Density Estimation

-W9 - Anomaly Detection-

Intro:

- Anomaly detection is widely used in fraud detection (e.g. 'has this credit card been stolen?'). Given a large number of data points we may sometimes want to figure out which ones vary significantly from the average.
- For example we may want to detect defects on anomalies. We show how a dataset can be modeled using a Gaussian distribution, and how the model can be used for anomaly detection.

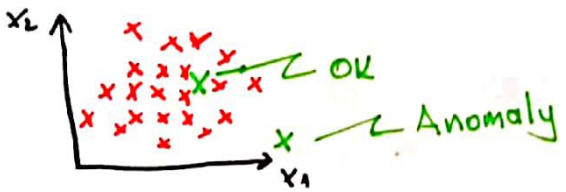
Anomaly Detection Example:

- Imagine that you are a manufacturer of aircraft engines, and you are doing some quality assurance testing and you measure some features like:
  - x_1 = heat generated
  - x_2 = vibration intensity
  - ⋮

We have a dataset:

$\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

- Let's say the data-set looks like:

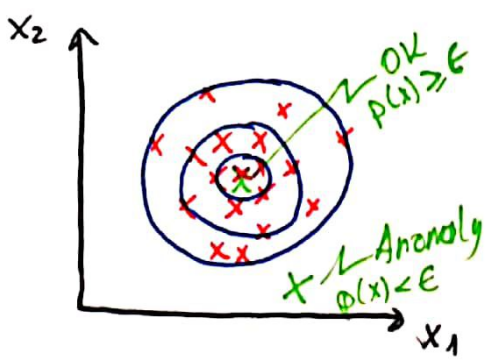


- Now let's say you have a new aircraft engine and it has some features: x_{test}. We want to know if this new engine is anomalous in any way. We want to know if it is an okay engine or maybe there is a problem

---

Density Estimation:

- For an anomaly detection problem we are given some dataset: $\{x^{(1)}, x^{(2)}, \dots x^{(m)}\}$ and we usually assume that these m examples are normal or non-anomalous examples and we want an algorithm to tell us if some new example x_{test} is anomalous



- The approach that we are gonna take is that, given this training set (unlabeled) we are going to build a model for $p(x)$.
- Build a model for the probability of x where x are these features x_1, x_2
- Then **if $p(x_{test}) < \varepsilon$** flag x_{test} as an **anomaly!**
- if **$p(x_{test}) \geq \varepsilon \rightarrow$ OK**

-W9 - Anomaly Detection - II -

Some Application Examples of Anomaly Detection:

- **Fraud Detection**
  - $x^{(i)}$ = features of user i's activities.
  - Model $p(x)$ from data.
  - Identify unusual users by checking which have $p(x) < \epsilon$
  - Some example features may be: x_1 = how often does user log in, x_2 = number of what pages visited, x_3 = number of transactions x_4 = number of posts, x_5 = typing speed of the user, etc...

- **Manufacturing**

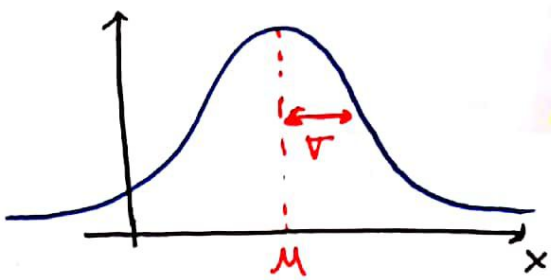
- **Monitoring computers in a data center and detect anomaly**
  - $x^{(i)}$ = features of each machine
  - features might be memory use, number of disk accesses/sec, CPU load, CPU load / network traffic.
  - Model $p(x)$
  - if for a test computer $p(x_{test}) \leq \epsilon$: there might be something wrong with the test computer.

-W9- Gaussian Distribution- or Normal Distribution

Gaussian (Normal) Distribution:

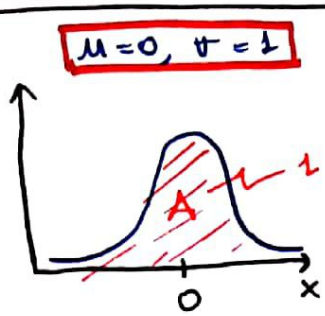
- Say x is a real valued random variable: $x \in \mathbb{R}$. If the probability distribution of x is gaussian with mean μ and variance σ^2 then it is showed as:
  - $$x \sim \mathcal{N}(\mu, \sigma^2)$$ ($\sim$: distributed as)

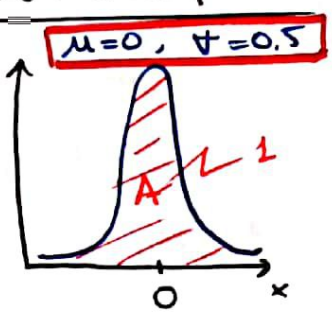
- If we plot the Gaussian Probability Density function:

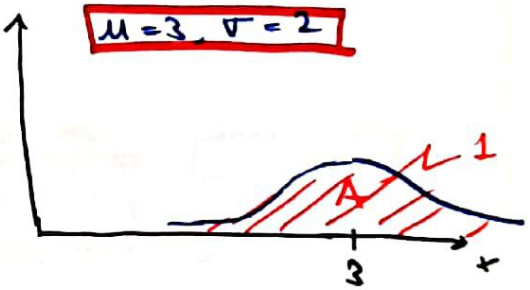


$$p(x) = p(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\cdot\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Some Gaussian Distribution Examples:

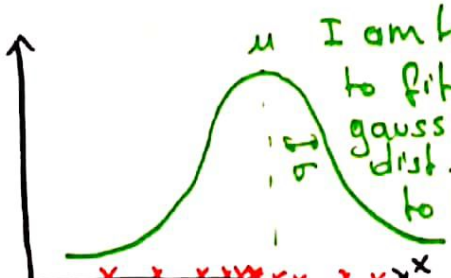






Parameter Estimation Problem:

- Let's say we have a dataset of m examples and each of these examples is a real number:
  - $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ and $x^{(i)} \in \mathbb{R}$



I am trying to fit a gaussian dist. func. to the data

- Let's say I suspect that these examples came from a Gaussian distribution
- I suspect each of my examples $x^{(i)} \sim \mathcal{N}(\mu, \sigma^2)$
- The problem of parameter estimation is, given my data set, I want to try to figure out what are the values of μ and σ^2
  → it is an approximation.

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

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (x^{(i)} - \mu)^2$$

* Maximum likelihood estimation of σ^2 we use m instead of $m-1$

Density Estimation _W9 - Anomaly Detection Algorithm-

- Let's say we an unlabeled training set: $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$
- Each example is $x \in \mathbb{R}^n$
- We need to model the $p(x)$ where x is a vector $\in \mathbb{R}^n$
  - $p(x) = p(x_1; \mu_1, \sigma_1^2) \cdot p(x_2; \mu_2, \sigma_2^2) \cdot \dots \cdot p(x_n; \mu_n, \sigma_n^2)$
  - we will assume each future is distributed according to a Gaussian Distribution:
    - $x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ • $x_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$ • $x_n \sim \mathcal{N}(\mu_n, \sigma_n^2)$

Note: The equation above corresponds to an independence assumption on the features x_1 through x_n. But in practice it turns out the algorithm which will be described works just fine whether or not these features are independent.

- $p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$ just a compact form of the above eq.

Put Everything Together: Anomaly Detection Algorithm

1) Choose features x_i that you think might be indicative of anomalous examples. $\{x^{(1)}, \dots, x^{(m)}\}$ → Dataset

2) Fit parameters $\mu_1, \dots, \mu_n, \sigma_1^2, \dots \sigma_n^2$

- $\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$
- $\sigma_j^2 = \frac{1}{m} \sum_{i=1}^{m} (x_j^{(i)} - \mu_j)^2$

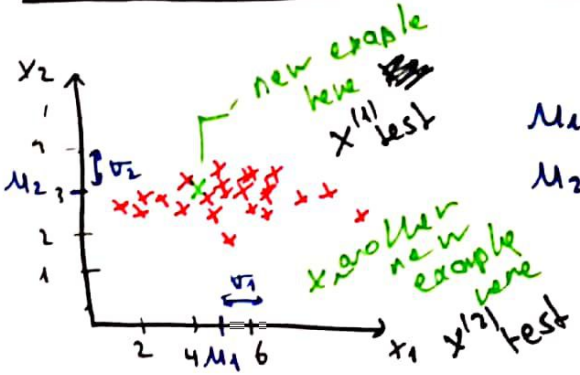
Find $\mu_1, \mu_2, \dots$

$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix}$, $\sigma = \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_n \end{bmatrix}$

3) Given new example x, compute $p(x)$:

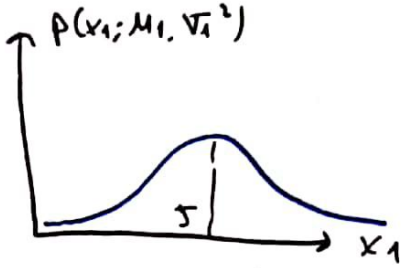
- $p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \cdot \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$
- Anomaly if $p(x) < \varepsilon$

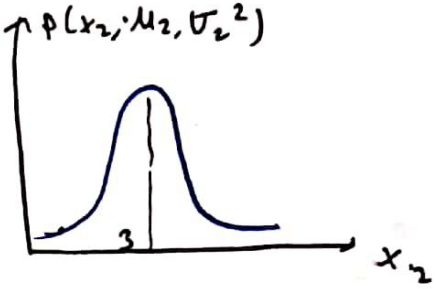
Anomaly Detection Example



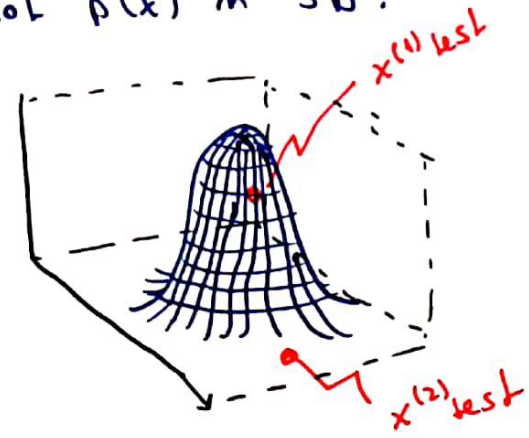
$\mu_1 = 5$, $\sigma_1 = 2$

$\mu_2 = 3$, $\sigma_2 = 1$





- Plot $p(x)$ in 3D:



- Let's choose

  $\varepsilon = 0.02$

- Later we'll see how to choose ε as well.

- $p(x^{(1)}_{test}) = 0.0426 \geq \varepsilon$ $x^{(1)}$ test is not an ANOMALY

  $p(x^{(2)}_{test}) = 0.0021 < \varepsilon$ $x^{(2)}$ test is an ANOMALY

-W9- Building an Anomaly Detection System -

Intro

- We will talk about how to develop a specific application of anomaly detection to a problem. In particular we will focus on the problem of how to evaluate an anomaly detection algorithm.

The Importance of real-number evaluation

- We've already talked about real-number evaluation.
- When developing a learning algorithm (choosing features, etc.), making decisions is much easier if we have a way of evaluating our learning algorithm.
- Thus, in order to be able to develop an anomaly detection system quickly, it would be really helpful to have a way of evaluating an anomaly detection system.
- In order to evaluate the A.D. System we need a labeled data of anomalous and non-anomalous examples. ($y=0$ if normal, $y=1$ if anomalous).
- We think our training set: $x^{(1)}, x^{(2)}, \dots, x^{(m)}$ as a large collection of normal / non-anomalous examples. But it is okay if there are few anomalies slip into your unlabeled training set.
- Next we are going to define our Cross-Validation and Test sets:
  - Cross Validation Set: $(x_{cv}^{(1)}, y_{cv}^{(1)}), \dots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$
  - Test Set: $(x_{test}^{(1)}, y_{test}^{(1)}), \dots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$
  - For both CV and Test sets we include some examples that we known to be anomalous

Aircraft Engines Motivating Example

- Let's say we have a data set like:
  - 10000 - as far as we know - good (normal) engines
  - 20 flawed engines (anomalous) ~ Typically b/w 2-50

~ it is okay if there are bad couple of engines inside since 10000. it is not going to affect the $p(x)$

- Given this data set a typical way to split it into training - CV and Test sets:
  - 6000 good (unlabeled) engines ← Training Set → Used for fitting $p(x)$
  - 2000 good engines ($y=0$), 10 anomalous ($y=1$) ← CV Set
  - 2000 good engines ($y=0$), 10 anomalous ($y=1$) ← Test Set

- So we are using Training Set to fit the $p(x)$. Then we can use CV to select features or select the ϵ, etc.. Finally after we fit our complete model we can test the performance of the final model on the test set.

- W9 - Building an Anomaly Detection System -

Algorithm Evaluation

→ Fit model $p(x)$ on training set $\{x^{(1)}, \dots, x^{(m)}\}$

→ On cross-validation or test sets, predict

$$y = \begin{cases} 1 & \text{if } p(x) < \varepsilon \quad \text{(anomaly)} \\ 0 & \text{if } p(x) \geq \varepsilon \quad \text{(normal)} \end{cases}$$

→ **Possible evaluation metrics:**

⊗ Since the data is skewed, because $y=0$ is much more common compared to $y=1$. Classification accuracy would not be a good evaluation metrics.

- Instead we can use evaluation metrics like:
  - → True positive, false positive, false negative, true negative
  - → Precision/Recall
  - → F_1-score

→ We can use CV set to choose ε. We can try many different values of ε and pick the value that maximizes F_1 score on the CV set.

→ We make decisions like "what features to use", "what value of ε should I use" by evaluating the algorithm on the CV set. When we find the final model we can evaluate it on the Test Set.

- In the last part we talked about the process of evaluating on anomaly detection algorithm. There, we used some labeled data with examples that we knew were either anomalous or not anomalous ($y=0$ or $y=1$)

(*) So the question then arises: If we have a labeled data why don't we just use a supervised learning algorithm? why don't we just use Logistic Regression or a Neural Network to try to learn directly from the labeled data?

Some Guidelines for when we should probably use an A.D. algorithm

Anomaly Detection vs Supervised Learning

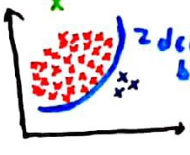
→ If we have very small number positive examples ($y=1$). (0-20 is common).
→ We will have relatively large number of negative ($y=0$) examples
↳ use it to fit $p(x)$

(*) There might be many different "types" of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like; future anomalies may look nothing like any of the anomalous examples we've seen so far.

→ Typically we have large number of positive and negative examples

(*) Enough positive examples for algorithm to get a sense of what positive examples are like, future positive examples ➡ likely to be similar to ones in the training set.

Note: Assume we've used a supervised learning alg. instead of A.D.. Since $y=1$ (blue) examples are only these three are not enough info. What if a new anomaly occurs like the green one. Then supervised model is not going to work properly.



→ For the spam classifier example, there were many different "types" of anomalies (spam mails) but we had enough positive examples (spam mails) so we've used a supervised learning.

Application Areas:

Anomaly Detection	vs	Supervised Learning
• Fraud detection (can shift if we have too many $y=1$ ex.s)		• Email spam classification
• Manufacturing		• weather prediction
• Monitoring machines in a data center		• Cancer classification
⋮		⋮

When we are applying anomaly detection, one of the things that has a huge effect on how well it does, is what features we choose.

Guidelines for Feature Selection:

→ One thing we did was to model the features using gaussian distribution, $p(x; \mu, \sigma^2)$. By plotting the histogram of every feature we can be sure that the distribution of the data follows the gaussian distribution. It usually work okay even if it's not gaussian but this one is a better practice. → Use hist command on matlab!

So if I have some features looks like doesn't follow the gaussian dist. I can manipulate these features and obtain new ones then I can use the new features for my Anomaly Detection Algorithm.

play with C to make it look like more gaussian ← 1/c

- $x_1 \leftarrow \log(x_1)$
- $x_2 \leftarrow \log(x_2 + c)$
- $x_3 \leftarrow \frac{1}{x_3}$

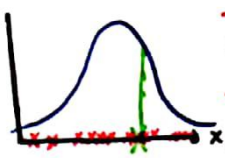
How Do we Come Up With Features for an Anomaly Detection Algorithm

→ What we want is: large $p(x)$ values for normal examples x. small $p(x)$ values for anomalous examples x.

→ The most common problem is: $p(x)$ is comparable (say, both large) for normal and anomalous examples.

Here we can benefit from the Error Analysis Procedure. It is very similar to the error analysis procedure we have seen for the supervised learning.

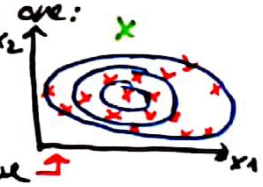
- Train a complek algorithm. Run it on the CV set and look at the example it gets wrong an see if we can come up with extra features to help the algorithm do better on the examples that it got wrong.



→ Say I have only 1 feature and a data set as shown. I am fitting a gaussian. And let's say I have an anomalous ex. as the green one:

→ $p(x)$ is quite high for the green ex. Thus alg. fails to classify.

→ Look at what went wrong for that particular example and come up with a new feature (another characteristic of an anomalous aircraft engine) Now I can come with a new figure



Choosing Features for Monitoring Computers in a Data Center

Choose features that might take on unusually large or small values in the event of an anomaly,

- memory use of computer (x_1)
- x_2 = number of disk accessed
- x_3 = CPU load
- x_4 = Network Traffic

Maybe x_3 and x_4 are linearly grow with each other then add another feature to distinguish anomaly

- x_5 = CPU load / Network Traffic or
- x_6 = (CPU load)^2 / Network Traffic

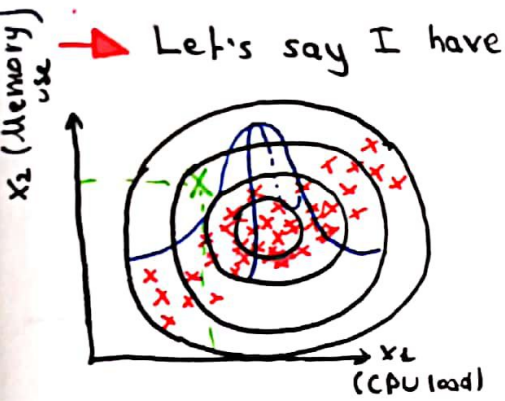
-W9- Multivariate Gaussian Distribution -

- OPTIONAL -

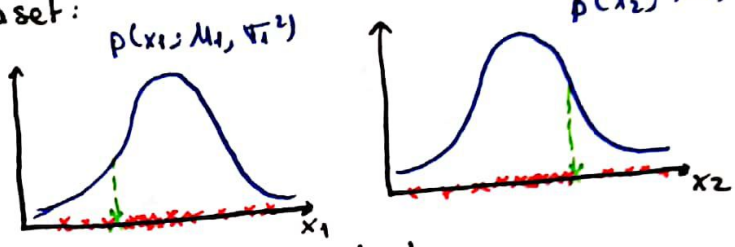
→ We'll talk about one possible extension of the Anomaly Detection Algorithm we have developed so far. This extension uses multivariate gaussian distribution and it has some adv.s and disadv.s. It can sometimes catch some anomalies that the earlier algorithm didn't.

Motivating Example: Monitoring Machines in a Data Center

→ Let's say I have my unlabeled data as plotted below:



→ If I try to fit a Gaussian $p(x)$ to this dataset:



→ $p(x)$ will be $p(x_1) \cdot p(x_2)$.

✳ Thus by defining $p(x) = p(x_1) \cdot p(x_2) \cdot \dots$ the Gaussian will always be ~~axis aligned ellipses~~. Thus we can not model the data as we want.

✳ For ex. the green example is actually an anomaly but since the gaussian is ~~not elliptical~~ axis-aligned our model fails to classify it as an anomaly.

→ To solve this problem and shape the gaussians more flexible way we will use Multivariate Gaussian (Normal) Distribution.

Multivariate Gaussian (Normal) Distribution

→ For features $x \in \mathbb{R}^n$.

→ Instead of modeling $p(x) = p(x_1) \cdot p(x_2) \cdot \dots$ seperately. We will model $p(x)$ all in one go.

→ For this new modeling the parameters will be:

$\mu \in \mathbb{R}^n$ (mean vector), $\Sigma \in \mathbb{R}^{n \times n}$ covariance matrix

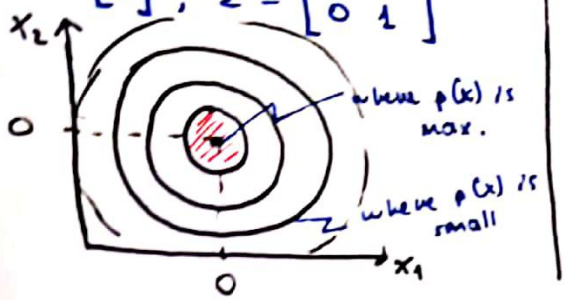
$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} \cdot |\Sigma|^{1/2}} \cdot \exp\left(-\frac{1}{2}(x-\mu)^T \cdot \Sigma^{-1} \cdot (x-\mu)\right)$$

$|\Sigma|$: Determinant of Σ

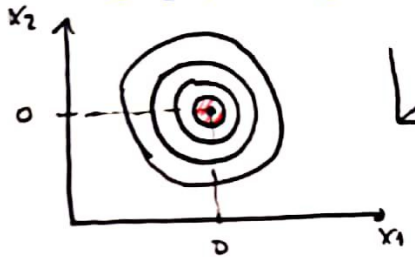
Now we know how to define a Multivariate Gaussian Distribution $p(x; \Sigma, \mu)$. Let's look at how do they look for different values of μ and Σ.

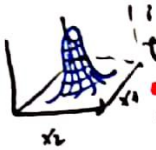
Multivariate Gaussian Examples

$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$



$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $\Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix}$

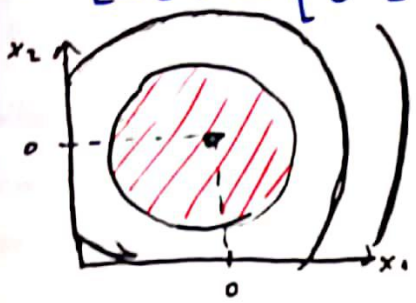




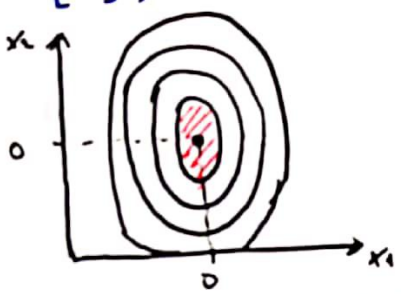
So when I shrunk the diagonal values it affects similarly.

- The height increases, σ_1 and σ_2 decreased

$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $\Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$



$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $\Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 1 \end{bmatrix}$



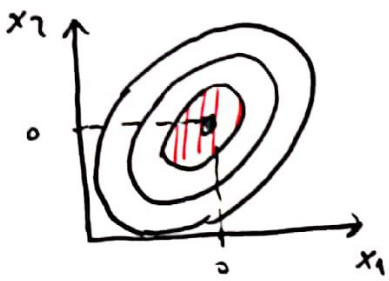
σ_1 is smaller than σ_2

We obtained an elliptic gaussian.

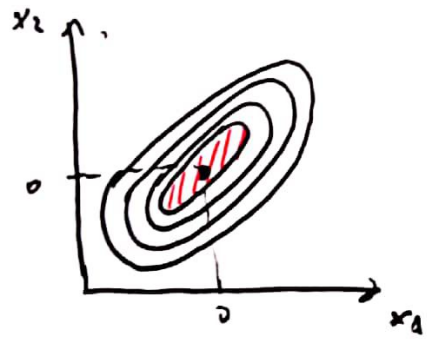
What happens if we change the off-diagonal values: We can model the correlated data.

If we increase the off-diagonal values it will affect the shape like we are pulling the gaussian from the ends in $x = y$ line:

$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $\Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$



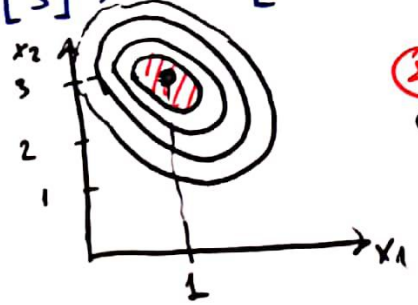
$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $\Sigma = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}$



Off-diagonals must be equal to the same value

$$\Sigma = \begin{bmatrix} \sigma_{x_1}^2 & \sigma_{x_1} \cdot \sigma_{x_2} \\ \sigma_{x_2} \cdot \sigma_{x_1} & \sigma_{x_2}^2 \end{bmatrix}$$

$\mu = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$, $\Sigma = \begin{bmatrix} 1 & -0.9 \\ -0.9 & 1 \end{bmatrix}$



If off-diags. are negative then we are pulling in the $x = -y$ line.

Recap Multivariate Gaussian (Normal) Distribution

- Parameters: μ, Σ where $\mu \in \mathbb{R}^n$: mean vector, $\Sigma \in \mathbb{R}^{n \times n}$: covariance matrix

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} \cdot |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \cdot \Sigma^{-1} \cdot (x-\mu)\right)$$

Parameter Estimation Problem

- Given training set: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ where $x^{(i)} \in \mathbb{R}^n$

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

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

→ Just like the sigma (Σ) in the PCA part.

Put All This Together: Anomaly Detection with the Multivariate Gaussian:

→ 1) Fit model $p(x)$ by setting μ and Σ using the training set.

→ 2) When given a new example x_{new}, compute $p(x_{new})$. Flag an anomaly if $p(x) < \varepsilon$

Relationship Between Multivariate Gaussian Dist. Model and the Original Model

→ Original model was: $p(x) = p(x_1; \mu_1, \sigma_1^2) \cdot p(x_2; \mu_2, \sigma_2^2) \cdot \ldots \cdot p(x_n; \mu_n, \sigma_n^2)$

→ It turned out that the original model is a special case ~~when~~ of Multivariate Gaussian when off-diagonals are always zero! So for the original model the contours of the gaussian is always axis-aligned

Axis Aligned: 

Not Axis-Aligned: 

→ Original model is the same as a Multivariate Gaus. Model but with a constraint and the constraint is that: $\Sigma =$ 

When Do we use Original Model and when Gaussian Mixture Model:

Original Model (used more often)	vs	Multivariate Gaussian Model
→ If anomaly occurs with the unusual combination of features like the example (2 pages before)		→ Automatically captures correlations between features.
→ We can use original model but we need to create (manually) features to capture anomalies. Like in the example of CPU load (3 pages before)		→ Computationaly more expensive
→ Computationaly cheaper (alternatively scales better to large n)		→ Lets say $n = 100000$ original model will work OK. But for this one $\Sigma \in \mathbb{R}^{n \times n}$ computing Σ^{-1} will be comp. exp.
→ Works okay even if m is small.		→ Must have $m > n$, or else Σ is non-invertible.
		→ In practice use Multiv. G. M when $m \geq 10 \cdot n$

If Σ is non-invertible either $m \leq n$ or we have redundant features: $x_1 = x_2$, $x_3 = x_4 + x_5$ get rid of x_2 and x_3

-W9 - Part 2 - Recommender Systems

→ Recommender systems look at patterns of activities between different users and different products to produce recommendations. In this 2nd module of W9, we introduce recommender algorithms such as the collaborative filtering algorithm and low-rank matrix factorization.

→ Applications on websites like Amazon, Netflix, Ebay etc. looks at what movies/products you've rated/purchased in the past and recommend new movies/products.

☹☹☹ During this module we will also talk about something quite important. We've already seen that features have a big effect on the performance of the learning algorithm. => For some problems there are algorithms that can try to automatically learn a good set of features for you. Recommender systems are just a one example of these problems.

Example: Predicting Movie Ratings

→ Let's say you are a company that sells movies and you let your users to rate your movies from 0 stars to 5 stars

→ Assume I have a data set as below:

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)
Love at Last	5	5	0	0
Romance Forever	5	?	?	?
Cute Puppies of Love	?	4	0	?
Car Chases	0	0	5	4
Swords vs. Karate	0	0	5	?

↳ looks like Alice and Bob likes romance

↳ looks like Carol and Dave likes action movies

→ Let's introduce a new notation:

- n_u: # of users (4 here)
- n_m: # of movies (5 here)
- $r(i,j)$: If user j has rated movie i then it is equal to 1. o/w 0.
- $y(i,j)$: Rating given by user j to movie i (defined only if $r(i,j)=1$.)

→ Our purpose is to guess the values of ? data.

→ Here I have a small number of users and movies and only a small portion of the movies are unrated (?) for a user. In reality this fraction will be much bigger.

→ Our first approach to building a recommender system will be the content based recommendations.

Content Based Recommender Systems

Movie	Alice (1) $\theta^{(1)}$	Bob (2) $\theta^{(2)}$	Carol (3) $\theta^{(3)}$	Dave (4) $\theta^{(4)}$	x_1 (romance)	x_2 (action)
Love	5	5	0	0	0.9	0
Romance	5	?	?	0	1.0	0.01
Cute Love	?	4	0	?	0.99	0
Can chase	0	0	5	?	0.1	1.0
Karak	0	0	5		0	0.9

- $n_u = 4$
- $n_m = 5$
- $n = 2$

→ Let's say I have 2 features for each movie. Now for each user we consider a linear regression problem. For each user we have some data set and we need to find a set of parameters θ that ~~fits~~ $\theta_0 \cdot x_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2$ fits our data set.

→ Each movie can be represented with a feature vector. Just like houses.

- The first movie: Love can be represented as $x^{(1)} = \begin{bmatrix} x_0^{(1)} \\ x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} = \begin{bmatrix} 1 \\ 0.9 \\ 0 \end{bmatrix}$

→ For each user j, learn a parameter $\theta^{(j)} \in \mathbb{R}^3$ ($\mathbb{R}^{n+1}$). Predict user j as rating movie i with $(\theta^{(j)})^T \cdot x^{(i)}$ stars.

→ Diyelim ki Alice için θ parametresini yani $\theta^{(1)}$'i elde ettik. (Bunu nasıl elde edeceğimizi sonra göreceğiz.) Elde edilen parametre yardımıyla da 3. film olan Cute Love'ın Alice'den kaç puan alacağını bulmak istiyoruz:

- $x^{(3)} = \begin{bmatrix} 1 \\ 0.99 \\ 0 \end{bmatrix}$ - $\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$ $\Rightarrow (\theta^{(1)})^T \cdot x^{(3)} = 5 \times 0.99 = 4.95$ ///

→ Özet olarak movie rating'i tahmin etme problemini linear regression problemine çevirdik. Her filmi 2 feature ile temsil ediyoruz. Her user için ayrı bir linear regression problemi olduğunu düşün. Line fitting yapacağız ama her user için datasel farklı dolayısıyla farklı bir line oturtmalıyız.

Devam ediyor...

W3-Part 2 - Recommender Systems III -

Problem Formulation:

- $r(i,j) = 1$ if the user has rated movie i (0 otherwise)
- $y(i,j)$ = rating by user j on movie i (if defined)
- $\theta^{(j)}$ = parameter vector of user j
- $x^{(i)}$ = feature vector of movie i → $(\theta^{(j)})^T \cdot x^{(i)}$: Rate for movie i, user j.
- $m^{(j)}$ = number of movies rated by user j.

$\theta^{(j)} \in \mathbb{R}^{n+1}$

(4): TO LEARN $\theta^{(j)}$:

minimize cost function just like in the part of linear regression

$$\min_{\theta^{(j)}} \frac{1}{2m^{(j)}} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T \cdot x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2m^{(j)}} \cdot \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

(sum over every rated movie; reg. term)

→ To simplify the eq. I will get rid of the $m^{(j)}$ terms. The result will be same.

Optimization Objective:

To learn $\theta^{(j)}$ (parameter for user j):

$$\min_{\theta^{(j)}} \frac{1}{2} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T \cdot x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

To learn $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$:

$$\min_{\theta^{(1)}, \dots, \theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T \cdot x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

$$= J(\theta^{(1)}, \dots, \theta^{(n_u)})$$

We can minimize the Cost Function J by Gradient Descent:

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T \cdot x^{(i)} - y^{(i,j)} \right) \cdot x_k^{(i)} \quad (\text{for } k = 0)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \underbrace{\left(\sum_{i:r(i,j)=1} \left((\theta^{(j)})^T \cdot x^{(i)} - y^{(i,j)} \right) \cdot x_k^{(i)} + \lambda \theta_k^{(j)} \right)}_{\frac{\partial}{\partial \theta_k^{(j)}} J(\theta^{(1)}, \dots, \theta^{(n_u)})} \quad (\text{for } k \neq 0)$$

→ We can use some advanced opt. alg.s instead of Grad. Desc.

→ This approach is called content based approach because we assume that we have features available to us. Features capture what is the content of these movies.

W9 - Collaborative Filtering -

→ The algorithm that we are going to talk about has very interesting property that it does: feature learning.

→ Data set'i hatırlarsan her filmi x_1 ve x_2 feature'ı ile temsil etmiştik. Ayrıca bu feature'ların her film için değerlerini bildiğimizi kabul ettik. Ancak gerçekte bu feature'ları bilmemiz zor. Birilerinin filmleri tek tek izleyip feature'ları ataması gerekli bu da zor ve zaman alıcı. O zaman data set şöyle olsun:

Movie	Alice(1) $\theta^{(1)}$	Bob(2) $\theta^{(2)}$	Carol(3) $\theta^{(3)}$	Dave(4) $\theta^{(4)}$	x_1 (romance)	x_2 (action)
Love	5	5	0	0	?	?
Romance	5	?	?	0	?	?
Cute Love	?	4	0	?	?	?
Car chase	0	0	5	4	?	?
Karate	0	0	5	?	?	?

→ Let's assume each use has told us how much they like romantic movies and how much they like action movies:

- $\theta^{(1)} = [0\ 5\ 0]^T$ • $\theta^{(2)} = [0\ 5\ 0]^T$ • $\theta^{(3)} = [0\ 0\ 5]^T$ • $\theta^{(4)} = [0\ 0\ 5]^T$
- Carol for ex. likes action movies thus $\theta_3^{(3)} = 5$ and doesn't like romance.

→ Bu durumda problem verilen θ'ları kullanarak $x^{(1)}, x^{(2)}, \dots x^{(nm)}$'i bulmak oluyor. Mesela 1. filme Alice ve Bob 5 vermiş o zaman $x_1^{(1)}$ 1 olabilir çünkü Alice ve Bob Romance çok seviyor. Matematiksel olarak yazarsak:
↳ Ama olmayada bilir yani romance'i 5'lik seviyor ama bu filme 3 vermiş olabilir.

→ What feature vector should $x^{(1)}$ be so that $(\theta^{(1)})^T \cdot x^{(1)} \approx 5$ and $(\theta^{(2)})^T \cdot x^{(1)} \approx 5$ and $(\theta^{(3)})^T \cdot x^{(1)} \approx 0$ and $(\theta^{(4)})^T \cdot x^{(1)} \approx 0$ • $x = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$

Optimization Algorithm: Let's formalize above problem:

Given $\theta^{(1)}, \dots, \theta^{(nu)}$, to learn $x^{(i)}$:

$$\min_{x^{(i)}} \frac{1}{2} \sum_{j:r(i,j)=1} \left((\theta^{(j)})^T \cdot x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (x_k^{(i)})^2$$

↳ choose $x^{(i)}$ to minimize error.

Given $\theta^{(1)}, \dots, \theta^{(nu)}$ to learn $x^{(1)}, \dots, x^{(nm)}$:

$$\min_{x^{(1)}, \dots, x^{(nm)}} \frac{1}{2} \sum_{i=1}^{nm} \sum_{j:r(i,j)=1} \left((\theta^{(j)})^T \cdot x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{i=1}^{nm} \sum_{k=1}^{n} (x_k^{(i)})^2$$

-W9- Collaborative Filtering - II

→ Putting everything together: the alg. we talked about 2 pages ago and the algorithm we talked 2 page ago:

Collaborative Filtering

→ Previously we have seen: Given $x^{(1)}, \ldots, x^{(n_m)}$ (and movie ratings) we can estimate $\theta^{(1)}, \ldots, \theta^{(n_u)}$.

→ What we've also seen: Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$ (and movie ratings) we can estimate $x^{(1)}, \ldots, x^{(n_m)}$.

⛤ Here is how collaborative filtering works:

- Randomly guess θ
- Based on guessed θ estimate x values for different movies
- Then using these x features estimate θ (a better one)
- So on... Keep iterating $\theta \to x \to \theta \to x \to \theta \to x \ldots$

→ For this problem this process is possible when each user rates multiple movies and each movie is rated by multiple users. So we can do this back and forth process.

⛤ This is just an initial collaborative filtering algorithm. Daha sonra daha efficient bir versiyonunu göreceğiz. Collaborative demenin sebebi bir çok user'ın verisinin kullanılarak herhese daha iyi önermeler yapabiliyor olmamız. Every user is helping the algorithm to learn better features and these features will be used to make better predictions for everyone else.

→ Özetle: Feature'ları bilirsek sütun bazında tahmin yapabileceğimiz θ değerlerini linear reg. ile elde edebiliyoruz. Benzer şekilde θ'lar bilindiğinde satır bazında tahmin yapabileceğimiz x değerleri elde edilebilir. Yani Bir kullanıcının oyladığı filmlere bakarak yeni bir filme kaç puan vereceğini veya bir film için hangi kullanıcıların ne oy verdiğine bakarak yeni bir kullanıcının ne oy vereceğini tahmin ediyoruz.

Random θ ile başlayıp $x \to \theta \to x \to \theta$ pattern'i izleyerek θ ve x'i optimize edebiliyoruz.

W9 - Collaborative Filtering Alg. III -

→ By putting previous ideas together we'll come up with a collaborative filtering algorithm:

Collaborative Filtering Optimization Algorithm

→ Given $x^{(1)}, \dots, x^{(n_m)}$, estimate $\theta^{(1)}, \dots, \theta^{(n_u)}$:

$$\min_{\theta^{(1)},\dots,\theta^{(n_u)}} \frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^{(j)})^2$$

ikisi de aynı her rated movie ve user i,j pair için toplama yapılıyor. Biri satır diğeri sütun sütun

→ Given $\theta^{(1)}, \dots, \theta^{(n_u)}$, estimate $x^{(1)}, \dots, x^{(n_m)}$

$$\min_{x^{(1)},\dots,x^{(n_m)}} \frac{1}{2}\sum_{i=1}^{n_m}\sum_{j:r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^{(i)})^2$$

→ Collaborative Filtering Alg. için sıralı θ ve x arasında git-gel yapabilecegimizi görmüştük böylece random bir θ ile başlayıp θ ve x'i optimize edebilirdik.

→ It turns out that there is a more efficient algorithm that doesn't need to go back and forth between x's and the θ's but that can solve for x and θ simultaneously.

(☆) Minimizing $x^{(1)}, \dots, x^{(n_m)}$ and $\theta^{(1)}, \dots, \theta^{(n_u)}$ simultaneously:

$$J(x^{(1)},\dots,x^{(n_m)},\theta^{(1)},\dots,\theta^{(n_u)}) = \frac{1}{2}\sum_{(i,j):r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^{(i)})^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^{(j)})^2$$

$$\min_{\substack{x^{(1)},\dots,x^{(n_m)}\\ \theta^{(1)},\dots,\theta^{(n_u)}}} J(x^{(1)},\dots,x^{(n_m)},\theta^{(1)},\dots,\theta^{(n_u)})$$

→ Instead of sequentially going back and forth we are just minimizing the cost function w.r.t. both sets of parameters simultaneously

→ Note we are not using ~~$x_0 = 1$~~ convention thus $x \in \mathbb{R}^n$, $\theta \in \mathbb{R}^n$ since we are learning our features algorithm can learn a feature = 1 always if it is needed.

(☆) Aslında burada olayı şöyle düşünebilirim: Line fitting yapıyorum cost için (puan tahmini - gerçek puan)^2 kullanıyorum. Puan tahmini de etkileyen x ve θ normalde sadece θ'yı minimize ederken burada x'i de minimize ediyorsun. x de bir parametre gibi düşünülüyor o kadar.

- W9 - Collaborative Filtering Alg. IV -

Collaborative Filtering Algorithm: Guideline

1) Initialize $x^{(1)}, \dots, x^{(n_m)}, \Theta^{(1)}, \dots, \Theta^{(n_m)}$ to small random values. (like NN algorithm.)

2) Minimize $J(x^{(1)}, \dots x^{(n_m)}, \Theta^{(1)}, \dots \Theta^{(n_u)})$ using Gradient Descent (or an Adv. Opt. Alg.) E.g. for every $j = 1, \dots, n_u$, $i = 1, \dots n_m$:

- $x_k^{(i)} := x_k^{(i)} - \alpha \left(\sum_{j : r(i,j)=1} \left((\Theta^{(j)})^T \cdot x^{(i)} - y^{(i,j)} \right) \Theta_k^{(j)} + \lambda x_k^{(i)} \right)$ $\quad \frac{\partial}{\partial x_k^{(i)}} J(\dots)$

- $\Theta_k^{(j)} := \Theta_k^{(j)} - \alpha \left(\sum_{i : r(i,j)=1} \left((\Theta^{(j)})^T \cdot x^{(i)} - y^{(i,j)} \right) \cdot x_k^{(i)} + \lambda \Theta_k^{(j)} \right)$ $\quad \frac{\partial}{\partial \Theta_k} J(\dots)$

3) For a user with parameter Θ and a movie with (learned) features x, predict a star rating of $\Theta^T \cdot x$

(*) $x_0 = 1$ artık yok! $x \in \mathbb{R}^n$, $\Theta \in \mathbb{R}^n$ böyle olunca artık ~~tüm~~ x ve Θ'ları regülarize ediyoruz Θ_0 için ayrı bir case yok!

W9 - Low Rank Matrix Factorization -

→ Bu kısımda algoritmanın vectorsation implementation'ından bahsedeceğiz. Bunun yanında verilen bir filme veya ürüne benzeyen diğer film/ürün'leri nasıl bulabileceğimizden bahsedeceğiz.

Alternative Way of Writing out the Predictions of the Collaborative Filtering Algorithm:

→ Film rating tablosunu alıp matrix şeklinde yazabiliriz. i: filmler, j: users.

- $Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 5 & 0 \end{bmatrix}$ ← $y^{(1,4)}$
- $n_m = 5$, $n_u = 4$

Predicted Ratings: ↳ $\theta^{(j)} \cdot x^{(i)}$

$$\begin{bmatrix} (\theta^{(1)})^T \cdot (x^{(1)}) & (\theta^{(2)})^T \cdot (x^{(1)}) & \cdots & (\theta^{(n_u)})^T \cdot (x^{(1)}) \\ (\theta^{(1)})^T \cdot (x^{(2)}) & \vdots & & \\ \vdots & & & \\ (\theta^{(1)})^T \cdot (x^{(n_m)}) & \cdots & & (\theta^{(n_u)})^T \cdot (x^{(n_m)}) \end{bmatrix}$$

→ There is a vectorized way:

each row is a movie

$$X = \begin{bmatrix} -(x^{(1)})^T- \\ -(x^{(2)})^T- \\ \vdots \\ -(x^{(n_m)})^T- \end{bmatrix} \qquad \Theta = \begin{bmatrix} -(\theta^{(1)})^T- \\ -(\theta^{(2)})^T- \\ \vdots \\ -(\theta^{(n_u)})^T- \end{bmatrix}$$

(*) Given vectorized x and Θ, the matrix of predictions will be: $X \cdot \Theta^T$: Low rank matrix factorization.

Using Learned Features to Find Related Movies:

→ For each movie/product i, we learn a feature vector $x^{(i)} \in \mathbb{R}^n$

→ Bu algoritmayla feature öğrenince bu feature'ların ne olduğunu bilmeyiz sadece filmleri karakterize eden bazı özellikler olduklarını bilebiliriz

- mesela x_1 = romance, x_2 = action, x_3 = comedy, x_4 = --- gibi bir şey olabilir

Ancak bunları böyle anlayamayız insanın anlayabileceği bir feature değil.

→ How to find movies j related to movie i?

(*) Bunu feature values ile yapabiliriz. Eğer iki film n dimensional uzayda birbirine yakınsa demekki bunlar aşağı yukarı aynı tür filmlerdir.

(*) if $\|x^{(i)} - x^{(j)}\|$ is small movie j and i are "similar"

(*) If I need to recommend 5 similar movies to i then find the 5 movies j with the smallest $\|x^{(i)} - x^{(j)}\|$.