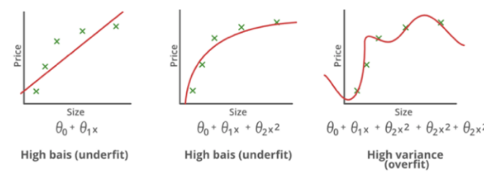
# Supervised Learning

## Linear Regression: *Trend, Market estimates, Forecasts*

**Step 1. Hypothesis:**

$$h_\theta(x)$$

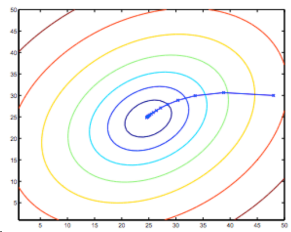



**Step 2. Cost**

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

**Step 3: Gradients**

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



## Logistic Regression: *Binary classes*

**Step 1. Hypothesis:**

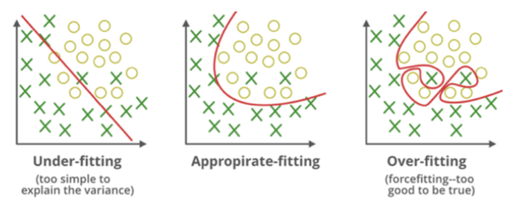
$$\begin{cases} h_\theta(z) = g(\theta^T x) \\ z = \theta^T x \\ g(z) = \dfrac{1}{1 + e^{-z}} \end{cases}$$

**Step 2. Cost**

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

**Step 3. Gradients:**

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

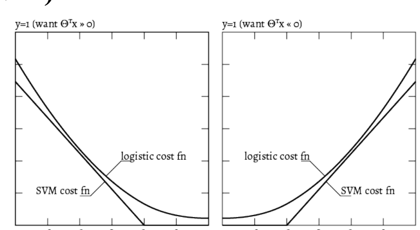



## Support Vector Machines (SVM)

**Cost**

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

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




## SVM with Gaussian Kernel

**Step 1. Hypothesis**

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

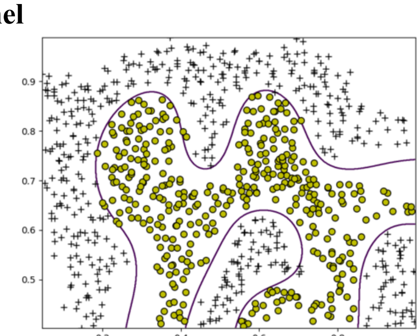
*Predict "y=1" if* $\theta^T f \geq 0, \ \theta_0 f_0 + \theta_1 f_1 + \cdots + \theta_m f_m \geq 0$

**Step 2. Training**

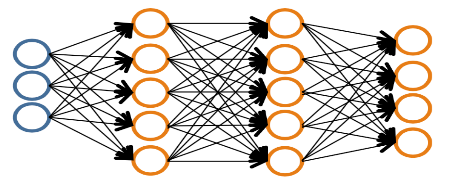
$$\min J(\theta) = C\textstyle\sum_{i=1}^{m}[y^{(i)}\,cost_1(\theta^T f_i) + (1 - y^{(i)})\,cost_0(\theta^T f_i)] + \frac{1}{2}\sum_{j=1}^{m}\theta_j^2$$

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

*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**

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

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

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

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

**Step 4. Backpropagation to compute partial derivatives**

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

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

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

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

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

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

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

**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 $\Theta$ .

```
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: *Clustering*

**Step 1. Centroids**

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

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

**Step 2. Means**

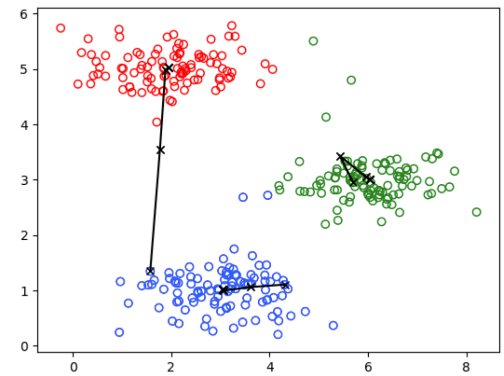
$$\mu_k = \frac{\sum_{i=1,\ \{c^{(i)}=k\}}^{m} x^{(i)}}{\sum_{i=1,\ \{c^{(i)}=k\}}^{m} 1}$$

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

**Step 3. Cost function**

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

K=3



## Principal Component Analysis (PCA):

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

**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.**

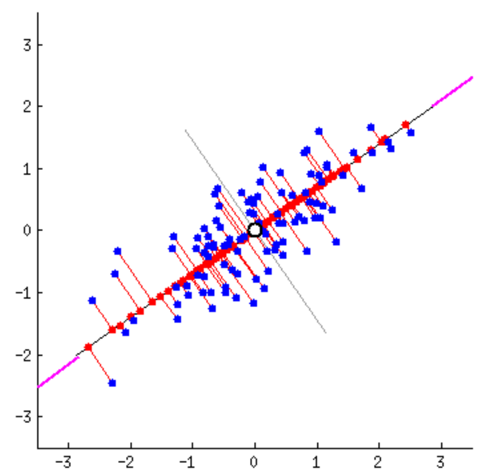
$\boldsymbol{sigma} = \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{x^{(i)}})(\boldsymbol{x^{(i)}})^T = \frac{1}{m}\boldsymbol{X^T X}$

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

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

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

$2D \rightarrow 1D$



**Step 3. Pick the smallest value of k,**

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

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

$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \le 0.01 \quad \rightarrow \quad \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \ge 0.99$$

99% of variance retained

## **Anomaly Detection:** *Fraud detection, Intrusion detection, system health, monitoring*

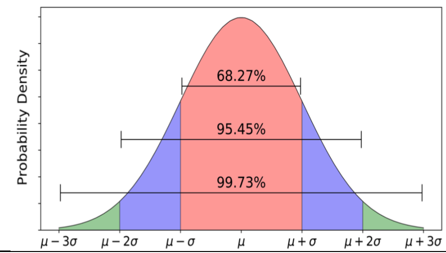
**Gaussian (Normal) distribution**

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

**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**

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

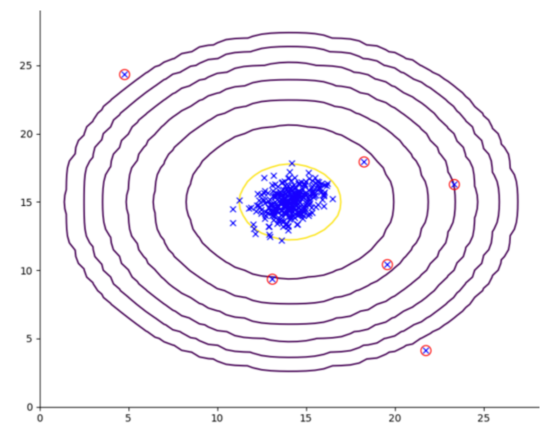
$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$$

**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 $p(x)$**

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

Probability:

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

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