *n* x 1 so *z* ends up being of dimensions *k* x 1.  $u_{\text{reduce}}^T$  gives us the  $u^{(i)}$  values as rows

### Principal Component Analysis (PCA) algorithm

From  $[U, S, V] = \text{svd}(\text{Sigma})$ , we get:

$$\rightarrow U = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(n)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times n}$$

$x \in \mathbb{R}^n \rightarrow z \in \mathbb{R}^k$

$$z^{(i)} = \underbrace{\begin{bmatrix} | & | & \dots & | \\ u^{(1)} & u^{(2)} & \dots & u^{(k)} \\ | & | & \dots & | \end{bmatrix}^T}_{n \times k \text{ } U_{\text{reduce}}} x^{(i)} = \underbrace{\begin{bmatrix} -(u^{(1)})^T \\ \vdots \\ -(u^{(k)})^T \end{bmatrix}}_{k \times n} \underbrace{x^{(i)}}_{n \times 1}$$

Andrew N

- Summarizing, after mean normalization and optional feature scaling we compute sigma using a vectorized implementation in octave. We then use the svd function to get the matrix *U*. Next we take the first *k* columns of *U* using the octave command  $U(:, 1:k)$  and finally we compute *z* as follows:  $z = u_{\text{reduce}}^T * x$ .

### Principal Component Analysis (PCA) algorithm summary

→ After mean normalization (ensure every feature has zero mean) and optionally feature scaling:

$$\text{Sigma} = \frac{1}{m} \sum_{i=1}^m (x^{(i)})(x^{(i)})^T$$

→  $[U, S, V] = \text{svd}(\text{Sigma});$

→  $U_{\text{reduce}} = U(:, 1:k);$

→  $z = U_{\text{reduce}}' * x;$

$x \in \mathbb{R}^n$      ~~$x_0 = 1$~~

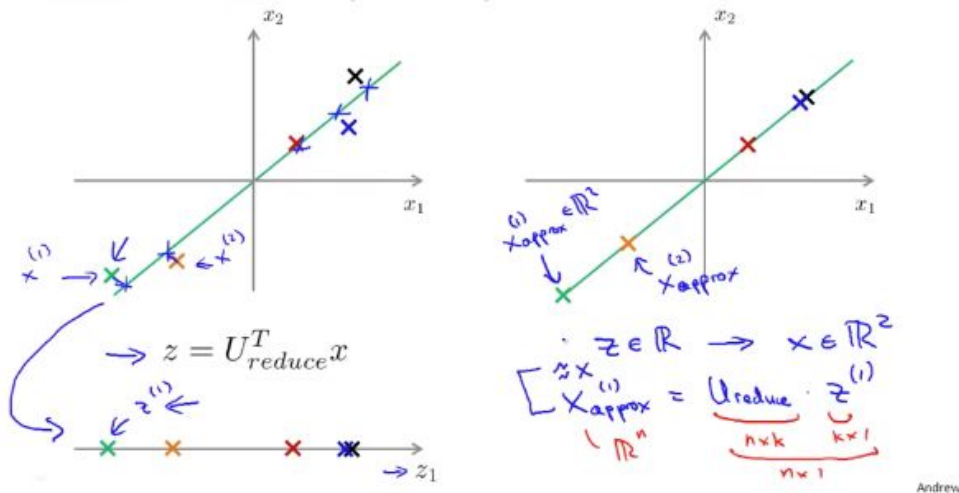
$X = \begin{bmatrix} -x^{(1)T} \\ \vdots \\ -x^{(m)T} \end{bmatrix}$ 

$\rightarrow \text{Sigma} = (1/m) * X' * X;$

## Applying PCA: Reconstruction from Compressed Representation

- We have been talking about PCA as an algorithm that 'compresses' by reducing dimensions. Since it is reducing dimensions there should also be a way to decompress back to the original dimensions. We will talk about that method now.

### Reconstruction from compressed representation



## Applying PCA: Choosing the number of Principal Components

- In PCA we are reducing from  $N$  dimensions to  $K$  dimensions,  $K$  is also called the number of principal components. We are going to talk about how we chose the value for  $K$ .

### Choosing $k$ (number of principal components)

Average squared projection error:  $\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2$

Total variation in the data:  $\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2$

Typically, choose  $k$  to be smallest value so that

$$\frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} \leq \frac{0.01}{0.10} \quad \frac{(1\%)}{(10\%)}$$

→ "99% of variance is retained"  
~~95%~~ to 90%

The average squared projection error is  $(1/m) * \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2$  and the total variation of the data is expressed as  $(1/m) * \sum_{i=1}^m \|x^{(i)}\|^2$  so a way to choose k is to divide average squared projection error over variance and pick the k that makes that value  $\leq 0.01$  (1%). It can go all the way down to 15%.

- A way to compute k (shown in the left size) is to start with k=1 and then do the PCA process: compute  $u_{reduce}, z^{(1)}, z^{(2)} \dots z^{(m)}$ , etc then check if our division is less than 0.01. If yes you are done, if not, try with k=2 and so on. This way is very inefficient so we try another method: when we call  $\text{svd}(\text{Sigma})$  we get a matrix S where only the diagonal has values. We use those values to compute  $1 - (\sum_{i=1}^k s_{ii} / \sum_{i=1}^n s_{ii})$  and we can use that value as our equivalent to check if that value is less than 1%. Conversely we can also check if

$$\sum_{i=1}^k s_{ii} / \sum_{i=1}^n s_{ii} \geq 0.99$$

**Choosing k (number of principal components)**

Algorithm:

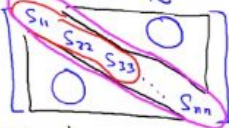
Try PCA with ~~k=1~~ ~~k=2~~ ~~k=3~~ ~~k=4~~ ...

Compute  $U_{reduce}, z^{(1)}, z^{(2)}, \dots, z^{(m)}, x_{approx}^{(1)}, \dots, x_{approx}^{(m)}$

Check if  $\frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} \leq 0.01?$

$k=17$

$\rightarrow [U, S, V] = \text{svd}(\text{Sigma})$

$\rightarrow S =$  

For given k  $1 - \frac{\sum_{i=1}^k s_{ii}}{\sum_{i=1}^n s_{ii}} \leq 0.01$

$\rightarrow \frac{\sum_{i=1}^k s_{ii}}{\sum_{i=1}^n s_{ii}} \geq 0.99$

Andrew N

- Summarizing, we choose k by using the svd command and then pick the smallest value of k that makes the inequality below true

**Choosing k (number of principal components)**

$\rightarrow [U, S, V] = \text{svd}(\text{Sigma})$

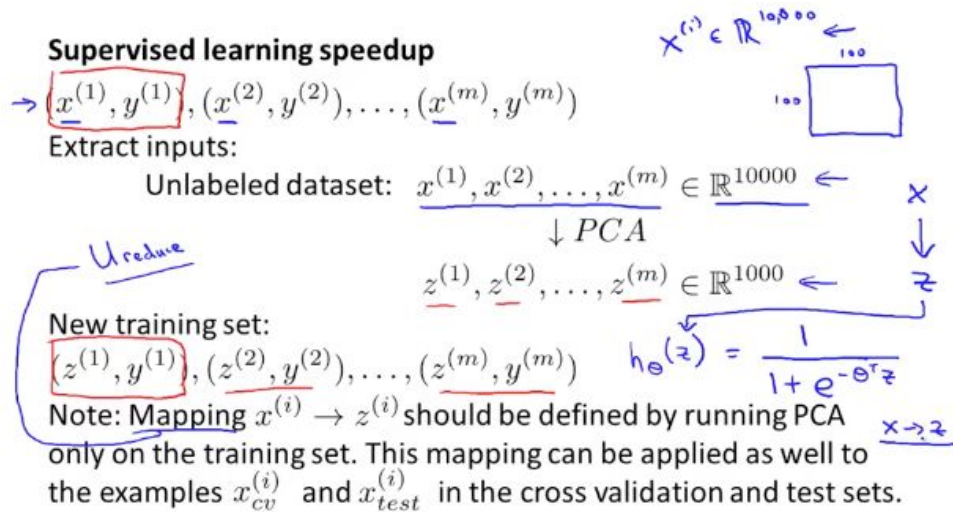
Pick smallest value of k for which

$\frac{\sum_{i=1}^k s_{ii}}{\sum_{i=1}^n s_{ii}} \geq 0.99$   $k=100$

(99% of variance retained)

Advice for applying PCA

- In previous classes we talked about how PCA can be used to speed up the running time of a learning algorithm. We'll now talk about what that means with a supervised learning example. Suppose we have a training set like the one below with  $x, y$  value pairs. We extract the  $x$  values and we end up with a unlabeled dataset with, in this case, 10,000 dimensions. We then apply PCA to produce a vector of  $z$  values with dimension 1,000. Our new training set is then composed of  $z$  and  $y$  values. We can then apply a supervised learning algorithm to this new training set (logistic regression shown below).



It's important to remember the  $x^{(i)} \rightarrow z^{(i)}$  mapping defined by PCA should only be applied to the training set. Once we found all the relevant parameters we can then apply the same mapping to other examples in the cross validation or tests sets

- Summarizing PCA can be applied to reduce memory needed to store data and to speed up a learning algorithm. In this case we choose  $k$  by deciding the % of variance that we will retain. Another use of PCA is visualization but that one  $k$  is either 2 or 3 since we can only comprehend 2 or 3 dimensional visualizations.

## Application of PCA

### - Compression

- Reduce memory/disk needed to store data
- Speed up learning algorithm ←

Choose  $k$  by % of variance retain

### - Visualization

$k=2$  or  $k=3$

- There is also a bad use of PCA: to prevent overfitting. The logic for using it to prevent overfitting is as follows: we use  $z^{(i)}$  instead of  $x^{(i)}$  to reduce the number of features from



n to k. Less features, less likely to overfit. Professor Ng does not recommend using it and prefers that we use regularization instead.

### Bad use of PCA: To prevent overfitting

→ Use  $z^{(i)}$  instead of  $x^{(i)}$  to reduce the number of features to  $k < n$ . — 10000

Thus, fewer features, less likely to overfit.

*Bad!*

This might work OK, but isn't a good way to address overfitting. Use regularization instead.

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \boxed{\frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2} \quad \leftarrow$$

- Here's another example where PCA shouldn't be used. Below is a project plan that has as the 2nd step PCA. Professor Ng recommends that we run the whole thing first without PCA and only if that doesn't do what we want then we can consider using PCA and  $z(i)$ .

### PCA is sometimes used where it shouldn't be

Design of ML system:

- - Get training set  $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$
- - ~~Run PCA to reduce  $x^{(i)}$  in dimension to get  $z^{(i)}$~~
- - Train logistic regression on  $\{(\cancel{z}^{(1)}, y^{(1)}), \dots, (\cancel{z}^{(m)}, y^{(m)})\}$
- - Test on test set: Map  $x_{test}^{(i)}$  to  $z_{test}^{(i)}$ . Run  $h_{\theta}(z)$  on  $\{(z_{test}^{(1)}, y_{test}^{(1)}), \dots, (z_{test}^{(m)}, y_{test}^{(m)})\}$

→ How about doing the whole thing without using PCA?

→ Before implementing PCA, first try running whatever you want to do with the original/raw data  $x^{(i)}$ . Only if that doesn't do what you want, then implement PCA and consider using  $z^{(i)}$ .