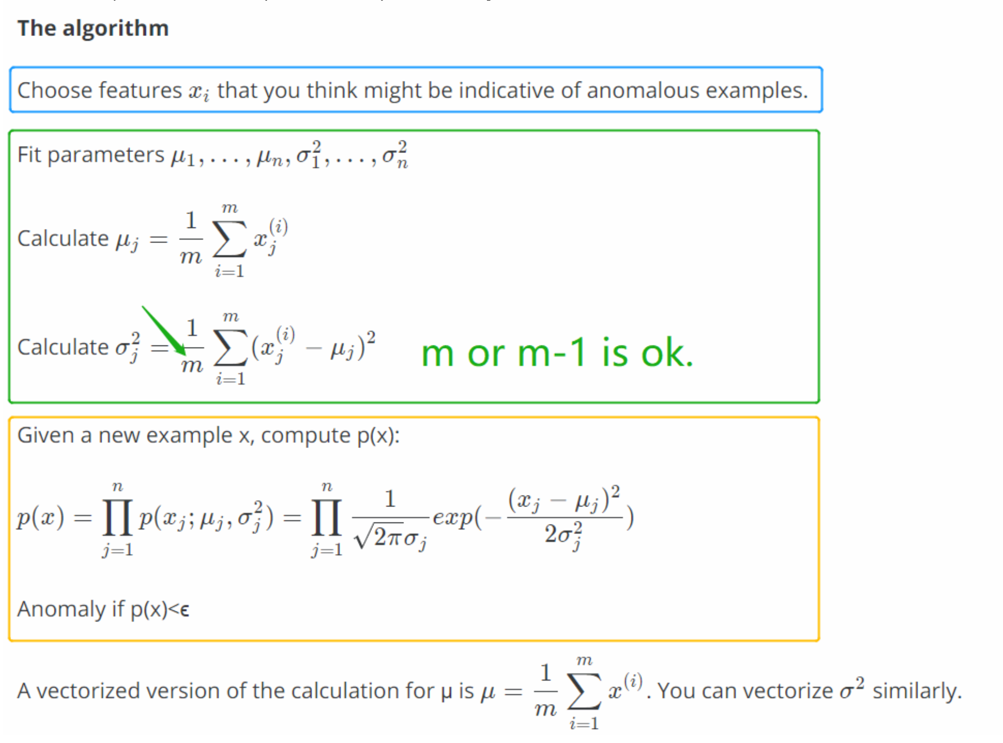
Anomaly Detection

Density Estimation

- **motivation**
  - when we are given new example, we want to examine whether it is different from previous general. (abnormal)
  - example
    - monitoring computers in data center
    - fraud detection: human or machine?
      - features like typing speed...
    - engine examine
- **Assumption**
  - object features's distribution are came from Gaussian Distribution.
  - each feature is **independent** of others. (relation between features would not be captured by model)
- when to implement?
  - we don't have fatal indicators or have too many indicators to evaluate object.
  - we want to detect abnormal/similarity between example and others.
- **Algorithm**
  - **summary:** suppose features **X** obey Gaussian Distribution, compute there μ (**expectation**) and σ (**standard deviation**). feed example to see probability.
  - 

The algorithm

Choose features x_i that you think might be indicative of anomalous examples.

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

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

Calculate $\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$ m or m-1 is ok.

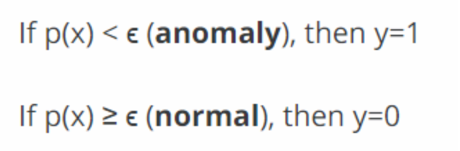
Given a new example x, compute p(x):

$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} exp(-\frac{(x_j-\mu_j)^2}{2\sigma_j^2})$$

Anomaly if p(x)<ϵ

A vectorized version of the calculation for μ is $\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$. You can vectorize σ^2 similarly.

Building a Detection System

- **real-number evaluation**
  - data set
    - have labels
    - usually skewed
  - split into training/ cross validation/ testing sets.
    - 

If p(x) < ϵ (**anomaly**), then y=1

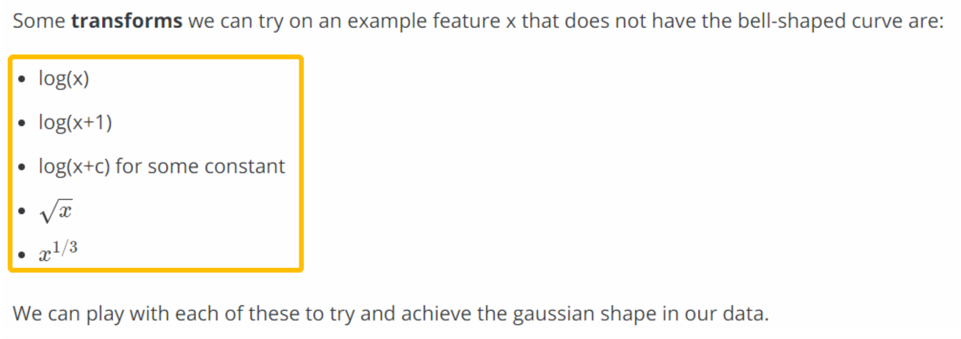
If p(x) ≥ ϵ (**normal**), then y=0

  - **Confusion matrix**👉**precision/recall, F-1 score** to evaluate

- here comes the question: if we have labeled data set, why don't we just use supervised learning algorithm? . Comparison below:

anomaly detection	situations	supervised learning
yes, with **litter positive examples** (y=1) and large number of negative examples (y=0)	skewed data?	no, set is more **evenly divided** into classes.
many features but not enough data for supervised learning, we choose anomaly detection.	feature size?	We have enough positive examples for the algorithm to get a sense of what new positives examples look like.
yes, we may get new features because future anomalies may look nothing like any of the anomalous examples we've seen so far.	flexible features?	no, we will face **similar** anomalous in future.

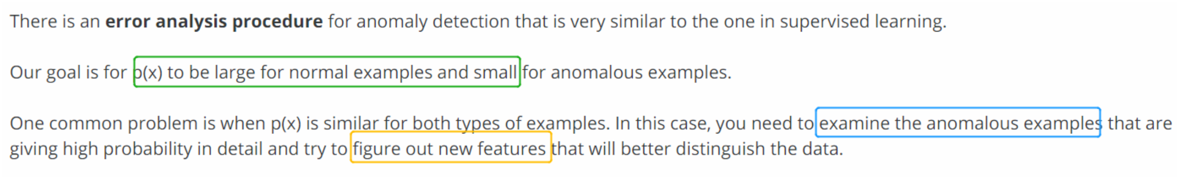
- **choosing features**
    - main idea: **fit Gaussian distribution** ☜ mapping/create new features



1. Some **transforms** we can try on an example feature x that does not have the bell-shaped curve are:

    - log(x)
    - log(x+1)
    - log(x+c) for some constant
    - $\sqrt{x}$
    - $x^{1/3}$

    We can play with each of these to try and achieve the gaussian shape in our data.



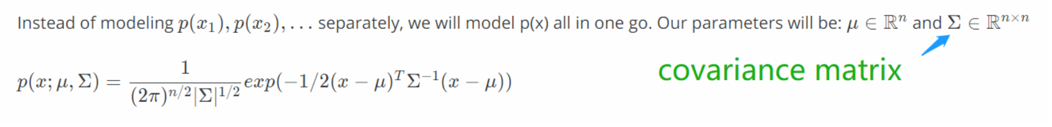
2. There is an **error analysis procedure** for anomaly detection that is very similar to the one in supervised learning.

    Our goal is for p(x) to be large for normal examples and small for anomalous examples.

    One common problem is when p(x) is similar for both types of examples. In this case, you need to examine the anomalous examples that are giving high probability in detail and try to figure out new features that will better distinguish the data.

Detection System with multivariate Gaussian Distribution

- using n-Gaussian distribution formula:

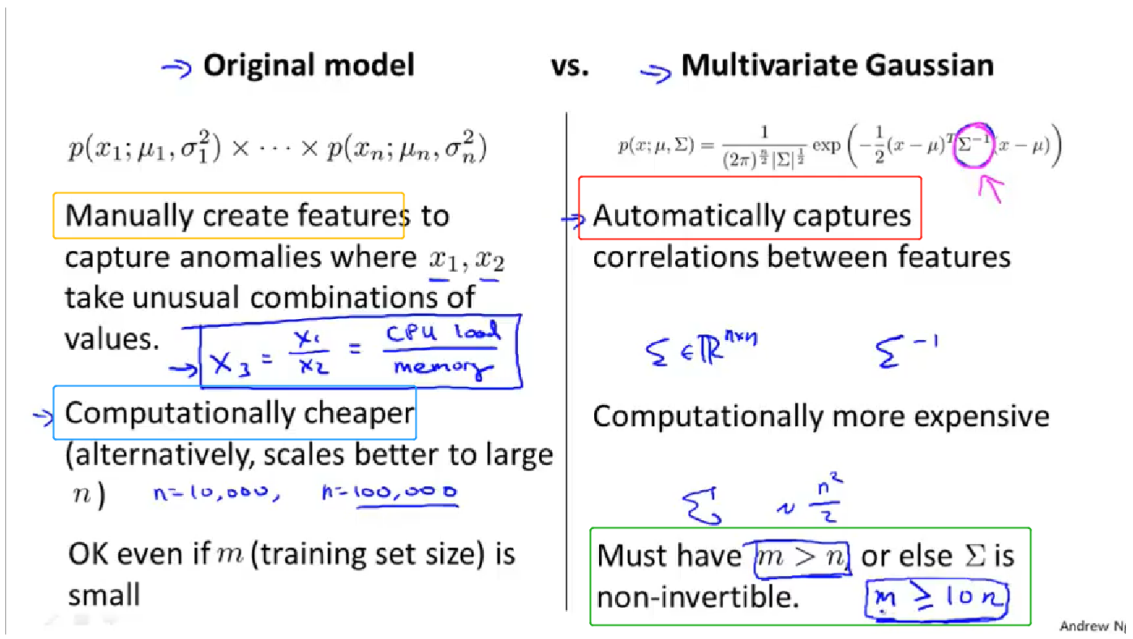


Instead of modeling $p(x_1), p(x_2), \dots$ separately, we will model p(x) all in one go. Our parameters will be: $\mu \in \mathbb{R}^n$ and $\Sigma \in \mathbb{R}^{n\times n}$

covariance matrix

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

- comparision



Original model vs. **Multivariate Gaussian**

$p(x_1;\mu_1,\sigma_1^2) \times \cdots \times p(x_n;\mu_n,\sigma_n^2)$

Manually create features to capture anomalies where x_1, x_2 take unusual combinations of values. $\boxed{x_3 = \frac{x_1}{x_2} = \frac{\text{CPU load}}{\text{memory}}}$

Computationally cheaper (alternatively, scales better to large n) $n=10{,}000$, $n=100{,}000$

OK even if m (training set size) is small

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

Automatically captures correlations between features

$\Sigma \in \mathbb{R}^{n\times n}$ Σ^{-1}

Computationally more expensive

$\Sigma \sim \frac{n^2}{2}$

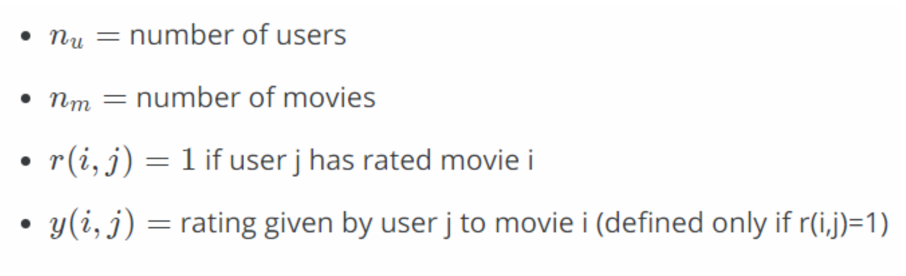
Must have $m > n$ or else Σ is non-invertible. $\boxed{m \geq 10n}$

Andrew Ng

- so when decide to use multivariate Gaussian model, remember to clear the redundant features first.

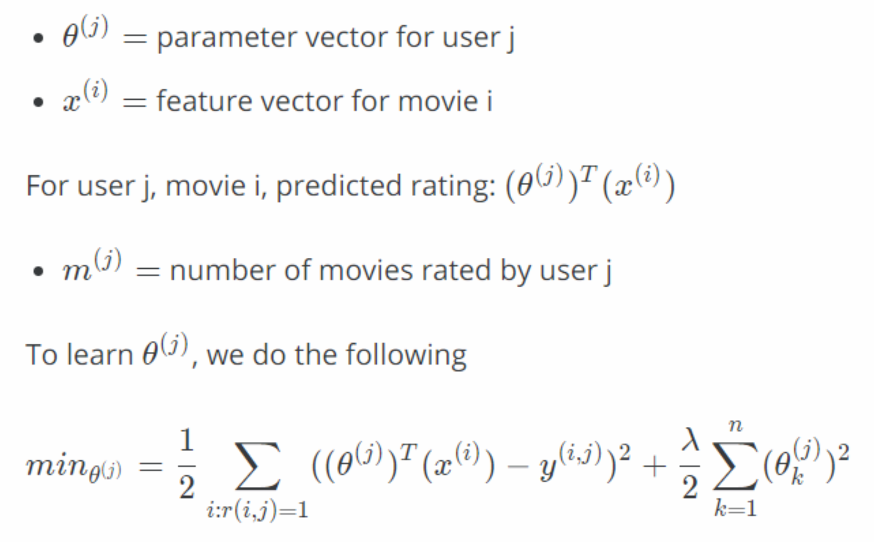
Recommender System

- denotation:



- n_u = number of users
- n_m = number of movies
- $r(i,j) = 1$ if user j has rated movie i
- $y(i,j)$ = rating given by user j to movie i (defined only if r(i,j)=1)

- **assumption:**
  - people who agreed in the past will agree in the future, and that they will like similar kinds of items as they liked in the past.
- seek for user preference



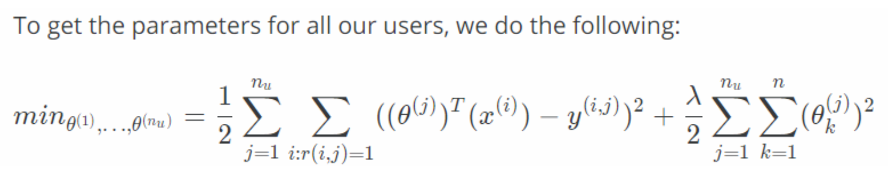
- $\theta^{(j)}$ = parameter vector for user j
- $x^{(i)}$ = feature vector for movie i

For user j, movie i, predicted rating: $(\theta^{(j)})^T(x^{(i)})$

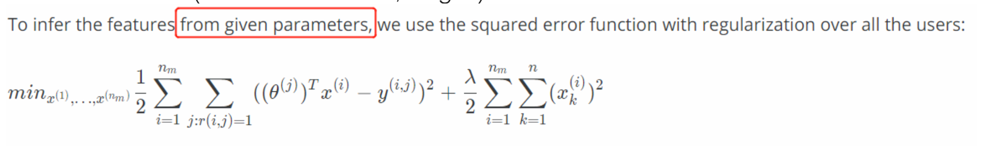
- $m^{(j)}$ = number of movies rated by user j

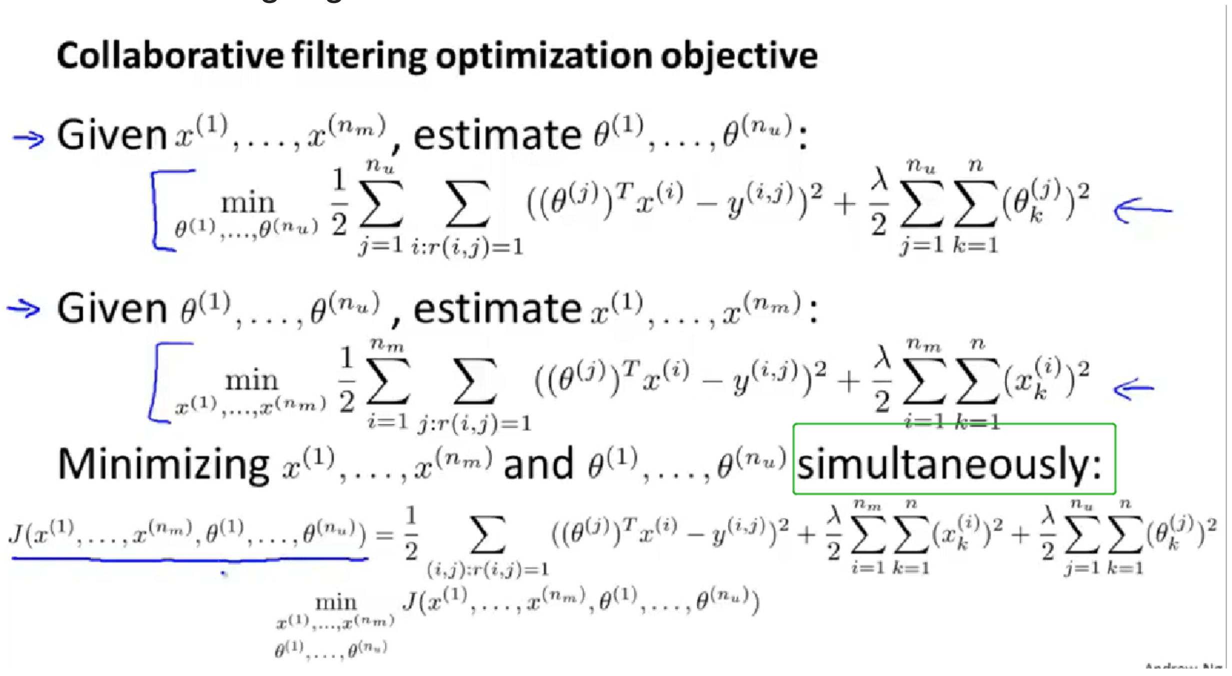
To learn $\theta^{(j)}$, we do the following

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


To get the parameters for all our users, we do the following:

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

  - for all users:
- seek for movie features (such as romantic, tragic)
  - 
    To infer the features from given parameters, we use the squared error function with regularization over all the users:

    $$min_{x^{(1)},\dots,x^{(n_m)}} \frac{1}{2}\sum_{i=1}^{n_m}\sum_{j:r(i,j)=1}((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^{(i)})^2$$
- Collaborative Filtering Algorithm
  - 
    Collaborative filtering optimization objective

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

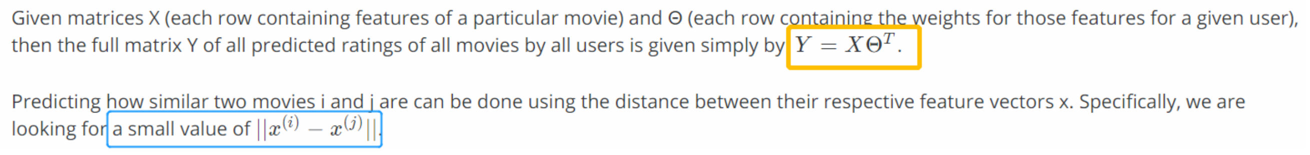
    $$\Big[\min_{\theta^{(1)},\dots,\theta^{(n_u)}} \frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r(i,j)=1}((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^{(j)})^2 \leftarrow$$

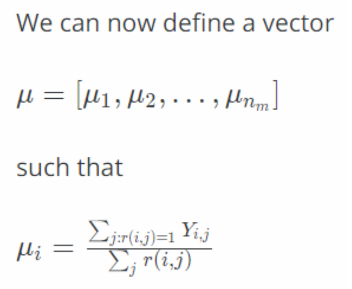
    $\rightarrow$ Given $\theta^{(1)},\dots,\theta^{(n_u)}$, estimate $x^{(1)},\dots,x^{(n_m)}$:

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

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

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

    $$\min_{\substack{x^{(1)},\dots,x^{(n_m)}\\ \theta^{(1)},\dots,\theta^{(n_u)}}} J(x^{(1)},\dots,x^{(n_m)},\theta^{(1)},\dots,\theta^{(n_u)})$$
- **Low Rank Matrix Factorization**
  - 
    Given matrices X (each row containing features of a particular movie) and Θ (each row containing the weights for those features for a given user), then the full matrix Y of all predicted ratings of all movies by all users is given simply by $Y = X\Theta^T$.

    Predicting how similar two movies i and j are can be done using the distance between their respective feature vectors x. Specifically, we are looking for a small value of $||x^{(i)} - x^{(j)}||$.
- **Mean Normalization**
  - assume that now comes a new user, how can we initialize its θ
  - 
    We can now define a vector

    $$\mu = [\mu_1, \mu_2, \dots, \mu_{n_m}]$$

    such that

    $$\mu_i = \frac{\sum_{j:r(i,j)=1} Y_{i,j}}{\sum_j r(i,j)}$$
  - normalize the data by subtracting u

---

development in collective filtering

1. user-based
2. item-based
3. model-based