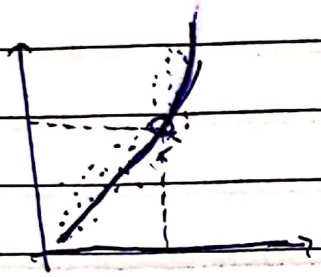
are vain make em:valid. While I'm all well awhat a wise words with just weaving up words on my workshirt. Xerox, my x-radiation holes, x-height letters the xylophones tones. Yellow back young are yesterdays lawn yard sell our yarn. Zigzag zombies, zoom into the zenith, zero in ZEBRAS!

Linear Regression

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \dots + \theta_n x_n \quad | \quad x_0 = 1 \;\&\; x_j = \text{feature } j \text{ of column } i$$



$$cost(h_\theta(x)) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \quad | \quad x^{(i)} \;\&\; y^{(i)} \rightarrow i^{th} \text{ row/training example}$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\times\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\cdot x_j^{(i)} \quad | \quad x_j = \text{Feature } j$$

VECTORIZED IMPLEMENTATION

Given matrixes

m → no. of examples

n → no. of features

- $[X]^{m \times (n+1)}$
- $[y]^{m \times 1}$
- $[theta]^{(n+1) \times 1}$

```
Hypothesis = X * theta; % Gives a m x 1 vector.
cost(h_theta(X)) = sum((hypothesis - y).^2))/(2*m); % 1x1 var.
Gradient = (1/m) * (X' * (hypothesis - y)); % Returns a (n+1)x1 vector
```

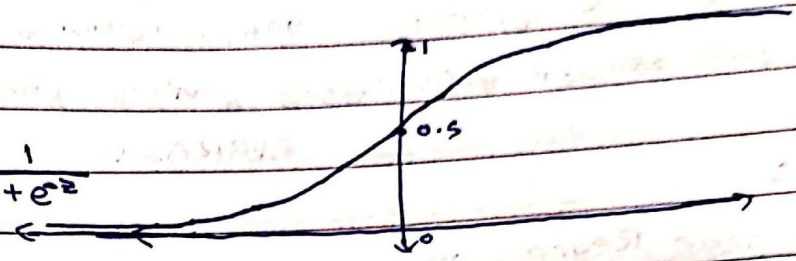
GRADIENT DESCENT:

```
theta = theta - (alpha) * Gradient;
```

$$\theta_j := \theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\,x_j^{(i)}$$

LOGISTIC REGRESSION:

Sigmoid function: $g(z) = \frac{1}{1+e^{-z}}$

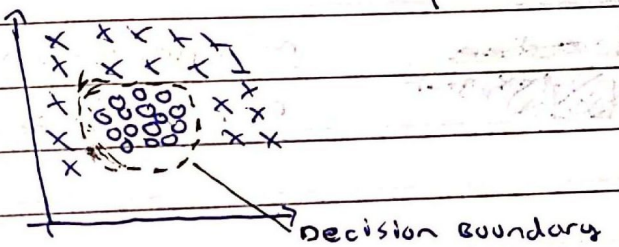


Hypothesis:

$h_\theta(x) = g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n)$ or $g(\theta^T x)$

Decision Boundary:

$x = 0$	$x = 1$
$g(x) < 0.5$ or $g(x) \leq 0.5$	$g(x) \geq 0.5$ or $g(x) > 0.5$
When, $\theta^T x < 0$ or $\theta^T x \leq 0$	When, $\theta^T x \geq 0$ or $\theta^T x > 0$



Cost function: ~~$J(\theta) = \text{cost}(h_\theta x^{(i)})$~~

$$\text{Cost}(h_\theta(x)) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \log(h_\theta(x^{(i)})) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right]$$

$$\frac{dJ}{d\theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y)\,x_j^{(i)}$$

$[x]^{m \times (n+2)}$, $[y]^{m \times 1}$, $[theta]^{(n+1) \times 1}$

m → no. of [illegible]

n → no. of features

VECTORIZED IMPLEMENTATION

```
z = X * theta; % Returns a m x 1 vector
sigmoid = 1./(1.+exp(-z)); % Returns our hypothesis. (m x 1) vector. 0 ≤ sigmoid ≤ 1
hypothesis = sigmoid;
cost = (-1/m)*((y' * log(hypothesis)) + (1-y)' * (1-log(hypothesis)))
gradient = (1/m)*(X' * hypothesis);
```

REGULARIZATION

$$Cost'(h(x)) = cost(h(x)) + \left(\frac{\lambda}{2m}\right)\sum_{j=1}^{n}\theta_j^2$$

*note: Not minimizing / regularizing θ_0 since it has no effect on the fn. IMO would negatively affect it

LINEAR REGRESSION:

$$\text{GRADIENT} = \frac{1}{m}\left[\sum_{i=1}^{m}\left((h(x^{(i)}) - y^{(i)})^2 x_j^{(i)}\right) + \lambda\theta_j\right] \quad \text{For } j \neq 0$$

IMPLEMENTATION: $[X]^{m \times (n+2)}$, $[y]^{m \times 1}$, $[\theta]^{(n+1) \times 1}$

```
gradient(1) = (1/m)*(X(:,1)' * (hypothesis - y));
gradient(2:end) = (1/m)*((X(:,2:end)' * (hypothesis - y))
                  + lambda * theta(2:end));
```

LOGISTIC REGRESSION:

$$\text{GRADIENT} = \frac{1}{m}\left[\sum_{i=1}^{m}\left((h(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}\right) + \lambda\theta_j\right]$$

IMPLEMENTATION: $[X]^{m \times (n+1)}$, $[y]^{m \times 1}$, $[\theta]^{(n+1) \times 1}$

```
gradient(1) = (1/m)*(X(:,1)' * (hypothesis - y));
gradient(2:end) = (1/m)*((X(:,2:end)' * (hypothesis - y)) + lambda*theta(2:end));
```

LINEAR REGRESSION VIA NORMAL EQUATIONS:

$$[\theta] = (X^T X)^{-1} X^T y$$

pseudo inverse.

→ theta = pinv(X'*X)*X'*y;

$X^T X$ could be non-invertible if we have
- Redundant features, example: when 2 features are ~~very closely~~ linearly dependant. (Ex: $x_1 = 326.4\, x_2$ always)
- Too many features (Ex: $m \leq n$).

To Fix: Delete some features or use regularization.

REGULARIZATION:

$$[\theta] = (X^T X + \lambda \cdot L)^{-1} X^T y.$$

L is an identity matrix with element (1,1) = 0. | $[L]^{(n+1)\times(n+1)}$

$$L = \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}$$

The regularized $[\theta]$ is always invertible.

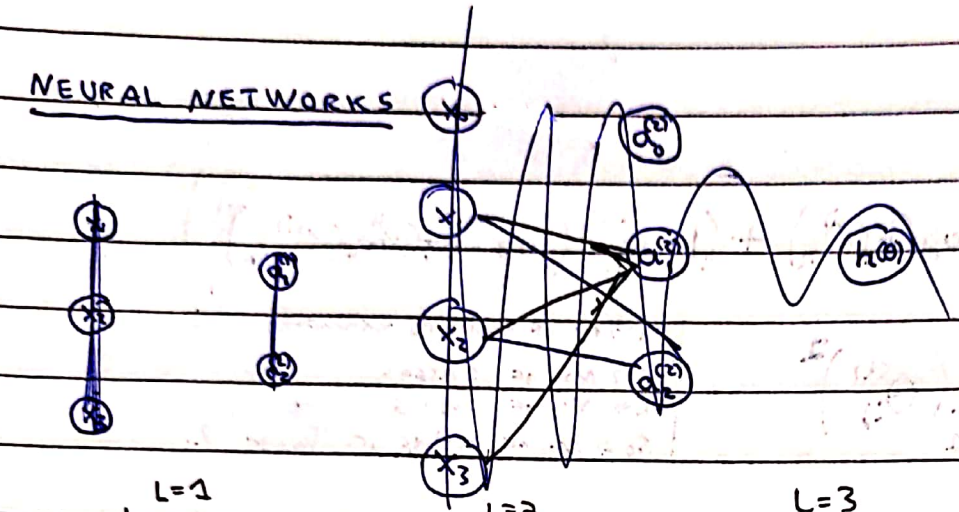
L = ones(n+1, n+1); L(1,1) = 0;

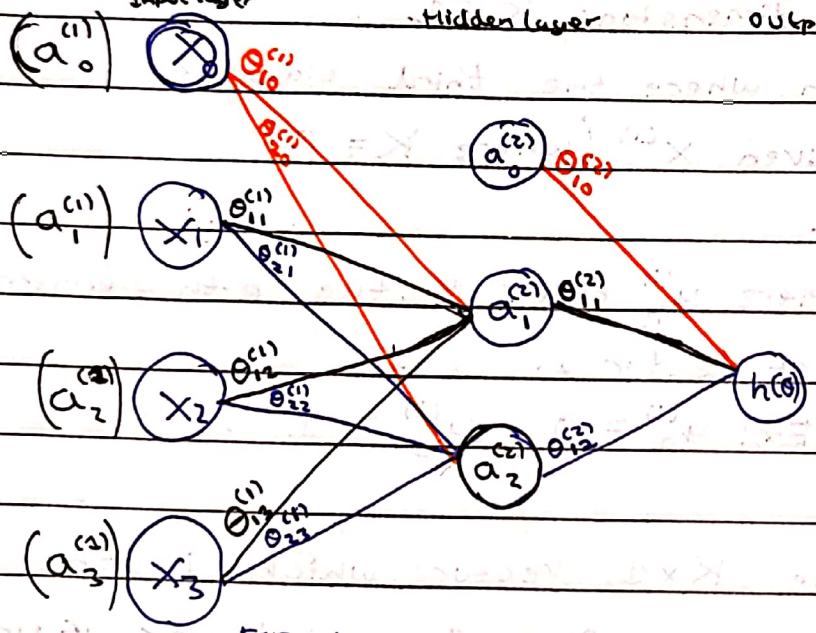
→ theta = pinv(X'*X + lambda*L)*X'*y;

GRADIENT DESCENT VS NORMAL EQUATIONS

GRADIENT DESCENT	NORMAL EQUATIONS
- Need to choose alpha	+ No need to choose alpha
- Needs many iterations	+ No need to iterate
+ $O(kn^2)$	- $O(n^3)$, need to calculate inv(X'*X)
+ Works well when n is large	- Slow if n is very large.
+ $n > 10,000$	+ $n \leq 10,000$

NEURAL NETWORKS





Simple neural network with 3 input nodes, one single hidden layer with 2 neurons and one output node.

x_0 & $a_0^{(2)}$ are bias'.

We can treat $[x]^{3\times 1}$ as $[a^{(1)}_{(1:3)}]^{3\times 1}$

i.e., a layer of neurons who's values are not learned as a function but instead given as input

FORWARD PROPAGATION

$a \rightarrow$ Is the activation function..

Ex: Sigmoid, ReLU, tanh, etc.

Let l denote the layer number, i denote the node number & j denote individual components.

Then $a_i^{(l)} = a\left(\sum_{j=0}^{n+1} \Theta_{ij}^{(l-1)} a_j^{(l-1)}\right)$

$n \rightarrow$ no. of nodes in the previous layer.

$a(x) \rightarrow$ activation function.

VECTORIZED IMPLEMENTATION:

If $s_n \rightarrow$ no. of nodes in layer n,

$$[a^{(l)}] = \begin{bmatrix} a_0^{(l)} \\ \vdots \\ a_n^{(l)} \end{bmatrix} = \begin{bmatrix} a^{(l)}(z_0) \\ \vdots \\ a(z_n) \end{bmatrix} = a([z])$$

$$[z]^{s_l \times 1} = [\Theta]^{s_l \times s_{l-1}} \times [x^{(l-1)}]^{s_{l-1} \times 1}$$

$(l-1)$

COST FUNCTION

$$J(\theta) = \frac{1}{m}\left(\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log\left(h_\theta(x^{(i)})_k\right) + \left(1-y_k^{(i)}\right)\left(\log\left(1-h_\theta(x^{(i)})_k\right)\right)\right]\right)$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_{l+1}}\sum_{j=1}^{S_l}\left(\theta_{ij}^{(l)}\right)^2$$

- $m \rightarrow$ No. of training examples
- $K \rightarrow$ No. of classes
- $L \rightarrow$ No. of layers
- $S_l \rightarrow$ No. of nodes in layer l.

$y^{(i)}$ refers to a vector of dimensions $K \times 1$.

For a classification problem where the third class is the correct answer for a given $x^{(i)}$ (where $K = 5$)

$y^{(i)}$ would look like this.

$$y^{(i)} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Here $y_k^{(i)}$ refers to the k-th element of vector y.

Ex: $y_2^{(i)} = 0$, $y_3^{(i)} = 1$

Similarly,

$h_\theta(x^{(i)})$ refers to the $K \times 1$ vector which is the collection of outputs of the final (output) layer of our neural network.

Ex: $h_\theta(x^{(i)}) = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.96 \\ 0.21 \\ 8.31 \end{bmatrix}$, Here $h_\theta(x^{(i)})_k$ refers to the k-th element of vector $h_\theta(x^{(i)})$.

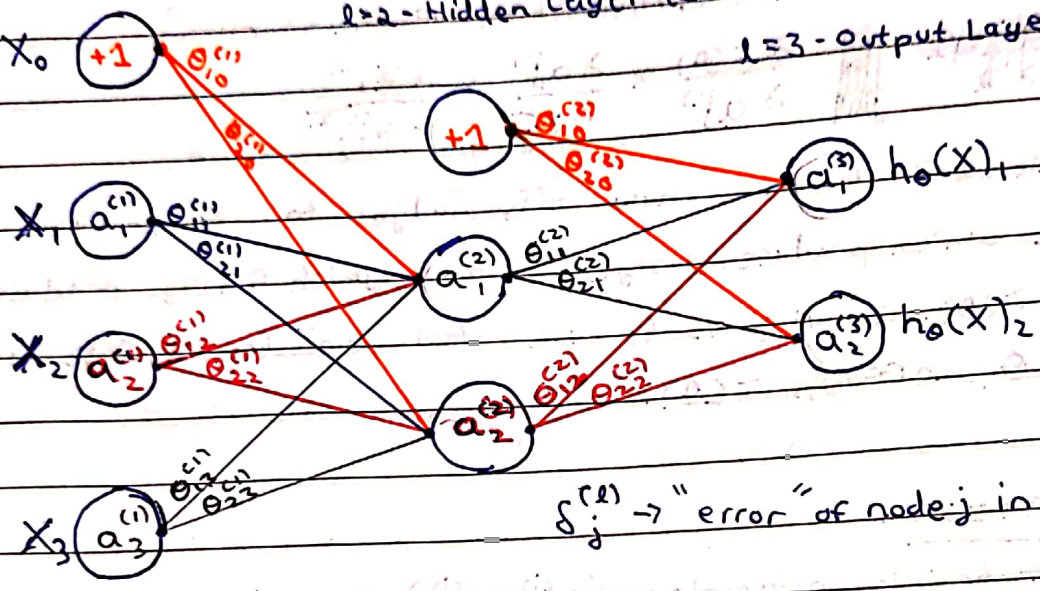
Ex: $h_\theta(x^{(i)})_2 = 0.2$, $h_\theta(x^{(i)})_3 = 0.96$.

Hence, the first part of our cost function calculates the sum of the error in the predictions by our neural network, i.e, error in each $h_\theta(x^{(i)})_k$ for its corresponding $y_k^{(i)}$

$l=1$ - Input Layer (3 units)

$l=2$ - Hidden Layer (2 units)

$l=3$ - Output Layer (2 units)



$\delta^{(l)}_j \rightarrow$ "error" of node j in layer l.

For $l = L$ (output layer) – [Vectorized implementation]

$$\delta^{(L)} = a^{(L)} - y$$

For $l \neq L$ & $l > 1$

$$\delta^{(l)} = (\theta^{(l)})^T \delta^{(l+1)} .* g'(z^{(l)})$$

$$\theta^{(2)} = \begin{array}{c|ccc} & x_0 & x_1 & x_2 \\ a^{(3)}_1 & \theta_{10} & \theta_{11} & \theta_{12} \\ a^{(3)}_2 & \theta_{20} & \theta_{21} & \theta_{22} \end{array}$$

$$[\theta^{(2)}]^T = \begin{array}{c|cc} & a^{(3)}_1 & a^{(3)}_2 \\ x_0 & \theta_{10} & \theta_{20} \\ x_1 & \theta_{11} & \theta_{21} \\ x_2 & \theta_{12} & \theta_{22} \end{array}$$

$$\delta^{(3)} = \begin{bmatrix} \delta^{(3)}_1 \\ \delta^{(3)}_2 \end{bmatrix} \rightarrow \begin{array}{l} \delta^{(3)}_1 = h_\theta(x^{(i)})_1 - y^{(i)}_1 \\ \delta^{(3)}_2 = h_\theta(x^{(i)})_2 - y^{(i)}_2 \end{array}$$

Intuition via example. On the right we have matrices $\theta^{(2)}$, $(\theta^{(2)})^T$ & $\delta^{(3)}$ displayed with their respective values as per the above neural network.

Let us now understand the term $(\theta^{(l)})^T \times \delta^{(l+2)}$ | $[\theta^{(2)}]^T \times \delta^{3} = ?$

$$\delta^{(2)} = \begin{bmatrix} \theta_{10}\delta^{(3)}_1 + \theta_{20}\delta^{(3)}_2 \\ \theta_{11}\delta^{(3)}_1 + \theta_{21}\delta^{(3)}_2 \\ \theta_{12}\delta^{(3)}_1 + \theta_{22}\delta^{(3)}_2 \end{bmatrix}$$

$\delta^{(2)}$ simply calculates the weighted sum of error due to θ in $\delta^{(3)}$ for every node in $a^{(2)}$

The $g'(z)$ term comes from the calculus derivative of the cost function. [Try to derive? Application of chain rule]

Ex: $\frac{\partial J(\theta)}{\partial \theta_{ij}^{(l-1)}} = \frac{\partial J(\theta)}{\partial a_j^{(l)}} \times \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \times \frac{\partial z_j^{(l)}}{\partial \theta_{ij}^{(l-1)}}$

useful to think about them as how tiny nudges to θ produce changes in z which produce changes in a, which finally produce changes in $J(\theta)$. Allowing us to use chain rule to calculate how a nudge to θ, affects $J(\theta)$.

BACK PROPOGATION:

~~1. Iterate over training examples $(x^{(i)}, y^{(i)})$ from $i=1$ to m.~~

1. Set $\Delta_{ij}^{(l)} = 0$ for all i, j & l.
2. Iterate over training examples $(x^{(i)}, y^{(i)})$ from $i=1$ to m.

for $i = 1:m$ [Vectorized Implementation]

(i) Set $a^{(1)} = x^{(i)}$ & add bias term.

(ii) Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3 \rightarrow L$.

(iii) Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

(iv) Compute $\delta^{(L-1)}, \delta^{(L-2)}, \rightarrow \delta^{(2)}$ using

$$\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} .* g'(z^{(l)})$$

(v) Sum all these $\delta_{ij}^{(l)}$ values per example iteration to $\Delta^{(l)}$

$$\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$$

end for

3. Calculate gradient terms from Δ.

If $j = 0$, $D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)}$

If $j \neq 0$, $D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \theta_{ij}^{(l)}$

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(l)}} = D_{ij}^{(l)}$$

EVALUATING A TRAINED ALGORITHM

- A common practice is to ~~evaluate the trained~~ split the dataset into a training dataset & testing dataset.

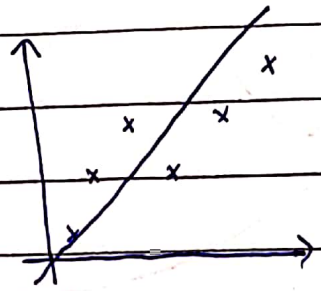
General practices: Training dataset - 70% Testing dataset - 30%.
(or)
Training dataset - 80% Testing dataset - 30%.

~~The algorithm and even the person~~ Neither the algorithm nor the creator of the algorithm should ever observe / infer any information from the test set. It should be a closely gaurded secret. In no way should it influence the algorithm.

BIAS VS VARIANCE

High Bias → underfitting

High Variance → Overfitting



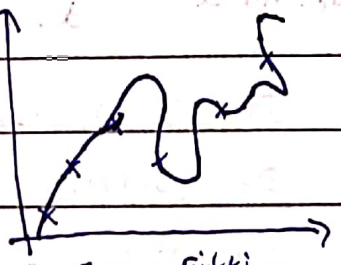
Under Fitting.

Choosing the right order of the polynomial terms in the hypothesis can help correct under & over fitting.

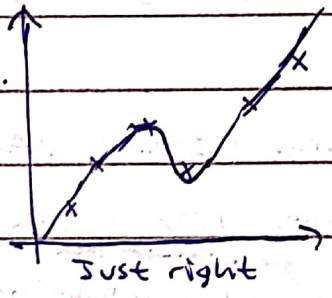
If d is the order of the polynomial,

Generally: If d is very less, → under Fitting

d is very large → over Fitting.



over fitting



Just right

① CHOOSING THE RIGHT VALUE FOR D:

We can split our dataset as follows,

60% → Training data

20% → Cross validation data

20% → Testing data

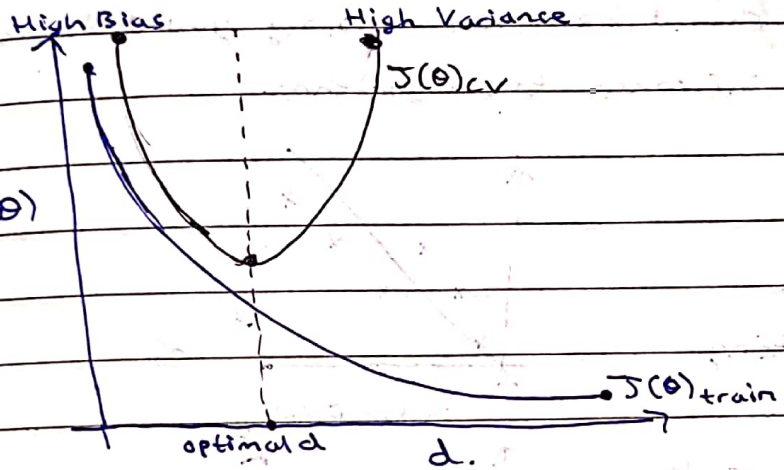
We can iterate over a range of values for d. Ex: [1 → 10]

$J(\theta)_{\text{cross validation}}$ is our cost function of choice.

Squared error cost function or logistic error cost function etc.

From the cross validation dataset we can select an optimal value for d.

The best value for d is the $J(\theta)_{\text{cross validation}}$ which gives the least value. for $(x^{(1)}, y^{(1)})$ from our cross-validation (CV) dataset.



If $J(\theta)_{training}$ is large → High bias (underfitting)
&& $J(\theta)_{cv}$ is large.

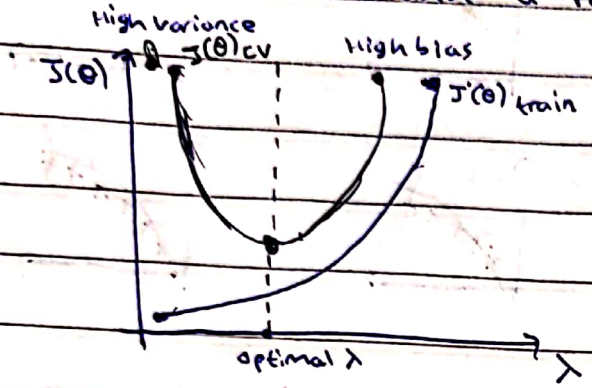
If $J(\theta)_{training}$ is very small → High variance (overfitting)
&& $J(\theta)_{cv}$ is large

② CHOOSING THE RIGHT VALUE OF λ

We can iterate over a range of values for λ. [Ex: 0.1 →(×2) 0.2, (×2) 0.4, (×2) 0.8 → 10]

$J(\theta)_{train}$ is the cost function of our choice over our training examples. and the regularization term. $J(\theta)_{cv}$ is just the cost function over our CV examples. without the regularization term. The [θ] parameters our cost function calculates its value upon is the theta calculated by minimizing

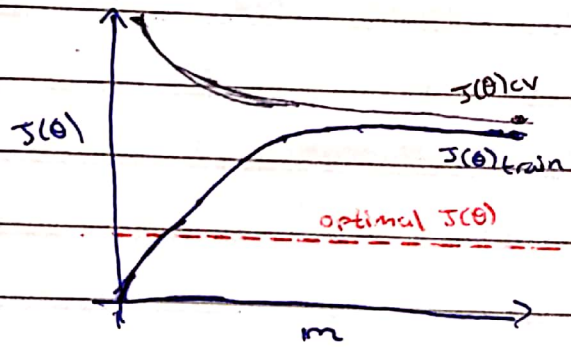
$J(\theta)$ which is the sum of the cost function of our choice and the regularization term with a factor of 'λ'.

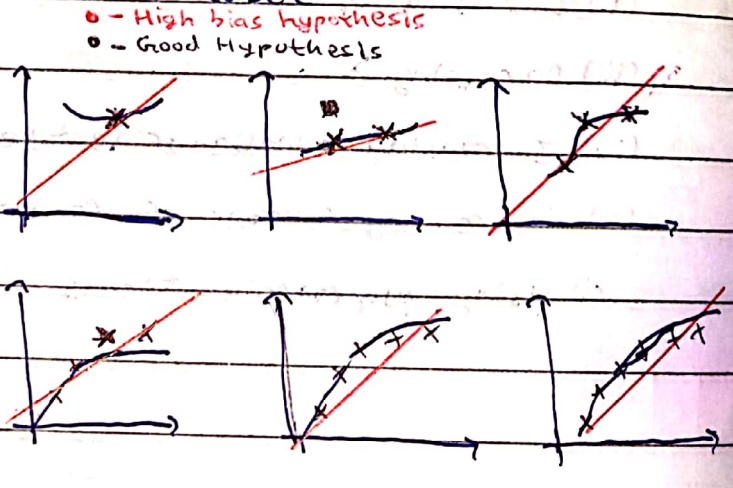


Following the same rules as when determining d, we can diagnose the problem as high bias or high variance and by iterating over different values for λ we can pick our optimal value for our regularization parameter 'λ'.

(3) LEARNING CURVES (Iterating over m)

Here $J(\theta)_{train}$ is the cost function on examples $(x^{(i)}, y^{(i)})$ from $i = 1 \to m$. $[\theta]$ is the parameter vector obtained by training on examples $(x^{(i)}, y^{(i)})$ from $i = 1 \to m$. $J(\theta)_{cv}$ is the cost function on examples $(x^{(i)}, y^{(i)})$ from all the examples in the cv dataset.



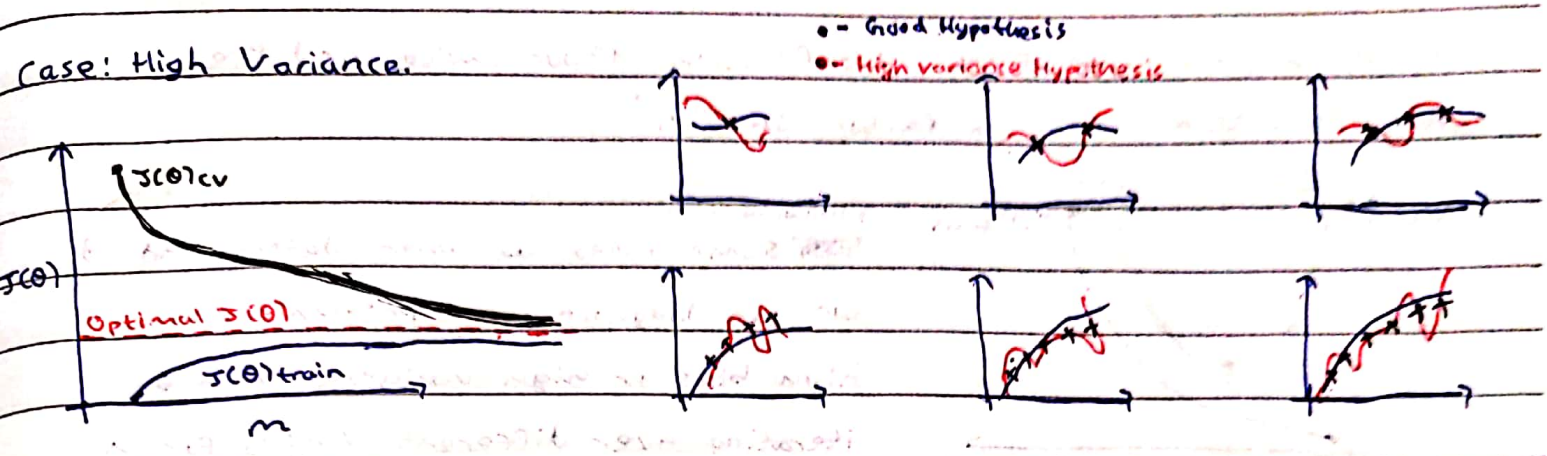


Case: High Bias

~~As is observed~~

From the above curve, we can infer that increasing the number of training examples alone cannot correct high bias problems.

Case: High Variance.



From the above observed curve, we can infer that increasing training examples does indeed help lower $J(\theta)_{cv}$. The 2 curves do converge eventually.

OPTIMIZING THE LEARNING ALGORITHM.

Potential methods & when to use them: ● - Problem it fixes

(i) Increasing training examples - High variance

(ii) Increasing λ - High variance.

(iii) Decreasing λ - High Bias.

(iv) Increasing number of features - High Bias

(v) Decreasing number of features - High variance

(vi) Using polynomial terms - High Bias

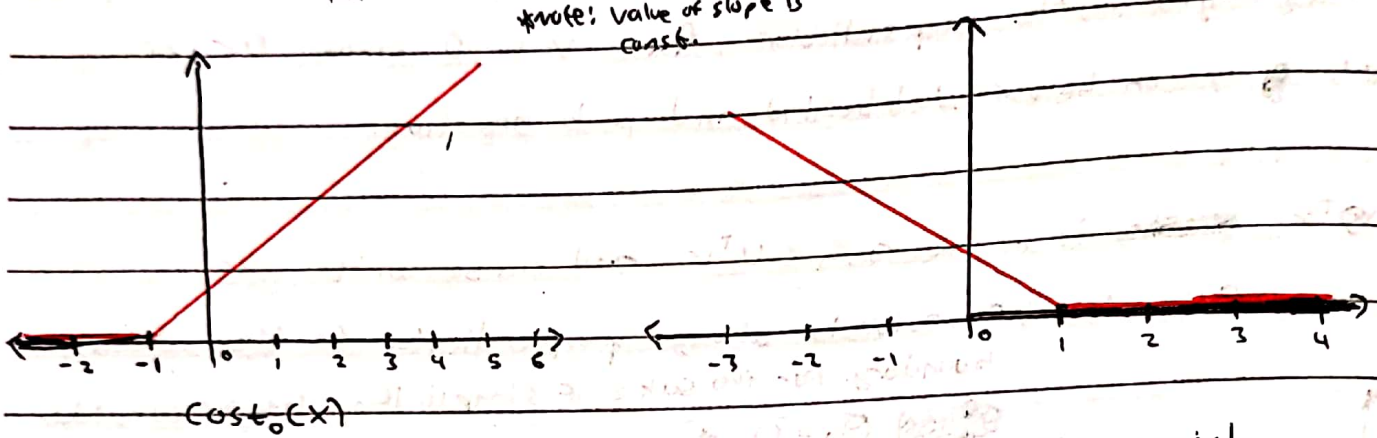
SVMs (SUPPORT VECTOR MACHINES)

Also called Large Margin Classifiers.

COST FUNCTION:

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

*note: value of slope is const.



$\text{cost}_0(x)$

The slope of the line does not matter much.

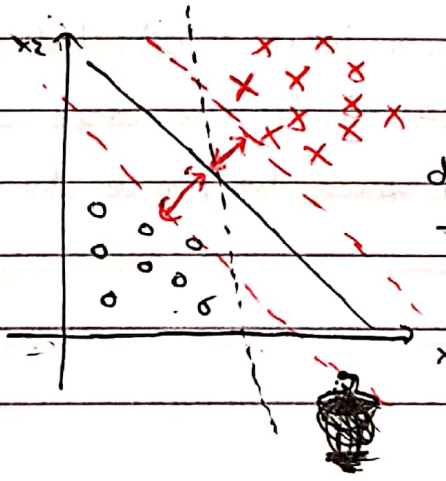
$$h_\theta(\theta^T x) \begin{cases} 0 \text{ when } h_\theta(x) \leq 0 \\ 1 \text{ when } h_\theta(x) \geq 0 \end{cases}$$

This cost function encourages our SVM to not 'just' predict the answer but instead predict it with more certainity.

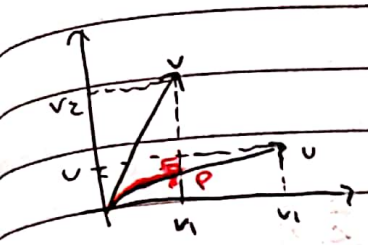
If we make C very large it will try to minimize the first term to 0.

$\therefore \theta^T x \geq 1$ (or) $\theta^T x \leq -1$ Then the minimization objective becomes

$h_\theta(x) = 1$ $h_\theta(x) = 0$. $\rightarrow C \times 0 + \sum_{i=1}^{n}\theta_i^2 = \sum_{i=1}^{n}\theta_j^2$



While both decision boundaries work, the SVM chooses the decision boundary marked by the non-dotted line. This is because it chooses the decision boundary with the maximum margin between it and the datapoints.



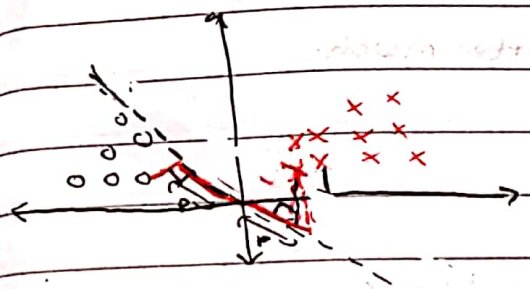
$$U = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad V = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$U^T V = u_1 v_1 + u_2 v_2 = p \|U\|$$

? can be proved.

p is the length of the perpendicular from V to U when $\theta < 180°$. $\theta > 180°$, $-v$ can be extended behind and p is negative.

$\theta^T x$ is of the form $U^T V$ and can be written as $p \|\theta\|$. Note: θ vector is always perpendicular to the decision boundary. For the sake of simplicity, let us consider $\theta_0 = 0$.



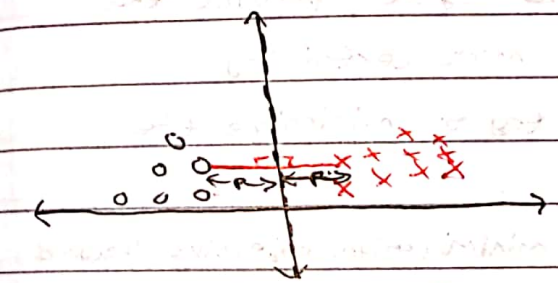
Minimization objective: for $y=0$ ($\theta^T x \leq -1$) for $y=1$ ($\theta^T x \geq 1$)

$\min \sum_{i=1}^{n} \theta_i^2$ when $\theta^T x \leq -1$ or $\theta^T x \geq 1$

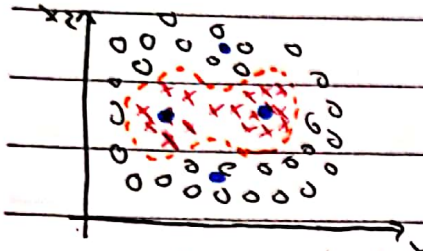
$\therefore \theta^T x = p \cdot \|\theta\|$

$\therefore$ IF $p\|\theta\| \geq 1$ or $p\|\theta\| \leq -1$

and if we are trying to minimize $\sum \theta^2 = \sum (\|\theta\|)^2$ then we must maximize p. p is the orthogonal distance between the examples and the decision boundary. Therefore, for large values of C it will choose the decision boundary which provides a large margin between itself and the training examples. However, if C is too large then it may be hypersensitive to outliers and hence be prone to overfitting.



KERNELS: They provide a new way for us to create complicated decision boundaries without using very high order polynomial features



- ← Decision boundary we wish to draw.
- ← Let us choose these 4 landmarks.

We can now compute a new set of features $f_1, f_2, f_3 \in f_4$ which are a measure of the similarity [distance] between $x^{(i)}$ & l_1, l_2, l_3 & l_4.

We can measure similarity through a 'kernel' function or similarity function.

$$f_j^{(i)} = \text{similarity}(x^{(i)}, l_j) = \exp\left(\frac{-\|x^{(i)} - l_j\|^2}{2\sigma^2}\right)$$

If we use no kernels & just use $\theta^T x$, it is also called a linear kernel.

Other kernels include the polynomial kernel

standard form: $f_j = (x^{(i)} * l_j + c)^n$

& other kernels which are specific to different use cases.

SVM training libraries have well optimized methods to train using features 'f' to train on instead of (computed using a gaussian kernel) instead of x, we can compute complicated decision boundaries.

While the concept of kernels can be used with other learning algorithms, kernels are specially optimized to be used with SVMs.

BIAS vs VARIANCE

Large C - High variance	Small C - Low variance
Large σ - High bias	Small C - Low Bias

*Note: Gaussian kernels can be slow to compute when using a large number of features since the no. of features generated = m.

While using SVM's landmarks are placed at all positions occupied by all training examples.

$x^{(i)} = l^{(i)}$ or $x = l$.

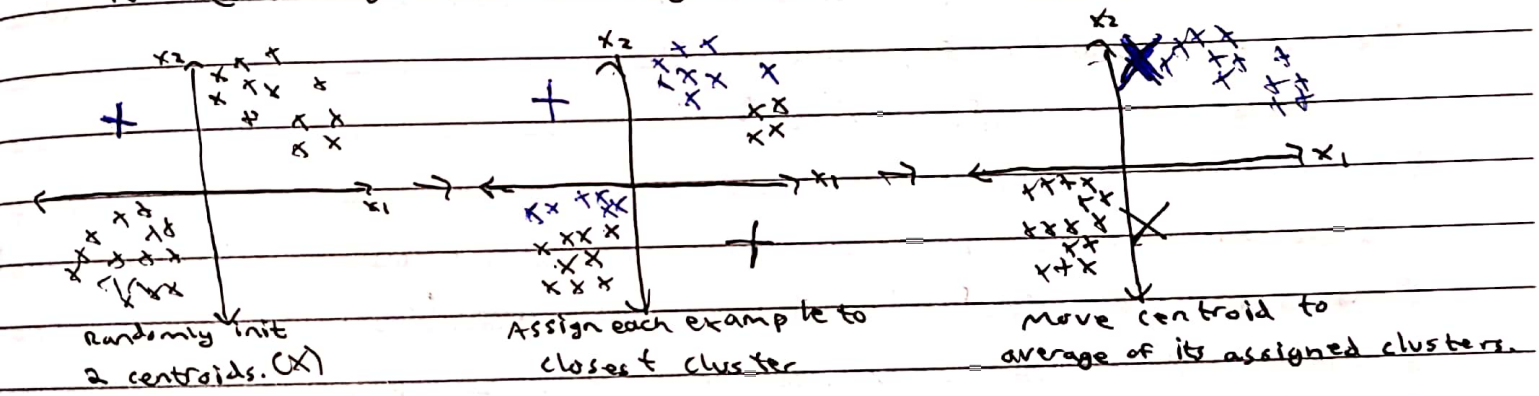
UNSUPERVISED LEARNING

The primary difference between supervised and unsupervised learning is that the training examples / data provided to an unsupervised learning algorithm is unlabelled.

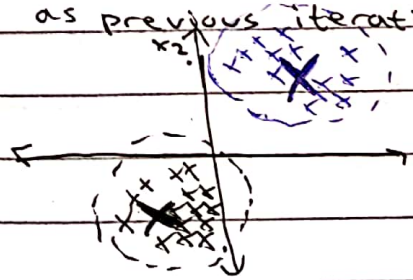
Training data → $\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}\}$ i.e., there is no $y^{(i)}$

Unsupervised learning algorithms try to find structure in unlabelled data

K-Means Algorithm - Widely used to find clusters in unlabelled data



→ Repeat algorithm until next position of each cluster is the same as previous iteration.



Result of k-means.

→ Two clusters were formed.

K-Means Algorithm.

- Randomly initiate K-cluster centroids ($\mu_1, \mu_2, \ldots, \mu_K$) at randomly chosen $x^{(i)}$ positions
- Assign $c^{(i)}$ to cluster k per example $x^{(i)}$ which minimizes the $\|x^{(i)} - \mu_k\|^2$ squared distance between them, i.e., the cluster k closest to $x^{(i)}$
- Move μ_k to the average of all the examples labeled to cluster k

OPTIMIZATION OBJECTIVE

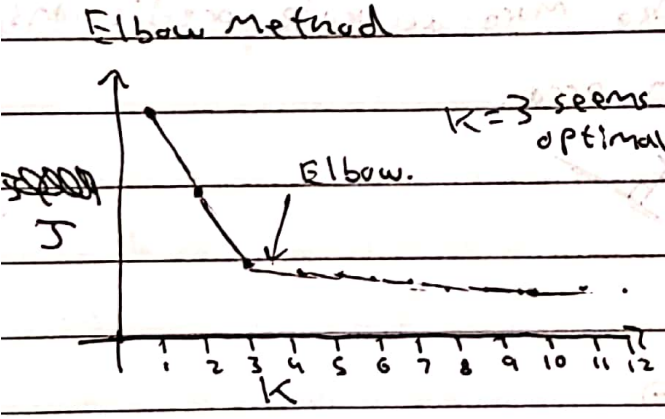
Cost function: $J(c^{(1)}, c^{(2)}, \ldots, c^{(m)}, \mu_1, \mu_2, \ldots \mu_K) = \frac{1}{m} \sum_{i=1}^{m} \|x^{(i)} - \mu_{c^{(i)}}\|^2$

It is possible for K-means to land in local optima for different random initializations of clusters $\mu_1, \ldots, \mu_K$.

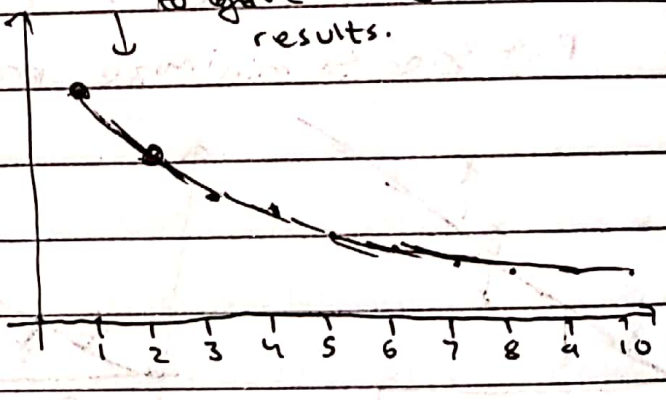
So it is advisable to run K-Means multiple times with different random initializations for the clusters.

CHOOSING K:

Elbow Method



But often real world examples tend to give more unambiguous results.



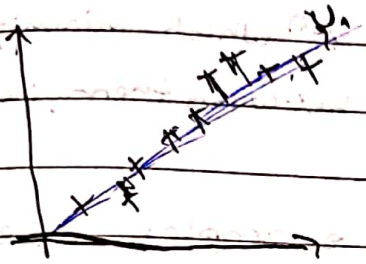
Hence the elbow method might not always work and it is better to handpick K depending on the situation/purpose for classification

DEBUGGING: Cost function J should always decrease or remain same with increase in K. If not then possible reasons are local optima & faulty code/implementation.

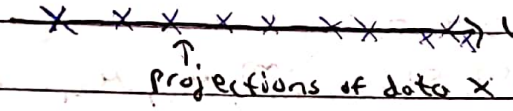
PRINCIPAL COMPONENT ANALYSIS (PCA)

It is used to reduce the dimensions of some given data.

Ex: Given 2D data



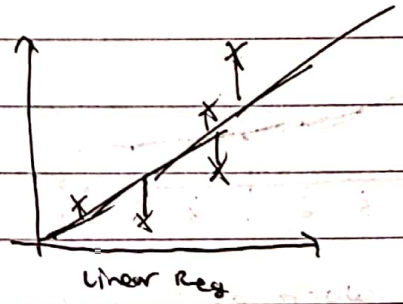
PCA helps find the vector $U_1 \in R^2$ such that the projection error, i.e., the perpendicular distance from the data point to the surface provided by the vector is minimum.

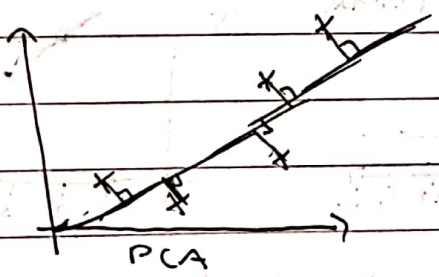


projections of data x on U_1 which now provide 1D vector of features $z \in R^1$

In general, PCA finds the direction / surface (upto 3D) i.e., the vectors $U_1, U_2, \ldots, U_K$ onto which to project the data X such that projection error between the data points & the vectors are minimum.

*Note: While linear regression tries to reduce the error between label y and prediction $h_\theta(x)$ to make more accurate predictions, PCA tries to minimize projection error.





THE PCA ALGORITHM

To reduce data of n-dimensions to K-dimensions.

Given $x^{(i)}$ is a row vector of all the features, i.e., $X \in \mathbb{R}^n$

* Preprocess data by applying mean normalization. Also feature scaling if necessary.

- Compute covariance matrix (Σ)

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

Vectorized implementation: If $X = [m \times n]$

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

Then covariance matrix

$$\Sigma = \left(\frac{1}{m}\right) * X' * X$$

- Compute eigen vectors of matrix Σ

- Using Singular Value Decomposition

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

- Using eigenvector function [only applies to square matrices]

$$[U, S, V] = \text{eig}(\text{covariance matrix});$$

Matrix U is a $n \times n$ matrix which contains the vectors $U_1, U_2, \ldots U_n$ as its columns.

$$\begin{bmatrix} | & | & & | \\ U_1 & U_2 & \cdots & U_n \\ | & | & & | \end{bmatrix}$$

svd can also be used directly on the data matrix X, and matrix V would contain our eigen vectors. We apply svd on the Σ matrix since $n \times n$ is generally more computably reasonable than $n \times n$ and potentially for the right S matrix. For a covariance matrix, U & V are the same.

Matrix U:

$$\begin{bmatrix} | & | & | & & | \\ u_1 & u_2 & u_3 & \dots & u_n \\ | & | & | & & | \end{bmatrix}$$

K

we select the first K vectors.

$U_{reduce} = U(:, 1:K)$;

$x^{(i)} \in R^n$ & $z^{(i)} \in R^K$

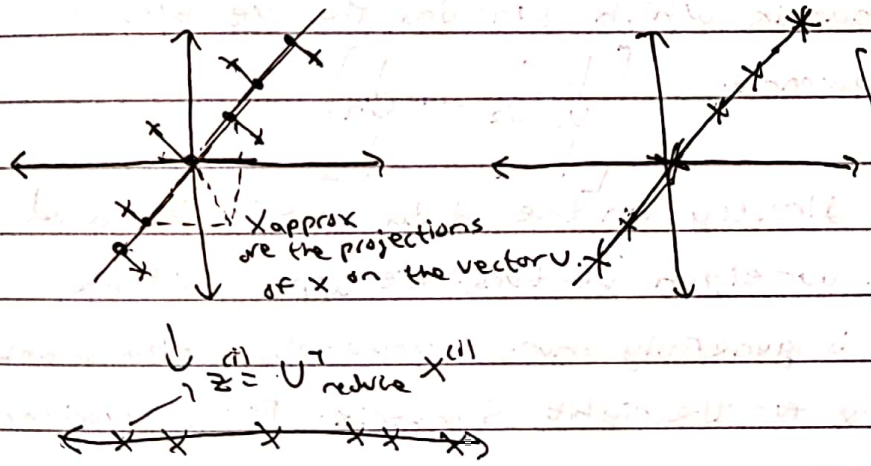
$$z^{(i)} = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \cdots & u_K \\ | & | & & | \end{bmatrix}^T x^{(i)} = \underset{K \times n}{\begin{bmatrix} — u_1 — \\ — u_2 — \\ \vdots \\ — u_K — \end{bmatrix}} \times \underset{n \times 1}{\left[x^{(i)}\right]} = \underset{K \times 1}{\left[z^{(i)}\right]}$$

Ex:

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \end{bmatrix} \times \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ x_3^{(i)} \end{bmatrix} = \begin{bmatrix} u_{11} \cdot x_1^{(i)} + u_{12} \cdot x_2^{(i)} + u_{13} \cdot x_3^{(i)} \\ u_{21} \cdot x_1^{(i)} + u_{22} \cdot x_2^{(i)} + u_{23} \cdot x_3^{(i)} \end{bmatrix}$$

It maps the ~~[illegible]~~ data point x onto the eigen vector.

RECONSTRUCTION FROM REDUCED DIMENSION COMPRESSION



$$\underset{n \times 1}{\left[x_{approx}^{(i)}\right]} = \underset{n \times k}{\left[U_{reduce}\right]} \times \underset{k \times 1}{\left[z^{(i)}\right]}$$

CHOOSING A GOOD VALUE OF K FOR DIMENSIONALITY REDUCTION

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

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

We choose K to be the smallest natural number which satisfies the condition. → $\frac{\text{Average Squared Projection Error}}{\text{Total Variation in Data}} \leq 0.01$

0.01 → 1%, i.e, we can say that 99% of variance was retained

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

Acceptable values for variance retained are values upward of 85%

While applying svd,

$$[U, S, V] = \text{svd}(\text{co-variance matrix } \Sigma)$$

$$S = \begin{bmatrix} \sigma_{11} & & & \\ & \sigma_{22} & & \\ & & \ddots & \\ & & & \sigma_{nn} \end{bmatrix}_{n \times n}$$

→ It is a diagonal matrix.

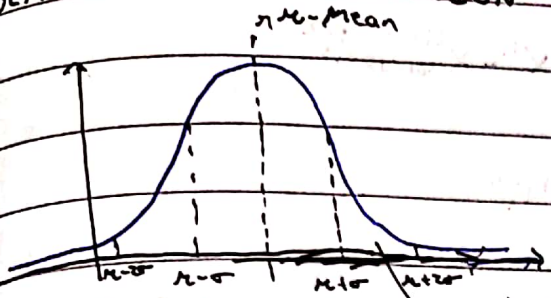
For a given k, we can calculate variance retained by.

99% variance retained

$$1 - \frac{\sum_{i=1}^{k} \sigma_{ii}}{\sum_{i=1}^{n} \sigma_{ii}} \geq 0.99$$

ANOMALY DETECTION ALGORITHM

GAUSSIAN DISTRIBUTION



σ - Standard deviation

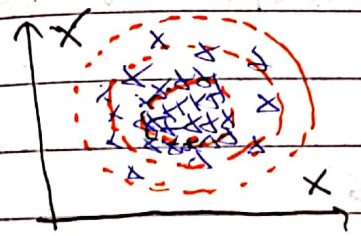
σ^2 - variance

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

Parametrised by μ & σ

Area under graph is always equal to one.

If we have features that can be mapped as a gaussian distribution we can use them to create an anamoly detection algorithm. Use hist(x) command in octave to see the histogram plot of our data. We can use transformations such as $\log(x+c)$ or x^c or $x^{\frac{1}{2}}$ to make our data distribution more similar to a normal gaussian distribution. We can also create new features which involve an operation between 2 existing features.

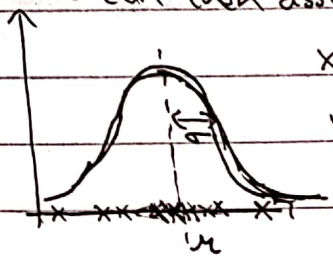


X - Anomalous data.

If $h_\theta(x) < \varepsilon$ ε Treshold constant, then it is classified as anomalous.

If our training data resembles a gaussian distribution, we can assume the data source to be distributed normally.

We can then assume x to come from a gaussian distribution.



$x^{(i)} \sim N(\mu, \sigma^2)$ $\quad x^{(i)} \in R$

We can now estimate parameters μ & σ using the dataset $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

The estimated μ

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

Now when $x^{(i)} \in R^n$ Then we find parameters μ_j & σ_j^2 for each feature of example $x^{(i)}$

Then $x_1 \sim N(\mu_1, \sigma_1^2)$, $x_2 \sim N(\mu_2, \sigma_2^2)$, ..., $x_j \sim N(\mu_j, \sigma_j^2)$

$$\therefore p(x) = p(x_1; \mu_1, \sigma_1^2) \times \ldots \times p(x_n; \mu_n, \sigma_n^2)$$

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

ALGORITHM:

1. Choose features x_j which you think are indicative of anomalous examples. It helps if these features also resemble a gaussian distribution. Use functions to transform them to resemble gaussian distributions if necessary & possible.

2. Calculate parameters $\mu_1, \ldots, \mu_n$ & $\sigma_1^2, \ldots, \sigma_n^2$

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

3. Given new example x, calculate $p(x)$ using estimated parameters.

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

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

Given a training set of ~~majority~~ minority anomalous & majority normal data

60% normal examples → training set

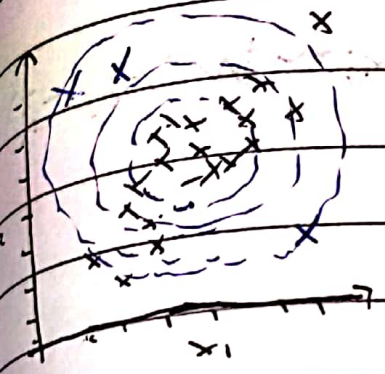
20% normal examples + 50% anomalous examples → CV-set

20% normal examples + 50% anomalous examples → test set.

Use metrics such as precision, recall, F1 score, etc to measure accuracy.

Can use CV-set to choose value of ε

MULTIVARIATE GAUSSIAN DISTRIBUTION



Given two features which vary linearly, the features in blue look like anomalies. But the normal model cannot classify them as anomalies. We need to create a feature $x_3 = \frac{x_2}{x_1}$ or something similar.