Naive Bayes

March 20, 2013

Spam Classification

- Training set - set of emails labeled spam/non-spam
- Each feature vector
  - has length N of number of words in a dictionary (e.g. $N = 50000$)
- if an email contains i-th word in a dictionary then
  - set $x_i = 1$
  - otherwise set $x_i = 0$

Bayes Law and Naive Assumption

$$P(y|x) = \frac{P(x,y)}{P(x)} = \frac{P(x|y)P(y)}{P(x)}$$

- y - class label
- $x = (x_1, x_2, ..., x_N)$ - feature vector
- to estimate $p(x|y)$ we assume that x_i's are independent given y

$$p(x_1, ..., x_{50000}|y) = p(x_1|y)p(x_2|y, x_1)...p(x_{50000}|y, x_1, ..., x_{49999})$$

$$= p(x_1|y)p(x_2|y)p(x_3|y)...p(x_{50000}|y) = \prod_{i=1}^{n} p(x_i|y)$$

Classificator

$$P(y|x_1, ..., x_{50000}) = \frac{p(x_1|y)...p(x_{50000}|y)}{p(x_1, ..., x_{50000})} p(y)$$

- Estimation of multidimensional distribution $p(x_1, ..., x_{50000})$ might be problematic
- Class label assigned for

$$\text{argmax}_y p(y|x) = \text{argmax}_y \left(\frac{p(x|y)p(y)}{p(x)} \right) = \text{argmax}_y p(x|y)p(y)$$

Laplace Smoothing

- Let imagine that classificator met some new word in email, that doesn't contained in the dictionary. Let it's index be $i = 50001$ then

$$P_(x_{50001}|y=0) = \frac{\text{number of items with } x_{50001} = 1 \text{ and } y = 0}{\text{number of items with } y = 0} = 0,$$

$$P_(x_{50001}|y=1) = \frac{\text{number of items with } x_{50001} = 1 \text{ and } y = 1}{\text{number of items with } y = 1} = 0.$$

  and resulting score

$$p(x_1|y=0) \ldots p(x_{50001}|y=0)p(y=0) = 0$$

$$p(x_1|y=1) \ldots p(x_{50001}|y=1)p(y=1) = 0$$

- That seems to be not good since, some other words might be used for correct classification.
- In this case let's artificially add 1 to nominator and 2 for denominator

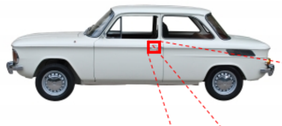
$$P_(x_{50001}|y=0) = \frac{\text{number of items with } x_{50001} = 1 \text{ and } y = 0 + 1}{\text{number of items with } y = 0 + 2},$$

$$P_(x_{50001}|y=1) = \frac{\text{number of items with } x_{50001} = 1 \text{ and } y = 1 + 1}{\text{number of items with } y = 1 + 2}.$$

Neural Networks

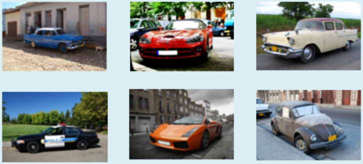
March 20, 2013

You see this:

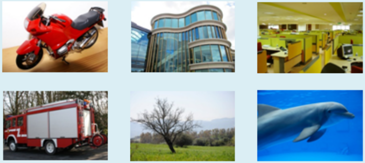


But the camera sees this:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50



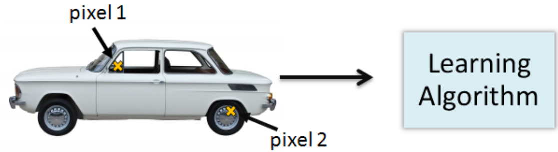
Cars

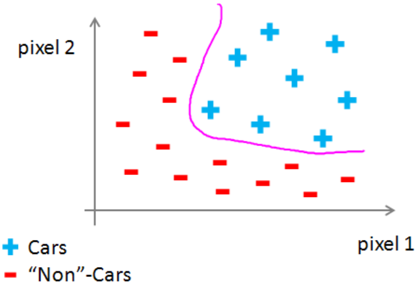


Not a car

Testing: 

What is this?



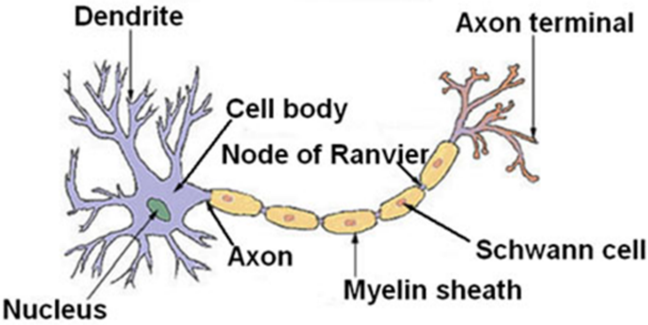


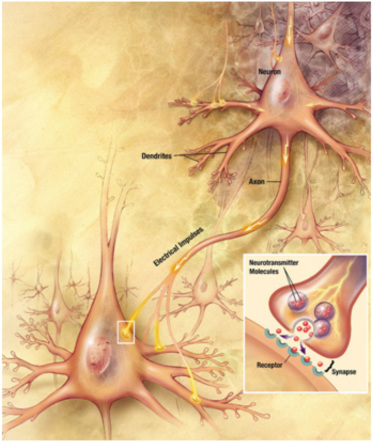
50 x 50 pixel images→ 2500 pixels

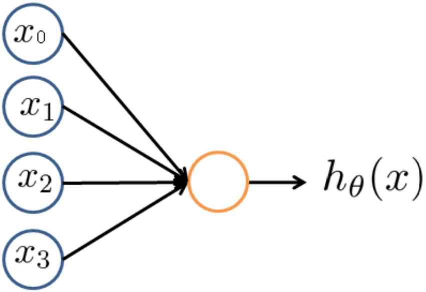
$n = 2500$ (7500 if RGB)

$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix}$$

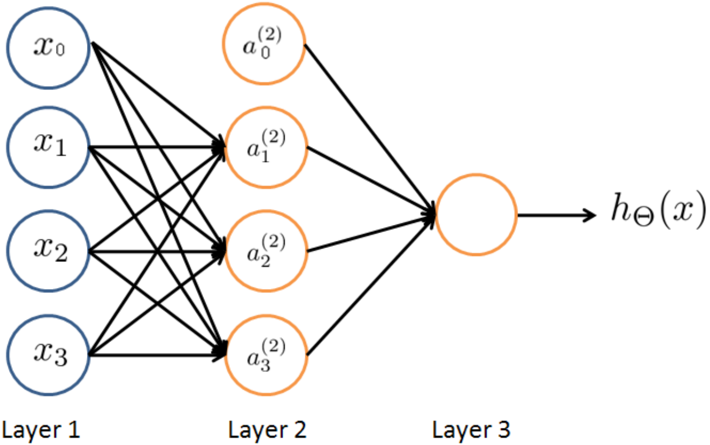
Quadratic features ($x_i \times x_j$): ≈3 million features

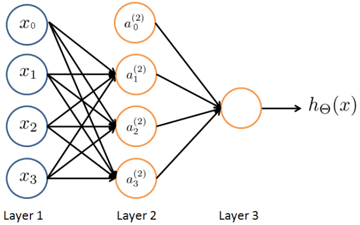

Dendrite
Axon terminal
Cell body
Node of Ranvier
Axon
Schwann cell
Myelin sheath
Nucleus


Neuron
Dendrites
Axon
Electrical impulses
Neurotransmitter Molecules
Receptor
Synapse



$$h_\theta(x) = \frac{1}{1 + exp\left\{-\theta x\right\}}$$


x_0
x_1
x_2
x_3
$a_0^{(2)}$
$a_1^{(2)}$
$a_2^{(2)}$
$a_3^{(2)}$
$h_\Theta(x)$
Layer 1
Layer 2
Layer 3



- $a_i^{(l)}$ - "activation" of unit i in layer l
- Θ^l - matrix of weights controlling function mapping from layer l to layer $l+1$
- Each $\Theta_{i,j}^{(l)}$ is edje from $a_j^{(l)}$ to $a_i^{(l+1)}$

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

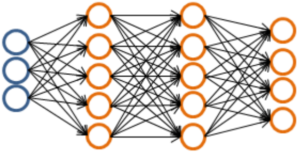
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

$$h_\theta(x) = g(\Theta_{10}^{(2)}a_0 + \Theta_{11}^{(2)}a_1 + \Theta_{12}^{(2)}a_2 + \Theta_{13}^{(2)}a_3)$$

If network has s_l units in layer l, s_{l+1} units in layer $l+1$, then $\Theta^{(l)}$ will be of dimension $s_{l+1} \times (s_l + 1)$

Forward Propagation



$$
\begin{aligned}
a^{(1)} &= x \\
z^{(2)} &= \Theta^{(1)} a^{(1)} \\
a^{(2)} &= g(z^{(2)}) \quad (\text{add} \quad a_0^{(2)}) \\
z^{(3)} &= \Theta^{(2)} a^{(2)} \\
a^{(3)} &= g(z^{(3)}) \quad (\text{add} \quad a_0^{(3)}) \\
z^{(4)} &= \Theta^{(3)} a^{(3)} \\
a^{(4)} &= h_\Theta(x) = g(z^{(4)})
\end{aligned}
$$

Example

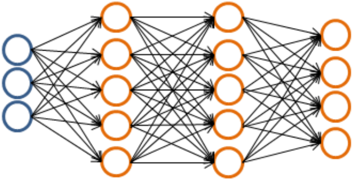
Multiple output units: One-vs-all

Pedestrian

Car

Motorcycle

Truck



$$h_\Theta(x) \in \mathbb{R}^4$$

Want $h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc.

when pedestrian when car when motorcycle

Cost Function and Gradient Computation

- Cost function

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=0}^{m}\sum_{k=1}^{K} y_k^{(i)} \log h_\Theta(x^{(i)})_k + (1 - y_k^{(i)}) \log(1 - h_\Theta(x^{(i)})_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}\left(\Theta_{j,i}^{(l)}\right)^2$$

- For minimization of $J(\Theta)$ we have to estimate

$$\frac{\partial J(\Theta)}{\partial \Theta_{i,j}^{(l)}}$$

and use gradient method for estimation of

$$\Theta_{i,j}^{(1)}, \Theta_{i,j}^{(2)}, \ldots, \Theta_{i,j}^{(L)}$$

Gradient computation: Backpropagation algorithm

- Intuition: $\delta_i^{(l)}$ = "error" of node i in layer l
- For each output layer estimate

$$\frac{\partial J(\Theta)}{\partial \Theta_{i,j}^{(l)}} = a_j^{(l)} \delta_i^{(l+1)} \quad (1)$$

$$\delta_i^{(L)} = a_i^{(L)} - y_i \quad (2)$$

$$\delta_i^{(l)} = g(z_i^l) g(1 - z_i^l) \sum_k \delta_k^{l+1} \Theta_{k,i}^l, \quad \text{where} \quad l = 2..L-1 \quad (3)$$

Intuition (1)

$$\frac{\partial J(\Theta)}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial J(\Theta)}{\partial z_i^{(l)}} \frac{\partial z_i^{(l)}}{\partial \Theta_{i,j}^{(l)}}$$

Let denote first factor as $\delta_i^{(l+1)} = \frac{\partial J(\Theta)}{\partial z_i^{(l)}}$

The second factor $\frac{\partial z_i^{(l)}}{\partial \Theta_{i,j}^{(l)}} = a_j^{(l)}$

$$\frac{\partial J(\Theta)}{\partial \Theta_{i,j}^{(l)}} = a_j^{(l)} \delta_i^{(l+1)}$$

Intuition (3)

$$\delta_i^{(l+1)} = \frac{\partial J(\Theta)}{\partial z_i^{(l)}} = \sum_k \frac{\partial J(\Theta)}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}}$$

$$z_k^{(l+1)} = \sum_j \Theta_{k,j}^{(l)} a_j^{(l)}, \quad \text{where} \quad a_k^{(l)} = g(z_j^{(l)})$$

Right factor in sum is $\frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}} = \Theta_{k,i}^{(l)} g'(z_i^{(l)})$

Left factor simply $\frac{\partial J(\Theta)}{\partial z_k^{(l+1)}} = \delta_k^{(l+2)}$

Finally we get

$$\delta_i^{(l+1)} = \sum_k \delta_k^{(l+2)} \Theta_{k,i}^{(l)} g'(z_i^{(l)})$$

Backpropagation Algorithm

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

Set $\triangle^{(l)} = 0$ (for all l).

For $i = 1$ to m

Set $a^{(1)} = x^{(i)}$

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

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

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\triangle^{(l)} := \triangle^{(l)} + \delta^{(l+1)} a^{(l)\top}$

$$D^{(l)} := \frac{1}{m} \triangle^{(l)} + \lambda \Theta^{(l)} \text{ if } j \neq 0$$

$$D^{(l)} := \frac{1}{m} \triangle^{(l)} \qquad \text{if } j = 0$$

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

Numerical Estimate of Gradient

$\theta \in \mathbb{R}^n$ (E.g. θ is "unrolled" version of $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$)

$\theta = \theta_1, \theta_2, \theta_3, \ldots, \theta_n$

$$\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1+\epsilon, \theta_2, \theta_3, \ldots, \theta_n) - J(\theta_1-\epsilon, \theta_2, \theta_3, \ldots, \theta_n)}{2\epsilon}$$

$$\frac{\partial}{\partial \theta_2} J(\theta) \approx \frac{J(\theta_1, \theta_2+\epsilon, \theta_3, \ldots, \theta_n) - J(\theta_1, \theta_2-\epsilon, \theta_3, \ldots, \theta_n)}{2\epsilon}$$

$$\vdots$$

$$\frac{\partial}{\partial \theta_n} J(\theta) \approx \frac{J(\theta_1, \theta_2, \theta_3, \ldots, \theta_n+\epsilon) - J(\theta_1, \theta_2, \theta_3, \ldots, \theta_n-\epsilon)}{2\epsilon}$$

Training a neural network

- Randomly initialize weights
- Implement forward propagation to get $h_\Theta(x)$
- Implement code to compute cost function $J(\Theta)$
- Implement backprop to compute partial derivatives $\frac{\partial J(\Theta)}{\partial \Theta^{(l)}_{i,j}}$
- for i in 1:m
  - Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$ (Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2..L$).
- Use gradient checking to compare $\frac{\partial J(\Theta)}{\partial \Theta^{(l)}_{i,j}}$ computed using backpropagation vs. using numerical estimate of gradient of $J(\Theta)$.
- Use gradient descent or advanced optimization method (BFGS ...) with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ