WEEK - 9

⇒ Bad use of PCA → To prevent overfitting

⇒ ANOMALY DETECTION (Unsupervised)

→ Fraud Detection

→ $x^{(i)}$ = features of user i's activities
→ Model $p(x)$ from data
→ Identify unusual users by checking which have $p(x) < \varepsilon$

→ Manufacturing

→ Density estimation → we make a cluster and if the test example is in the cluster, its OK. otherwise its an anomaly.

→ Dataset : $\{x^{(1)}, x^{(2)} \ldots x^{(m)}\}$
New engine : x_{test}

→ Monitoring computers in a data center

→ $x^{(i)}$ = features of machine i
→ x_1 = memory use, x_2 = number of disk accesses / sec,
→ x_3 = CPU load
→ x_4 = CPU load / network traffic
divided by.

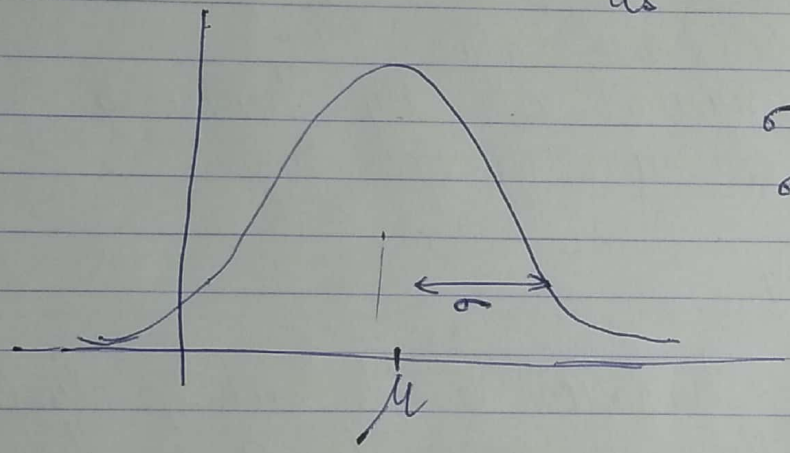
⇒ Gaussian (Normal) Distribution

Say, $x \in \mathbb{R}$. If x is a distributed gaussian with means μ, variance(σ^2)

$$x \sim \mathcal{N}(\mu, \sigma^2)$$

"distributed as"

stands for Normal.



σ = standard ~~variance~~ deviation

σ^2 = variance

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

```
mu = ((1/m) * sum(x))';
```

~~sigma2 = (1/m) * (sum((x-mu)'.^2))';~~

$$\Rightarrow \mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}, \quad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)} - \mu\right)^2$$

```
sigma2 = (1/m) * (sum((x-mu').^2))';
```

$$\Rightarrow \sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n \quad \text{(Add)}$$

$$\prod_{j=1}^{n} j = 1 \times 2 \times 3 \times \ldots \times n \quad \text{(multiply)}$$

$x_1 \sim N(\mu_1, \sigma_1^2)$

⇒ Density estimation

→ Training set: $\{x^{(1)}, \ldots, x^{(m)}\}$

→ Each example is $x \in \mathbb{R}^n$.

⇒ $P(x) =$ Probability of features

$= P(x_1, \mu_1, \sigma_1^2) * P(x_2, \mu_2, \sigma_2^2) \ldots P(x_n, \mu_n, \sigma_n^2)$

$$\boxed{= \prod_{j=1}^{n} P(x_j; \mu_j, \sigma_j^2)}$$

⇒ Anomaly Detection Algorithm

1) = Choose features x_i that you think might be indicative anomalous examples.

2) = For examples $\{x^{(1)}, x^{(2)} \ldots x^{(m)}\}$, Fit parameters $\mu_1, \ldots \mu_n, \sigma_1^2 \ldots \sigma_n^2$,

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

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

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

$$p(x) = \prod_{j=1}^{n} p(x_j ; \mu_j ; \sigma_j^2)$$

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

Anomaly if $p(x) < \varepsilon$

⇒ ~~Now~~ We can use cross validation set to choose parameter ε (epsilon)

⇒ When to use anomaly detection and when to use Supervised Learning.

Anomaly Detection	Supervised Learning
1) = Very small no. of positive examples ($y=1$). → for (0-50 no) & large no. of -ve examples ($y=0$)	1) = Large no. of positive and negative examples.
2) = Many different types of anomalies	2) = Enough +ve examples for algorithm to get a sense of what +ve examples are like.
3) = Future anomalies may look nothing like any of the anomalous examples we have seen.	3) = Future +ve examples likely to be similar to ones in training set.

⇒ Anomaly detection usually works good on skewed datasets.

⇒ It is a good habit to make the data gaussian before feeding it to the algo.

⇒ Multivariate Gaussian Distribution

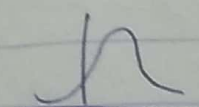
→ $x \in \mathbb{R}^n$. Dont model $p(x_1), p(x_2) \ldots$ etc separately.

→ Model $p(x)$ all in one go

Parameters :→ $\mu \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$ (covariance matrix)

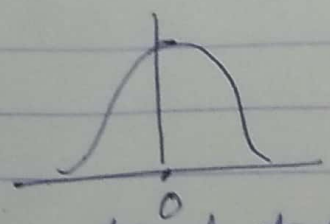
$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

$|\Sigma| \rightarrow$ determinant of Σ / det(Sigma)

→ The area under the bell curve  is always 1 (probability amounts to 1)

⇒ $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

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



centered at (0,0) ⇒ μ.

⇒ for varying the value of Σ, the curve flattens or changes its shape

→ for $\Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix}$ → curves become steeper & taller.

→ for $\Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$ → curves becomes flat & small.

→ for $\Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$ → curve steeps from N-W & S-E as value increases after 0

→ for $\Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$ → curve steeps from NE & SW as value decreases under 0

→ for $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 0.6 \end{bmatrix}$ → curve flattens from x_2 side

⇒ <u>Parameter fitting</u> →

Given training set $\{x^{(1)}, x^{(2)}, \ldots\ldots x^{(m)}\}$

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

⇒ Anomaly detection with Multivariate Gaussian

1) = Fit model $p(x)$ by setting

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

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

2) = Given a new example x, compute

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

Flag an anomaly if $p(x) < \varepsilon$

⇒ The original gaussian model is same as the multivariate gaussian ~~at~~ with a ~~constraint~~ constraint.

where $\Sigma = \begin{bmatrix} \sigma_1^2 & & & 0 \\ & \sigma_2^2 & & \\ & & \sigma_3^2 & \\ 0 & & & \ddots \; \sigma_n^2 \end{bmatrix}$ must have 0 everywhere except the diagonal

⇒ The contours of multivariate gaussian are axis aligned in the case of old model ⟶ The old model is a special case of multivariate gaussian.

Original Model	Multivariate Gaussian
$p(x_1; \mu_1; \sigma_1^2) \times \ldots \times p(x_n; \mu_n; \sigma_n^2)$	$p(x; \mu; \Sigma) = \frac{1}{(2\pi)^{n/2} \lvert\Sigma\rvert^{1/2}} \exp(\ldots)$
→ Manually create features to capture anomalies where x_1, x_2 take unusual combinations of values	→ Automatically captures corelations between features.
→ Computationally cheaper (scales better to large n)	→ Computationally more expensive
→ OK even if m training set size is small	→ Must have $m \geq n$, or else Σ is non-invertible. $(m \geq 10n)$

⇒ If Σ is non-invertible then —

1) = $m \leq n$

2) = redundant features (similar features)

RECOMMENDER SYSTEMS

→ n_u = no. of users.
→ n_m = no. of movies
→ m_j ⇒ no. of movies rated by user.
→ $r(i,j) = 1$, if user j has rated movie i

→ $y(i,j)$ = rating given by user j to movie i.
(depends only if $r(i,j) = 1$)

⇒ To learn $\underline{\theta^{(j)}}$ (for user j)

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

read as → summation of all values of i, (all movies rated by j) so the $r(i,j)$ is equal to 1.

$$+ \frac{\lambda}{2m^{(j)}} \sum_{k=1}^{n} \left(\theta_k^{(j)} \right)^2$$

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

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

$$J(\theta^{(1)}, \theta^{(2)} \ldots \theta^{(n_u)})$$

$$\frac{\partial}{\partial \theta}\left(J(\theta^1, \theta^2 \ldots \theta^{(n_u)})\right)$$

⇒ <u>Gradient descent update</u>

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

for $\underline{\underline{k=0}}$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) x_k^{(i)} + \lambda \theta_k^{(j)} \right)$$

(for $k \neq 0$)

⇒ Given $\theta^{(1)}, \theta^{(2)} \ldots \theta^{(n)}$, to learn $\underline{\underline{x^{(i)}}}$:

$$\rightarrow \min_{x^{(i)}} \frac{1}{2} \underbrace{\sum_{j:r(i,j)=1}}_{\text{sum of all users who rated the movie } i} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{k=1}^{n} \left(x_k^{(i)} \right)^2$$

⇒ Given $\theta^{(1)} \ldots \theta^{(n_u)}$, to learn $\underline{\underline{x^{(1)}, \ldots x^{(n_m)}}}$:

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

⇒ Collaborative Filtering Algorithm

→ Given $x^{(1)}, \ldots, x^{(n_m)}$ (and movie ratings), can estimate $\theta^{(1)}, \ldots, \theta^{(n_u)}$

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

→ So what we solve first →
→ we can guess random values of θ and the solve for X, then taking those values, we can get a better θ, and use this θ for new X and so on.

And we converge to reasonable features

⇒ Minimizing $x^{(1)}, \ldots x^{(n_m)}$ & $\theta^{(1)} \ldots \theta^{(n_u)}$ simultaneously

$$J\left(x^{(1)} \ldots x^{(n_m)}, \theta^{(1)} \ldots \theta^{(n_u)}\right) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} \left(\left(\theta^{(j)}\right)^T x^{(i)} - y^{(i,j)}\right)^2$$

$$\min_{\substack{x^{(1)} \ldots x^{(n_m)} \\ \theta^{(1)} \ldots x^{(n_u)}}}$$

$$+ \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} \left(x_k^{(i)}\right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(\theta_k^{(j)}\right)^2$$

⇒ Basically adding both the equations.

⇒ X_grad = (R.*(X*Theta' - Y))*Theta;
Y_grad = (R.*(X*Theta' - Y))'*X;

⇒ By using the above formula, we do not need to solve for θ, θ' and then for X, again & again.

We can minimize them simultaneously w.r.t each parameter.

⇒ **Algo**

1) = Initialize $x^{(1)} \dots x^{(n_m)}, \theta^{(1)} \dots \theta^{(n_u)}$ to small random values (this ensures symmetry breaking and ensures the algo learns features $x^{(1)} \dots x^{(n_m)}$ that are different from each other)

2) = Minimize $J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)} \dots \theta^{(n_m)})$ using gradient descent (or an advanced optimization algo). Eg. for every $j = 1, \dots n_u$, $i = 1 \dots n_m$

$$\frac{\partial}{\partial x_k^{(i)}} J() \rightarrow x_k^{(i)} := x_k^{(i)} - \alpha \left(\sum_{j:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) \theta_k^{(j)} + \lambda x_k^{(i)} \right)$$

$$\frac{\partial}{\partial x_k^{(j)}} J() \rightarrow \theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{j:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) x_k^{(i)} + \lambda \theta_k^{(j)} \right)$$

3) = For a user with parameters θ and a movie with (learned) features x, predict a star rating of $\theta^T x$.

⇒ Collaborative Filtering

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix} \quad \overset{\text{Predicted ratings}}{\begin{bmatrix} (\theta^{(1)})^T x^{(1)} & (\theta^{2})^T x^{1} & \cdots & (\theta^{(n_u)})^T x^{(1)} \\ (\theta^{1})^T x^{(2)} & & & (\theta^{(n_u)})^T x^{(2)} \\ \vdots & & & \vdots \\ (\theta^{1})^T x^{(n_m)} & (\theta^{2})^T x^{(n_m)} & & (\theta^{(n_u)})^T (x^{(n_m)}) \end{bmatrix}}$$

⇒ Let $x = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \\ — (x^{(n_m)})^T — \end{bmatrix}$, $\Theta = \begin{bmatrix} — (\theta^{(1)})^T — \\ — (\theta^{(2)})^T — \\ \\ — (\theta^{(n_u)})^T — \end{bmatrix}$

$$\Rightarrow \quad X\Theta^T = \begin{bmatrix} (x^{(1)})^T \theta^{(1)} & \cdots & (x^{(1)})^T (\theta^{(n_u)}) \\ & & \\ (x^{(n_m)})^T \theta^{(1)} & & (x^{(n_m)})^T (\theta^{(n_u)}) \end{bmatrix}$$

Aka → Low Rank Matrix Factorization.

⇒ To find similar movies, find the movies with smallest $\|x^{(i)} - x^{(j)}\|$, $x^{(i)}$ being your initial movie.

⇒ If a new user is added and it has rated no movies; So instead of giving 0 reviews to every movie, we can mean normalize each row and gives that value as rating.