Supervised Learning

Linear Regression
**Trend*
**Market estimates*
**Forecasts*

Step 1. Hypothesis:

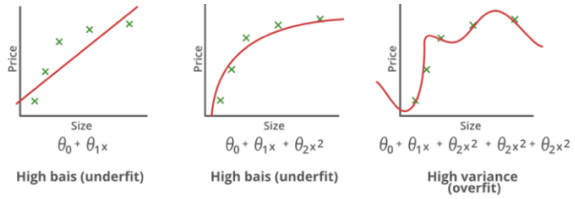
$$h_\theta(x)$$

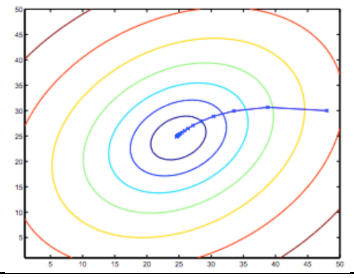
Step 2. Cost

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

Step 3: Gradients

$$\begin{cases} \theta_0 \coloneqq \theta_0 - \alpha\dfrac{1}{m}\displaystyle\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_0^{(i)} \\ \theta_j \coloneqq \theta_j\left(1 - \alpha\dfrac{\lambda}{m}\right) - \alpha\dfrac{1}{m}\displaystyle\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}, j = 1,2,3,\dots,n \end{cases}$$





Logistic Regression
**Binary classes*

Step 1. Hypothesis:

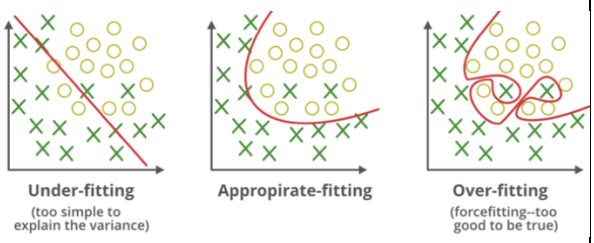
$$\begin{cases} h_\theta(z) = g(\theta^T x) \\ z = \theta^T x \\ g(z) = \dfrac{1}{1+e^{-z}} \end{cases}$$

Step 2. Cost

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

Step 3. Gradients:

$$\begin{cases} \theta_0 \coloneqq \theta_0 - \alpha\dfrac{1}{m}\displaystyle\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_0^{(i)} \\ \theta_j \coloneqq \theta_j\left(1 - \alpha\dfrac{\lambda}{m}\right) - \alpha\dfrac{1}{m}\displaystyle\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}, j = 1,2,3,\dots,n \end{cases}$$

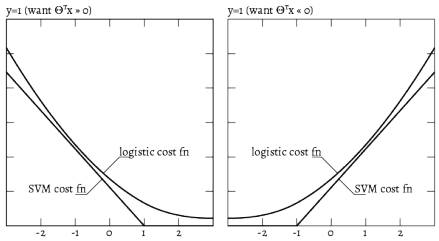


Support vector machines (SVM)

Cost

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

$$y^{(i)} = \begin{cases} 1, \ \theta^T x^{(i)} \geq 1 \\ 0, \ \theta^T x^{(i)} \leq -1 \end{cases}$$



SVM with Gaussian Kernel

Step 1. Hypothesis

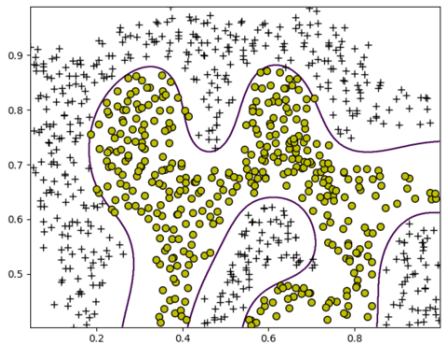
Given x, compute features $f \in \mathbb{R}^{m+1}$, *parameters* $\theta \in \mathbb{R}^{m+1}$
Predict "y=1" if $\theta^T f \geq 0, \ \theta_0 f_0 + \theta_1 f_1 + \cdots + \theta_m f_m \geq 0$

Step 2. Training

$$min\, J(\theta) = C\sum_{i=1}^{m}\left[y^{(i)}\, cost_1(\theta^T f_i) + \left(1-y^{(i)}\right) cost_0(\theta^T f_i)\right] + \frac{1}{2}\sum_{j=1}^{m}\theta_j^2$$

$$f_i = similarity\left(x, l^{(i)}\right) = \exp\left(-\frac{\left\|x - l^{(i)}\right\|^2}{2\sigma^2}\right), or = \left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)$$

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



Neural network

**Pattern recognition*
**Fraud detection*
**Deep learning.*

Step 1. Randomly initialize weights

Initialize parameters $\Theta^{(1)}, \Theta^{(2)}, \dots, \Theta^{(L-1)}$ $[-\epsilon, \epsilon]$(i.e. $-\epsilon \le \Theta_{ji}^{(l)} \le \epsilon$)

Step 2. Forward propagation

$$h_\Theta(x^{(i)}) \in \mathbb{R}^K \quad (h_\Theta(X))_i = i^{th}\ output$$

$a^{(1)} = x$
$z^{(2)} = \Theta^{(1)} a^{(1)}$
$a^{(2)} = g(z^{(2)})\ \ (add\ a_0^{(2)})$
$z^{(3)} = \Theta^{(2)} a^{(2)}$
$a^{(3)} = g(z^{(3)})\ \ (add\ a_0^{(3)})$
$z^{(4)} = \Theta^{(3)} a^{(3)}$
$a^{(4)} = h_\Theta(x) = g(z^{(4)})$

Step 3. Cost function $J(\Theta)$

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)} \log\left(h_\Theta(x^{(i)})\right)_k + \left(1 - y_k^{(i)}\right)\log\left(1 - \left(h_\Theta(x^{(i)})\right)_k\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{(l)}\right)^2$$

Step 4. Backpropagation to compute partial derivatives

$$\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$$

$\delta_j^{(l)}$ = "error" of node j in layer l.

$$\delta^{(2)} = \left(\Theta^{(2)}\right)^T \delta^{(3)} * g'(z^{(2)})$$
$$g'(z^{(2)}) = a^{(2)} * (1 - a^{(2)})$$
$$\delta^{(3)} = \left(\Theta^{(3)}\right)^T \delta^{(4)} * g'(z^{(3)})$$
$$g'(z^{(3)}) = a^{(3)} * (1 - a^{(3)})$$
$$\delta^{(4)} = a^{(4)} - y$$

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

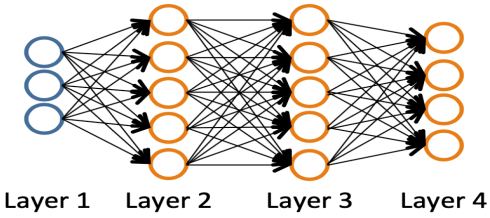
$$D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)}, if\ j \ne 0$$
$$D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)}, if\ j = 0$$

$$D_{ij}^{(l)} = \frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = \frac{\partial J(\Theta)}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial \Theta}$$

Step 5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using backpropagation vs. using numerical estimate of gradient of $J(\Theta)$.

Step 6. Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ .

```
result = minimize(cost_func, initial_nn_params, method='CG',
jac=grad_func,
          options={'disp': True, 'maxiter': 50.0})
nn_params = result.x
Jcost = result.fun
```



Unsupervised Learning

K-means

Step 1. Centroids

$$c^{(i)} = index\ of\ min\left\|x^{(i)} - \mu_j\right\|^2$$

$c^{(i)} \in \mathbb{R}^K, \mathrm{i} = 1,2,\dots,\mathrm{m}$ denotes the index of cluster centroids closet to $x^{(i)}$

Step 2. Means

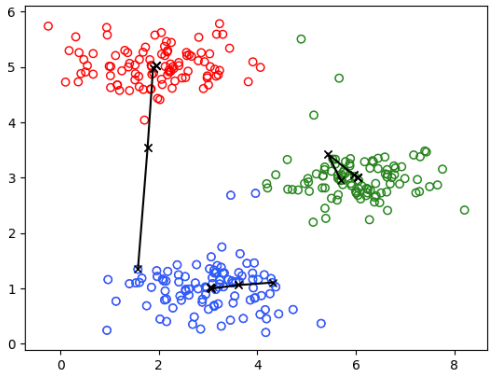
$$\mu_k = \frac{\sum_{i=1,\ \{c^{(i)}=k\}}^{m} x^{(i)}}{\sum_{i=1,\ \{c^{(i)}=k\}}^{m} 1}$$

$\mu_k \in \mathbb{R}^K, k = 1,2,\dots,K$ denotes the average(mean) of points assigned to cluster k

Step 3. Cost function

$$\boldsymbol{J_{(c,\mu)} = \sum_{i}^{m} \left\|x^{(i)} - \mu_{c^{(i)}}\right\|^2}$$

K=3



Principal Component Analysis (PCA)

**Dimensionality Reduction, *Facial recognition, *Data compression, *Computer vision and image compression*

Step1. Feature scaling (Mean normalization)

Mean: $\bar{X} = \mu_j = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$

Standard deviation: $s = \sigma = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(x^{(i)} - \mu)^2}$

Mean normalize: $x^{(i)} = \frac{x^{(i)} - \mu}{\sigma}$

Step 2. Calculate U, S, V.

$$\boldsymbol{sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T = \frac{1}{m}X^T X}$$

U, S, V = numpy.linalg.svd(sigma)

$$\boldsymbol{U} = \begin{pmatrix} | & | & | & | & | & | \\ \boldsymbol{u^{(1)}} & \boldsymbol{u^{(2)}} & \dots & \boldsymbol{u^{(k)}} & \dots & \boldsymbol{u^{(n)}} \\ | & | & | & | & | & | \end{pmatrix}$$

Ureduce = U[:, 0:K].T

Z = Ureduce*X = X_norm * U[:, 0:K]

X_approximate = X_recovered = Z * U[:, 0:K].T

Step 3. Pick the smallest value of k,

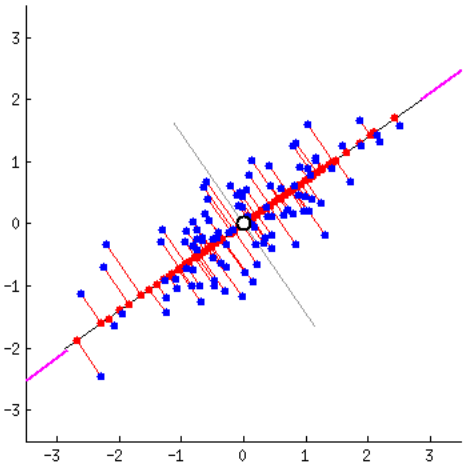
$$\frac{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)} - x_{approx}^{(i)}\right\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \leq 0.01?$$

$$S = \begin{pmatrix} S_{11} & & \dots & & 0 \\ & S_{22} & \dots & 0 & \\ \vdots & \vdots & S_{33} & \vdots & \vdots \\ & 0 & \dots & \ddots & \\ 0 & & \dots & & S_{nn} \end{pmatrix}$$

$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \leq 0.01 \ \rightarrow\ \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$$

99% of variance retained

$2D \rightarrow 1D$



Anomaly Detection
**Fraud detection*
**Intrusion detection*
**system health*
**monitoring*

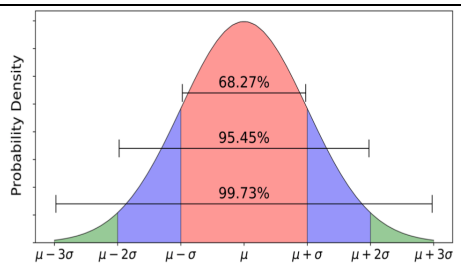
Gaussian (Normal) distribution

$$X \sim N(\mu, \sigma^2)$$

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

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

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



Original model

Step 1. Choose feature

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

Density estimation: $x_j \sim N(\mu_j, \sigma_j^2), j = 1, 2, \ldots, n$

Choose features x_i that might be indicative of anomalous examples.

Step 2. Fit parameters

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

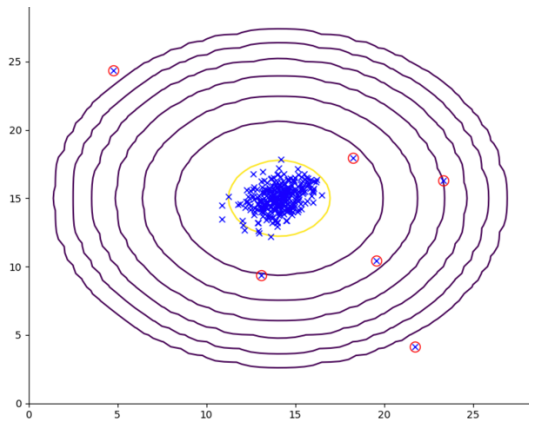
$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$$

Step 3. Given new example $x \in \mathbb{R}^n$, compute $p(x)$

Probability

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

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



Multivariate Gaussian

Step 1. Choose feature

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

Density estimation: $x_j \sim N(\mu_j, \sigma_j^2), j = 1, 2, \ldots, n$

Step 2. Fit parameters

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

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

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

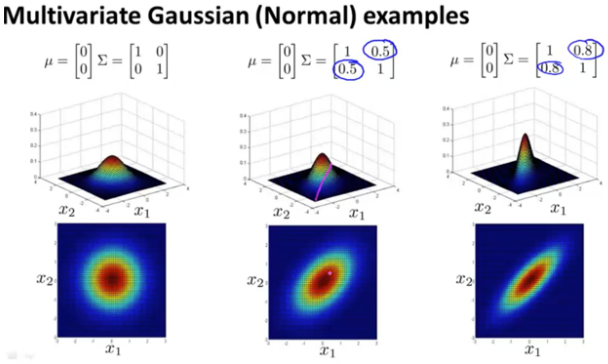
Step 3. Given new example x, compute $p(x)$

Diagonal Sigma: $\Sigma = \begin{pmatrix} \Sigma_1 & & \cdots & & 0 \\ & \Sigma_2 & \ldots & 0 & \\ \vdots & \vdots & \Sigma_3 & \vdots & \vdots \\ & 0 & \ldots & \ddots & \\ 0 & & \cdots & & \Sigma_n \end{pmatrix}$

Probability:

$$p(x_j; \mu_j, \Sigma) = \frac{1}{\sqrt{2\pi}|\Sigma|^{\frac{1}{2}}} e^{\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-u)\right)}$$

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



Machine Learning

Allen Sun

Contents

1 Machine learning

Machine learning is the science of getting computers to learn, without being explicitly programmed.

- **Cost function** – The cost function J is commonly used to assess the performance of a model with parameters θ and hypothesis function $h_\theta(x^{(i)})$.

$$J(\theta) = \sum_{i=1}^{m} C(h_\theta(x^{(i)}), y^{(i)})$$

- This minimization objective is expressed using the following notation, which simply states that we want to find the θ which minimizes the cost $J(\theta)$.

$$\min_\theta J(\theta)$$

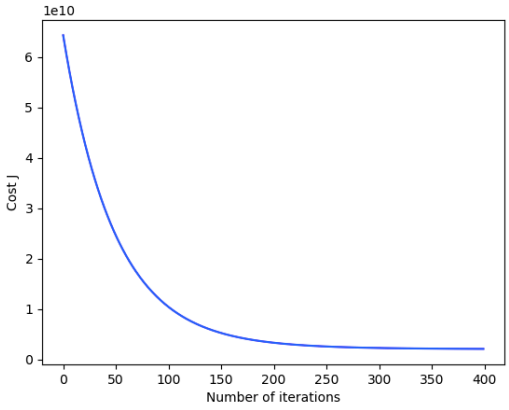


Figure 1 Change of cost function J with iterations

- **Gradient descent –** The gradient descent is used to minimize cost function J with learning rate α and derivative $\Delta j(\theta)$.

$$\theta := \theta - \alpha \Delta j(\theta)$$

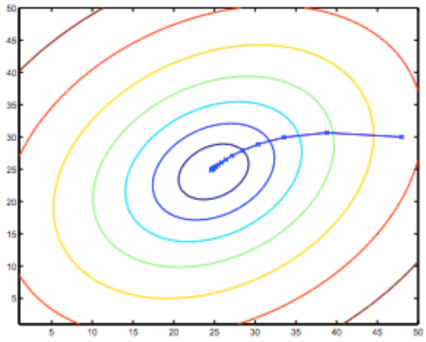


Figure 2 Example of batch gradient descent

2 Supervised learning

Already know what our correct output should look like, having the idea that there is a relationship between the input and the output

2.1 Linear regression

Regression, continuous, individual

1) Data & Hypothesis

Input data: X (m, n); dataset: $x^{(i)}, i = 1,2, \dots, m$; features: $x_j, j = 1,2, \dots, n$;
parameters: θ(n,1); actual output: y(m,1).

$$X_{(m\times n)} = \begin{pmatrix} x_1^{(1)} & \cdots & x_j^{(1)} & \cdots & x_n^{(1)} \\ \vdots & \ddots & \vdots & & \vdots \\ x_1^{(i)} & \cdots & x_j^{(i)} & \cdots & x_n^{(i)} \\ \vdots & & \vdots & \ddots & \vdots \\ x_1^{(m)} & \cdots & x_j^{(m)} & \cdots & x_n^{(m)} \end{pmatrix}, \quad \theta^T{}_{(n\times 1)} = \begin{pmatrix} \theta_1 \\ \vdots \\ \theta_j \\ \vdots \\ \theta_n \end{pmatrix}, \quad Y^T{}_{(m\times 1)} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(i)} \\ \vdots \\ y^{(m)} \end{pmatrix}$$

2) Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^{0,1,2} x_2^{0,1,2} + \cdots + \theta_n \prod_{j=1}^{n} x_j^{0,1,2,\dots,n}$$

3) Cost function:

We can measure the accuracy of our hypothesis function by using a **cost function**. This takes an average difference (actually a fancier version of an average) of all the results of the hypothesis with inputs from x's and the actual output y's.

Minimize cost:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

4) Gradients

When specifically applied to the case of linear regression, a new form of the gradient descent equation can be derived. We can substitute our actual cost function and our actual hypothesis function and modify the equation to:

Rate: α, regulation: λ

$$G(\theta) = \frac{\partial J(\theta)}{\partial \theta}$$

Repeat until convergence: {

$$\theta_0 \coloneqq \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_0^{(i)}$$

$$\theta_j \coloneqq \theta_j\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)},\ j = 1,2,3,\dots,n$$

}

$\alpha\frac{\lambda}{m}$ <1. Intuitively you can see it as reducing the value of θ_j by some amount on every update.

- **Advice:** Debugging a learning algorithm:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

Suppose you have implemented regularized linear regression to predict housing prices. However, when you test your hypothesis on a new set of houses, you find that it makes unacceptably large errors in its predictions. What should you try next?

- Get more training examples
- Try smaller sets of features
- Try getting additional features
- Try adding polynomial features $(x_1^2, x_2^2, x_1, x_{2,} etc.)$
- Try decreasing λ
- Try increasing λ

2.2 Logistic regression

Classification, Discrete, collective

1) Sigmoid function

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

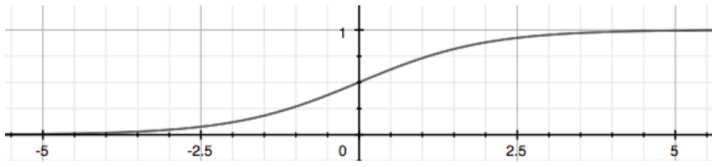
In order to get our discrete 0 or 1 classification, we can translate the output of the hypothesis function as follows:

$$\begin{cases} h_\theta(x) \geq 0.5 \Longrightarrow \boldsymbol{y = 1} \\ h_\theta(x) < 0.5 \Longrightarrow \boldsymbol{y = 0} \end{cases}$$

The way our logistic function g behaves is that when its input is greater than or equal to zero, its output is greater than or equal to 0.5:

$$g(z) \geq 0.5, \qquad when\ z \geq 0$$

The following image shows us what the sigmoid function $g(z)$ looks like:



$$z = 0, e^0 = 1 \Longrightarrow g(z) = 1/2$$
$$z \to \infty, e^{-\infty} \to 0 \Longrightarrow g(z) = 1$$
$$z \to -\infty, e^{\infty} \to \infty \Longrightarrow g(z) = 0$$

If our input to g is $\theta^T x$, then that means:

$$\begin{cases} h_\theta(z) = g(\theta^T x) \geq 0.5, \theta^T x \geq 0 \\ h_\theta(z) = g(\theta^T x) < 0.5, \theta^T x < 0 \end{cases}$$

From these statements we can now say:

$$\begin{cases} \theta^T x \geq 0 \Longrightarrow \boldsymbol{y = 1} \\ \theta^T x < 0 \Longrightarrow \boldsymbol{y = 0} \end{cases}$$

2) Cost function:

Minimize cost:

$$min\, J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) + (1-y^{(i)})\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

3) Gradients descent:
when computing the equation, we should continuously update the two following equations:
Repeat: {

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_0^{(i)}$$

$$\theta_j := \theta_j\left(1-\alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}, \qquad j = 1,2,3,\dots,n$$

}

2.3 Support vector machine (SVM)

SVM gives a cleaner, and more powerful way of learning complex non-linear functions.

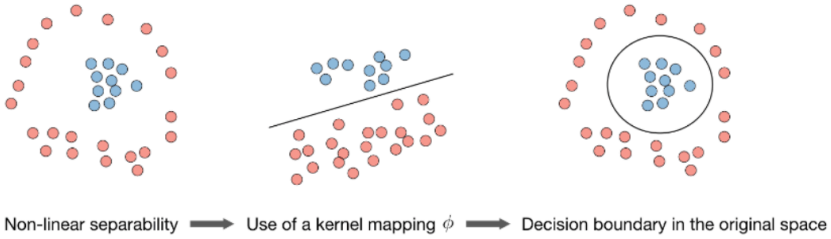


Figure 3 SVM Kernel mapping

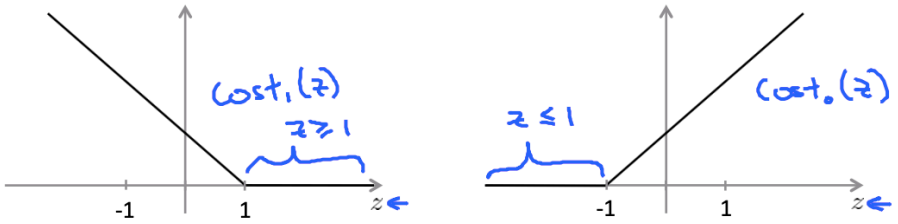
2.3.1 No kernel ("linear kernel")

Predict "y = 1" if $\boldsymbol{\theta^T x > 0}$

1) Hypothesis:

$$\boldsymbol{h_\theta(x) = \begin{cases} 1, & \theta^T x > 0 \\ 0, & otherwise \end{cases}}$$

2) Cost function:



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

$$y^{(i)} = \begin{cases} 1, & \theta^T x^{(i)} \geq 1 \\ 0, & \theta^T x^{(i)} \leq -1 \end{cases}$$

Large C: Lower bias, high variance.
Small C: Higher bias, low variance.

2.3.2 SVM with kernels, also called gaussian kernel

1) Hypothesis

Given x, compute features $f \in \mathbb{R}^{m+1}$, *parameters* $\theta \in \mathbb{R}^{m+1}$

Predict "y=1" if $\theta^T f \geq 0,\ \theta_0 f_0 + \theta_1 f_1 + \cdots + \theta_m f_m \geq 0$

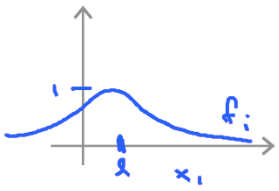
2) Training

$$min\, J(\theta) = C\sum_{i=1}^{m}\left[y^{(i)}\, cost_1(\theta^T f_i) + (1-y^{(i)})\, cost_0(\theta^T f_i)\right] + \frac{1}{2}\sum_{j=1}^{m}\theta_j^2$$

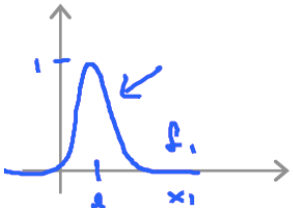
$$f_i = similarity\left(x, l^{(i)}\right) = \exp\left(-\frac{\left\|x - l^{(i)}\right\|^2}{2\sigma^2}\right), or = \left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)$$

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

Large σ^2 : Features f_i vary more smoothly.
Higher bias, lower variance.



Small σ^2 : Features f_i vary less smoothly.
Lower bias, higher variance.



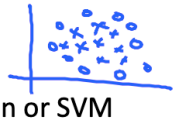
3) Multiclass classification

Many SVM packages already have built-in multiclass classification functionality.
Otherwise, use one-vs.-all method. (Train K SVMs, one to distinguish $y = i$ from the rest, for $i = 1, 2, \ldots, K$), get $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$Pick class i with largest

Logistic regression vs. SVMs

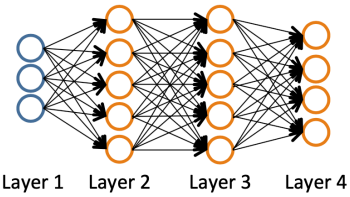
n =number of features ($x \in \mathbb{R}^{n+1}$), m = number of training examples

- If n is large (relative to m): (e.g. $n \geq m$, n = 10,000 , m = 10 ··· 1000)
  - Use logistic regression, or SVM without a kernel ("linear kernel")
- If n is small, m is intermediate: (n = 1-1000 , m = 10 - 10,000)
  - Use SVM with Gaussian kernel
- If n is small, m is large: (n = 1-1000, m = 50,000+)
  - Create/add more features, then use logistic regression or SVM without a kernel
- Neural network likely to work well for most of these settings, but may be slower to train.



2.4 Neural Network

Non-linear Classification



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

$L = total\ no.\ of\ layers\ in\ network$

S_l = no. of units (not counting bias unit) in layer l

Binary classification

$y = 0\ or\ 1$

1 output unit

$$h_\Theta \in \mathbb{R}$$
$$S_L = 1 \ \ (K = 1)$$

Multi-class classification (K classes)

$y \in \mathbb{R}^K$ E.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

$pedestrian\ \ car\ \ motor\ \ truck$

K output units

$$h_\Theta \in \mathbb{R}^K$$
$$S_L = K \ \ (K \geq 3)$$

- Training a neural network (e.g. L=4)
- Pick a network architecture (connectivity pattern between neurons)
- No. of input units: Dimension of features
- No. output units: Number of classes
- Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)

1) Randomly initialize weights

Initialize parameters $\Theta^{(1)}, \Theta^{(2)}, \dots, \Theta^{(L-1)}$

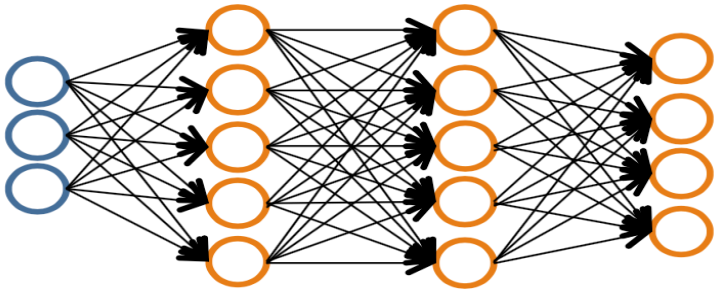
Initialize each $\Theta_{ji}^{(l)}$ to a random value in $[-\epsilon, \epsilon]$(i.e. $-\epsilon \leq \Theta_{ji}^{(l)} \leq \epsilon$)

epsilon = 0.12

```
theta = np.random.rand(input_layer, output_layer) * 2*epsilon - epsilon
```

2) Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$

$$h_\Theta(x^{(i)}) \in \mathbb{R}^K \ \ (h_\Theta(X))_i = i^{th}\ output$$



Layer 1 (Input layer): $a^{(1)} = x$

Layer 2 (Hidden layers): $z^{(2)} = \Theta^{(1)} a^{(1)}$, $a^{(2)} = g(z^{(2)})$

Layer 3 (Hidden layers): $z^{(3)} = \Theta^{(2)} a^{(2)}$, $a^{(3)} = g(z^{(3)})$

Layer 4 (Output layer): $z^{(4)} = \Theta^{(3)} a^{(3)}$, $a^{(4)} = h_\Theta(x) = g(z^{(4)})$

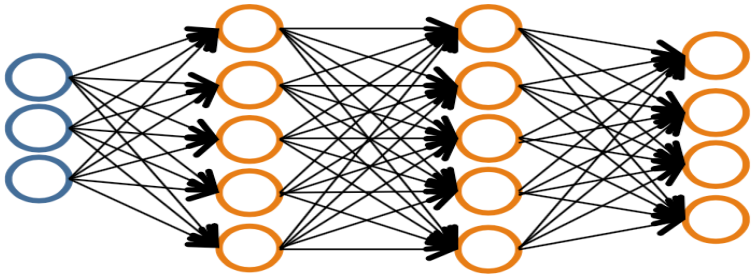
$(add\ a_0^{(2)})$ $(add\ a_0^{(3)})$

3) Implement code to compute cost function $J(\Theta)$

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log\left(h_\Theta(x^{(i)})\right)_k + \left(1-y_k^{(i)}\right)\log\left(1-\left(h_\Theta(x^{(i)})\right)_k\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{(l)}\right)^2$$

4) Implement backpropagation to compute partial derivatives $\frac{\partial}{\partial\Theta_{jk}^{(l)}}J(\Theta)$

$\delta_j^{(l)}$ = "error" of node j in layer l.



$$\delta^{(2)} = \left(\Theta^{(2)}\right)^T\delta^{(3)} * g'\left(z^{(2)}\right) \qquad \delta^{(3)} = \left(\Theta^{(3)}\right)^T\delta^{(4)} * g'\left(z^{(3)}\right) \qquad \delta^{(4)} = a^{(4)} - y$$

$$g'\left(z^{(2)}\right) = a^{(2)} * \left(1 - a^{(2)}\right) \qquad g'\left(z^{(3)}\right) = a^{(3)} * \left(1 - a^{(3)}\right)$$

Perform forward propagation and backpropagation using example $\left(x^{(i)}, y^{(i)}\right)$
(Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \ldots, L$).

- Set $\Delta_{ij}^{(l)} = 0$ (from all, l, i, j).
- Set $a^{(1)} = x$
- Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$
- Using y, compute $\delta^{(L)} = a^{(L)} - y$
- Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

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

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

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

$$D_{ij}^{(l)} = \frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta) = \frac{\partial J(\Theta)}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial\Theta}$$

5) Use gradient checking to compare $\frac{\partial}{\partial\Theta_{jk}^{(l)}}J(\Theta)$ computed using backpropagation vs. using numerical estimate of gradient of $J(\Theta)$.

i. Implement backpropagation to compute DELTA VECTOR (unrolled $D^{(1)}, D^{(2)}, \ldots$).
ii. Implement numerical gradient check to compute **gradient approximation**.
iii. Make sure they give similar values.

diff = slin.norm(numgrad-grad)/slin.norm(numgrad+grad)

```
print('If your backpropagation implementation is correct, then \n\
      the relative difference will be small (less than 1e-9). \n\
      \nRelative Difference: ', diff)
```

iv. Turn off gradient checking. Using backprop code for learning.

- **Important:** Be sure to disable your gradient checking code before training your classifier. If you run numerical gradient computation on every iteration of gradient descent (or in the inner loop of cost function your code will be very slow).

6) Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ .

```
result = minimize(cost_func, initial_nn_params, method='CG', jac=grad_func,
          options={'disp': True, 'maxiter': 50.0})
nn_params = result.x
Jcost = result.fun
```

3 Unsupervised learning

Fund hidden pattern in unlabeled data with little or no idea what our results should look like.
Don't necessarily know the effect of the variables.

3.1 K-means

1) Choose the value of K

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

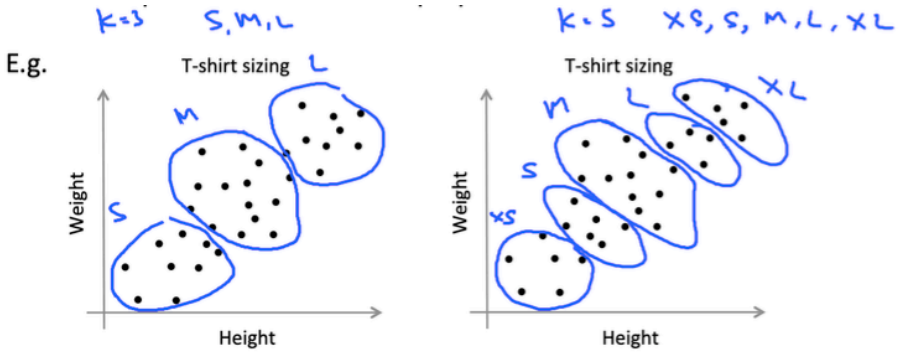


Figure 4 T-shirt size

2) Initialize centroids

Random initialize K clustering centroids $\mu_1, \mu_2, \ldots, \mu_K \in \mathbb{R}^n$

For i = 1 to 100{

Randomly pick examples from given points as K ($K < m$) different centroids

```
initial_centroids = random.sample(X.tolist(),  K)
```

cost function

$$J_{(c,\mu)} = \sum_{i}^{m} \left\| x^{(i)} - \mu_{c^{(i)}} \right\|^2$$

Return centroids of the smallest J.

}

3) Iteration
Set Iterate times, max_iters
For iters = 1 to max_iters {

$$c^{(i)} = index\ of\ min\left\|x^{(i)} - \mu_j\right\|^2$$

$c^{(i)} \in \mathbb{R}^K$, i $= 1,2,\ldots,$ m denotes the index of cluster centroids closet to $x^{(i)}$.

$$\mu_k = \frac{\sum_{i=1,\ \{c^{(i)}=k\}}^{m} x^{(i)}}{\sum_{i=1,\ \{c^{(i)}=k\}}^{m} 1}$$

$\mu_k \in \mathbb{R}^K, k = 1,2,\ldots,K$ denotes the average(mean) of points assigned to cluster k.
}

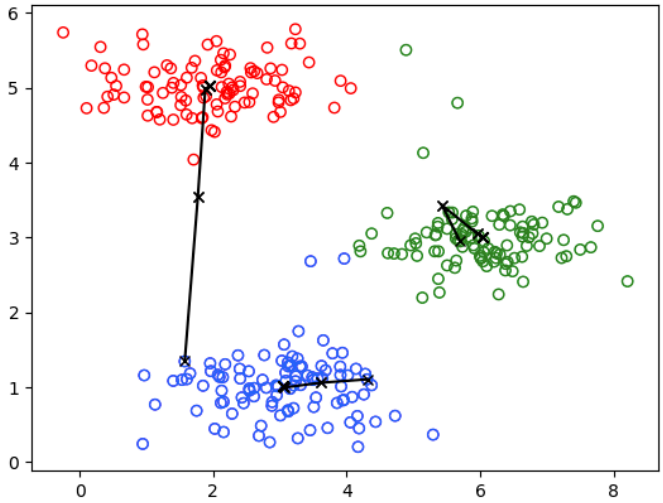


Figure 5 K clusters

3.2 Principal component analysis (PCA)

Dimensionality Reduction
Motivation:

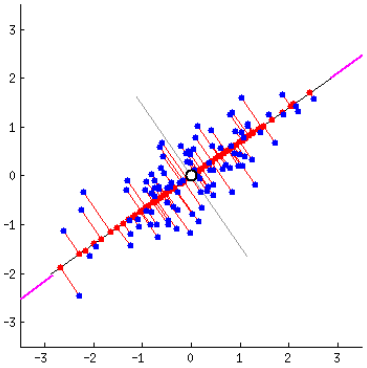


Figure 6 Reduce 2 dimensions to 1 dimension

1) Data processing

Training set: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}, x^{(i)} \in \mathbb{R}^n$
Processing: feature scaling (mean normalization) to ensure every feature has zero mean.

i. Mean value:

$$\bar{X} = \mu_j = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

ii. Standard deviation

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

iii. Replace each $x^{(i)}$

$$x^{(i)} = \frac{x^{(i)} - \mu}{\sigma}$$

2) PCA algorithm.
Reduce data from n-dimensions to k-dimensions

$$sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T = \frac{1}{m}X^T X \qquad U = \begin{pmatrix} | & | & | & | & | & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} & \cdots & u^{(n)} \\ | & | & | & | & | & | \end{pmatrix}$$

Principal Component Analysis (PCA) algorithm:

```
U, S, V = numpy.linalg.svd(sigma)
```

Dimension matrix:

```
Ureduce = U[:, 0:K].T
```

Reduced K-dimensions data:

```
Z = Ureduce * X = X_norm * U[:, 0:K]
```

Recover data to n-dimentions:

```
X_approximate = X_recovered = Z * U[:, 0:K].T
```

3) Choosing K (number of principal components)**:**
Pick the smallest value of K.

$$S = \begin{pmatrix} S_{11} & & \cdots & & 0 \\ & S_{22} & \cdots & 0 & \\ \vdots & \vdots & S_{33} & \vdots & \vdots \\ & 0 & \cdots & \ddots & \\ 0 & & \cdots & & S_{nn} \end{pmatrix}$$

Check if

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)} - x_{approx}^{(i)}\right\|^2}{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}\right\|^2} \leq 0.01? \qquad 1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \leq 0.01 \implies \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$$

99% of variance retained.

PCA is sometimes used where it shouldn't be

Design of ML system:

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

How about doing the whole thing without using PCA?

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

3.3 Anomaly detection

3.3.1 Original model

1) Choose feature:

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

Density estimation: $\boldsymbol{x_j \sim N(\mu_j, \sigma_j^2), j = 1, 2, \ldots, n}$

Choose features x_i that might be indicative of anomalous examples.

2) Fit parameters:

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

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

3) Given new example $\boldsymbol{x}$, compute $\boldsymbol{p(x)}$

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

Flag an anomaly if $\boldsymbol{p(x) < \epsilon}$

$$\boldsymbol{y = \begin{cases} 1, & if\ p(x) < \epsilon (anomaly) \\ 0, & if\ p(x) \geq \epsilon (normal) \end{cases}}$$

3.3.2 Multivariate Gaussian

Don't model $\boldsymbol{p(x_1), p(x_2)}$, ..., etc. separately. Model all $\boldsymbol{p(x)}$ in one go.

1) Choose feature

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

Density estimation: $\boldsymbol{x_j \sim N(\mu_j, \sigma_j^2), j = 1, 2, \ldots, n}$

2) Fit parameters

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

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

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

3) Given new example $\boldsymbol{x}$, compute $\boldsymbol{p(x)}$

$$p(x_j;\mu_j,\Sigma)=\frac{1}{\sqrt{2\pi}|\Sigma|^{\frac{1}{2}}}e^{\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-u)\right)}$$

$$\Sigma=\begin{pmatrix}\Sigma_1 & \cdots & & 0\\ & \Sigma_2 & \ldots & 0 & \\ \vdots & \vdots & \Sigma_3 & \vdots & \vdots\\ & 0 & \ldots & \ddots & \\ 0 & & \cdots & & \Sigma_n\end{pmatrix}$$

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

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

- The importance of real-number evaluation

i. When developing a learning algorithm (choosing features, etc.), making decisions is much easier if we have a way of evaluating our learning algorithm.
Assume we have some labeled data, of anomalous and non-anomalous examples. ($y=0$ if normal, $y=1$ if anomalous).
Training set 60%: $x^{(1)},x^{(2)},\ldots,x^{(m)}$ (assume normal examples/not anomalous)
Cross validation set 20%:$\left(x_{cv}^{(1)},y_{cv}^{(1)}\right),\ldots,\left(x_{cv}^{(m_{cv})},y_{cv}^{(m_{cv})}\right)$
Test set 20%: $\left(x_{test}^{(1)},y_{test}^{(1)}\right),\ldots,\left(x_{test}^{(m_{cv})},y_{test}^{(m_{cv})}\right)$

ii. Algorithm evaluation
Possible evaluation metrics
Precision

$$Precision(P)=\frac{TP}{TP+FP}$$

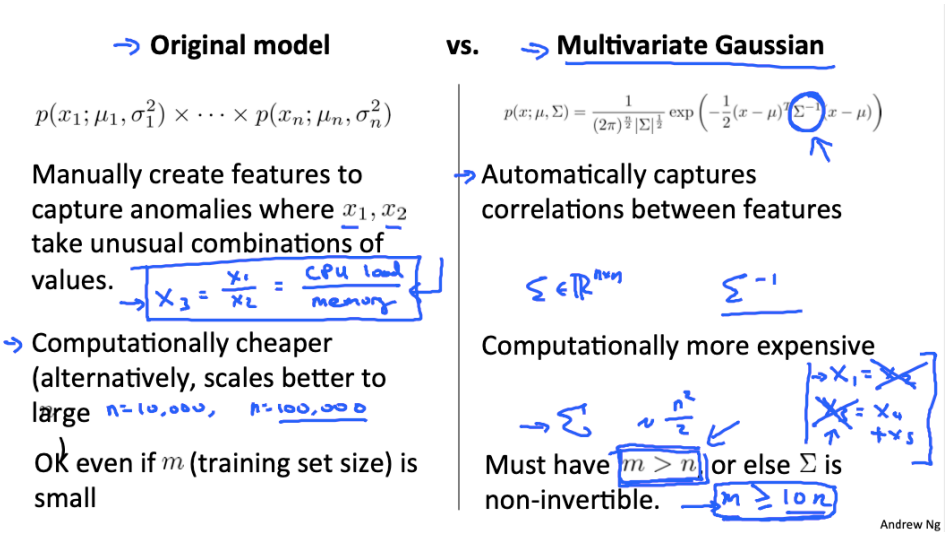
Recall

$$Recall(R)=\frac{TP}{TP+FN}$$

F_1-Score

$$F_1Score=2\frac{PR}{P+R}$$

Comparison between these two models



3.3.3 Collaborative filtering

Recommender system

1) Problem formulation

$r(i,j) = 1$ if user j has rated movie (0 otherwise)

$y^{(i,j)}$ = rating by user j on movie i (if defined)

$\theta^{(j)}$= parameter vector for user j

$x^{(i)}$= feature vector for movie i

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

$m^{(j)}$= no. of movies rated by user j

2) Initialization

Initialize $\left\{x^{(1)}, \dots, x^{(n_m)}\right\}$ and $\left\{\theta^{(1)}, \dots, \theta^{(n_u)}\right\}$to small random values.

3) Cost function

i. Minimize $\left\{x^{(1)}, \dots, x^{(n_m)}\right\}$ and $\left\{\theta^{(1)}, \dots, \theta^{(n_u)}\right\}$

Min $J\left(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}\right)$

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

ii. using gradient descent (or an advanced optimization algorithm). E.g. for every $j = 1,2,\dots,n_u;\ i = 1,2,\dots,n_m$:

Gradients

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

4) Prediction.

For a user with parameters $\theta^{(j)}$ and a movie with(learned) features $x^{(i)}$, predict a star rating of

$$\left(\theta^{(j)}\right)^T x^{(i)} + \mu_i$$

$\mu_i$is the mean value of rated users for movie $x^{(i)}$.

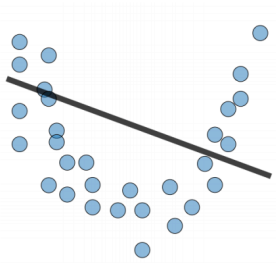
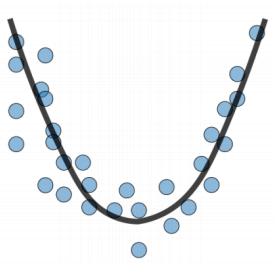
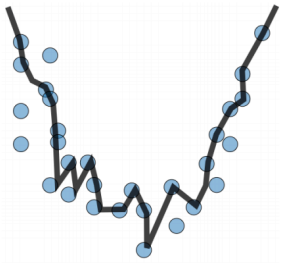
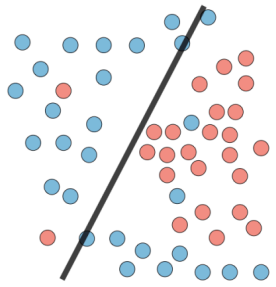
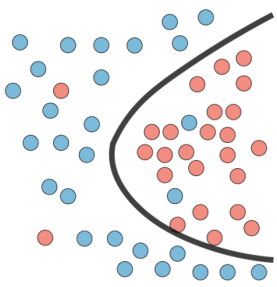
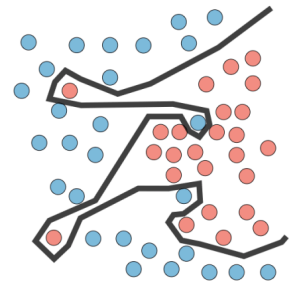
$$\mu_i = mean\left(\sum_{j:r(i,j)=1}^{n} x_j^{(i)} \right)$$

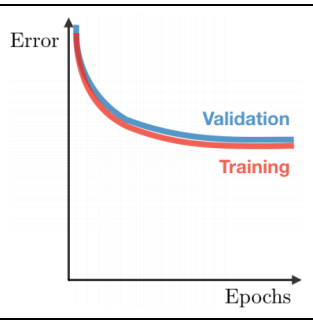
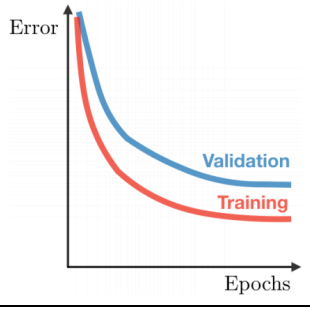
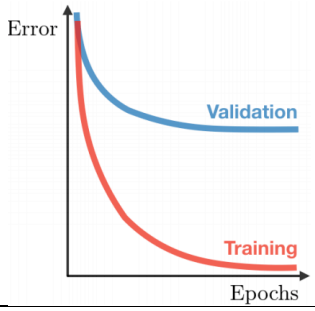
Anomaly detection	vs.	Supervised learning
Very small number of positive examples ($y = 1$). (0-20 is common).		Large number of positive and negative examples.
Large number of negative ($y = 0$) examples. $p(x)$		
Many different "types" of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like;		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 training set.
future anomalies may look nothing like any of the anomalous examples we've seen so far.		Spam

4 Tips

4.1 Diagnostics

- **Bias** – The bias of a model is the difference between the expected prediction and the correct model that we try to predict for given data points.
- **Variance** – The variance of a model is the variability of the model prediction for given data points.
- **Bias/variance tradeoff** – The simpler the model, the higher the bias, and the more complex the model, the higher the variance.

	Underfitting	Just right	Overfitting
Symptoms	- High training error - Training error close to test error - High bias	- Training error - Slightly lower than test error	- Low training error - Training error much lower than test error - High variance
Regression			
Classification			

Deep Learning (Neural network)			
Remedies	- Complexify model - Add more features - Train longer		- Regularize Remedies - Get more data

4.2 Model Selection and Train/Validation/Test Sets

One way to break down our dataset into the three sets is:

i. Training set: 60%
ii. Cross validation set: 20%
iii. Test set: 20%

We can now calculate three separate error values for the three different sets using the following method:

i. Optimize the parameters in Θ using the training set for each polynomial degree.
ii. Find the polynomial degree d with the least error using the cross-validation set.
iii. Estimate the generalization error using the test set with Jtest(Θ(d)), (d = theta from polynomial with lower error);

This way, the degree of the polynomial d has not been trained using the test set.

4.3 Gaussian (Normal) distribution

$$X \sim N(\mu, \sigma^2)$$

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

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

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

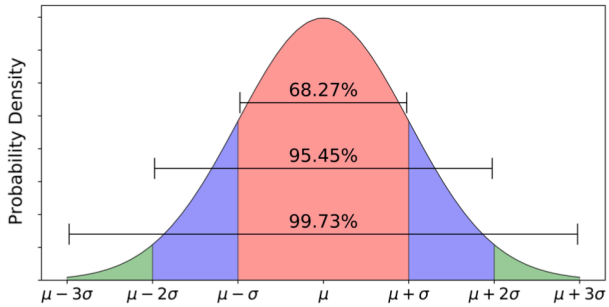


Figure 7 68% of the data is within 1 standard deviation, 95% is within 2 standard deviation, 99.7% is within 3 standard deviations

4.4 F_1-score

F-score is a measure of a test's accuracy.

		Actual class: 1/+	Actual class: 0/-
Predicted class	1/+	True positive (TP)	False positive (FP)
	0/-	False negative (FN)	True negative (TN)

Precision

$$\boldsymbol{Precision(P) = \frac{TP}{\#Predicted\ P} = \frac{TP}{TP + FP}}$$

Recall

$$\boldsymbol{Recall(R) = \frac{TP}{\#Actual\ P} = \frac{TP}{TP + FN}}$$

F_1-score

$$\boldsymbol{F_1 Score = 2\frac{PR}{P + R}}$$