Linear Regression

Simply put we have data and we try to learn a line which passes through it in best fit possible.

Supervised Learning:-

Testing set

↓ Input

Learning Algo

↓ Output

→ function h (hypothesis) → Predicts Price of house

IP- size of house

→ When designing a learning algo we need to ask how do you represent h?

In Linear regression:-

$$h(x) = \theta_0 + \theta_1 x$$

We input size x and it outputs a function which is a linear combination of size x.

More generally:-

$x_1 = \text{size}$, $x_2 = \text{No. of bedrooms}$

$$h(x_1, x_2) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

In other words:-

$$h(x) = \sum_{j=0}^{2} \theta_j x_j$$

where $x_0 = 1$

Here,

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} \qquad x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

$\theta \rightarrow$ Parameters

m = # Training samples

x = "Inputs" / feature

y = "output" / Target Variable

$(x, y) \rightarrow$ one training example

$(x^i, y^i) \rightarrow$ ith training example

$n \rightarrow$ # features

→ Choose θ such that $h(x) \approx y$ for training examples.

$h_\theta(x)$ depicts that h depends both on params and input features x

In Linear Regression algos aim is to minimise the least squares error:-

$$\text{Minimise}\left(h_\theta(x) - y\right)^2$$

Basically we need to choose values of θ that minimises least squares error

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

we add $\frac{1}{2}$ to make math simpler.

$$\therefore \min_{\theta} \left(J(\theta)\right)$$

*** Gradient Decent:-**

① Start with some θ (say $\theta = \vec{0}$)

Keep changing θ to minimise $J(\theta)$

One step of gradiant decent is implemented as follows:–

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Learning Rate

:= depicts assignment

Some math:–

$$\boxed{\frac{\partial J(\theta)}{\partial \theta_j}} = \frac{\partial}{\partial \theta_j}\left(\frac{1}{2}(h_\theta(x) - y)^2\right)$$

$$= 2 \times \frac{1}{2} \frac{\partial}{\partial \theta_j}(h_\theta(x) - y)$$

$$= (h_\theta(x) - y) \frac{\partial}{\partial \theta_j}(\theta_0 x_0 + \theta_1 x_1 + \dots + \theta_n x_n - y)$$

$$= \boxed{(h_\theta(x) - y) x_j}$$

Hence one step of gradient decent is the following:–

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

∴ The gradiant decent algorithm is to repeat eq (1) until convergance for $j = \{1, 2, \dots, n\}$

where n is the no. of features.

We find $\frac{\partial}{\partial \theta_j} J(\theta)$ by summing RHS over m — That's why it is also called Batch Gradient decent.

Efficient algo than this is stocastic gradient decent.

Repeat {

for every j

for $j = 1$ to m {

$$\theta_j := \theta_j - \alpha (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

}

}

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \\ \frac{\partial J}{\partial \theta_2} \end{pmatrix}$$

Hence shorter way of computing global minima which is computed in gradient decent is by taken derivative of $J(\theta)$ wrt θ and equating it to $\vec{0}$.

J is a function which maps vector θ to a real value.

$$\nabla_\theta J(\theta) = \vec{0}$$

→ If A is $n \times n$ matrix $\text{tr} A$ = Sum of primary diagonal elements

$$\text{tr} A = \text{tr} A^T$$

If $f(A) = \text{tr} AB$

Then

$$\nabla_A f(A) = B^T$$

→ $\text{tr} AB = \text{tr} BA$

→ $\text{tr} ABC = \text{tr} CAB$

$$\nabla_A \text{tr} A A^T C = CA + C^T A$$

Now,

we know that,

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(h(x^{(i)}) - y^{(i)}\right)^2$$

$$X\theta = \begin{bmatrix} —(x^1)^T— \\ —(x^2)^T— \\ \vdots \\ —(x^m)^T— \end{bmatrix} \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{pmatrix} = \begin{bmatrix} x^{1^T}\theta \\ x^{2^T}\theta \\ \vdots \\ x^{m^T}\theta \end{bmatrix}$$

$$= \begin{bmatrix} h_\theta(x^1) \\ \vdots \\ h_\theta(x^m) \end{bmatrix}$$

Also,

$$\vec{y} = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^m \end{bmatrix}$$

Then,

$$J(\theta) = \frac{1}{2}(X\theta - y)^T(X\theta - y)$$

Hence,

$$\nabla_\theta J(\theta) = \nabla_\theta \frac{1}{2}(X\theta - y)^T(X\theta - y)$$

$$= \frac{1}{2}\nabla_\theta(\theta^T X^T - y^T)(X\theta - y)$$

$$= \frac{1}{2}\nabla_\theta\left[\theta^T X^T X\theta - \theta^T X^T y - y^T X\theta + y^T y\right]$$

$$= \frac{1}{2}\left[X^T X\theta + X^T X\theta - X^T y - X^T y\right]$$

$$= X^T X\theta - X^T y \overset{\text{set}}{=} \vec{0}$$

$$\Rightarrow X^T X\theta = X^T y$$ "Normal eq"

and,

$$\theta = (X^T X)^{-1} X^T y \quad -②$$

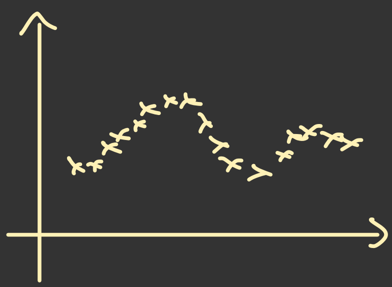
Hence, if we compute ② then we get the value of θ that corresponds to global minimum in one single step.

Lecture - 3

Locally weighted Regression :-

Note that - Parametric learning algos - Fit fixed set of parameters (θ_i) to data.

Locally weighted regression is a 'non parametric' learning algo as data / params you need to keep grows linearly with the size of data.



To evaluate h at certain x:

Linear Reg: Fit θ to minimise

$$\frac{1}{2}\sum_i (y^{(i)} - \theta^T x^{(i)})^2$$

Return $\theta^T x$

In locally weighted regression:-

Fit θ to minimise

$$\sum_{i=1}^{m} w^{(i)} (y^{(i)} - \theta^T x^{(i)})^2$$

where $w^{(i)}$ is a weight function

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2}\right)$$

If $|x^{(i)} - x|$ is small, $w^{(i)} \approx 1$

$x \to$ location where you want to make a prediction

$x^{(i)} \to$ Input x for your ith training example.

If $|x^{(i)} - x|$ is large, then $w^{(i)} \approx 0$

$\to$ We use a hyperparameter τ which is the bandwidth and is used to define the width of the neighbourhood we are using.
Say

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

* Probabilistic interpretation of linear regression :-

Why least squares?

Assume $y^{(i)} = \theta^T x^{(i)} + \varepsilon^{(i)}$
$\hookrightarrow$ error

$$\varepsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$$

$$P(\varepsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{(\varepsilon^{(i)})^2}{2\sigma^2}\right)$$

We make the assumption that the ε error terms are Independently and Identically distributed.

This implies that:-

$$P(y^{(i)} | x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right),$$

Read as - parametrised by

This means that

$$P(y^{(i)} | x^{(i)}; \theta) \sim \mathcal{N}(\theta^T x^{(i)}, \sigma^2)$$

$$\mathcal{L}(\theta) = P(\vec{y}\,|\,x;\theta)$$
$$= \prod_{i=1}^{m} P(y^{(i)}\,|\,x^{(i)};\theta)$$
$$= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{(y^{(i)}-\theta^T x^{(i)})^2}{2\sigma^2}\right)$$

$\mathcal{L}(\theta) \rightarrow$ Likelihood of θ

Log likelihood :-

$$\ell(\theta) = \log \mathcal{L}(\theta)$$
$$= \log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp(---)$$
$$= \sum_{i=1}^{m} \left[\log \frac{1}{\sqrt{2\pi}\sigma} + \log \exp(---)\right]$$
$$= m \log \frac{1}{\sqrt{2\pi}\sigma} + \sum_{i=1}^{m} -\frac{(y^i - \theta^T x^i)^2}{2\sigma^2}$$

Maximum likelyhood Estimation:-

(MLE)

Chose θ to maximise $\mathcal{L}(\theta)$

i.e choose θ to minimise:-

$$\frac{1}{2}\sum_{i=1}^{m} (y^i - \theta^T x^i)^2 = J(\theta)$$

Classification Problem:-

Binary Classification:-

$$y = \{0,1\}$$

Logistic Regression:

Most commonly used classification algorithm

Want $h_\theta(x) \in [0,1]$

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

$$g(z) = \frac{1}{1+e^{-z}} \rightarrow \text{Sigmoid or logistic function}$$

$$P(y=1\,|\,x;\theta) = h_\theta(x)$$
$$P(y=0\,|\,x,\theta) = 1 - h_\theta(x)$$
$$y \in \{0,1\}$$
$$\therefore P(y\,|\,x;\theta) = h(x)^y (1-h(x))^{1-y}$$

So, likelihood function now becomes :-

$$\mathcal{L}(\theta) = P(\vec{y}\,|\,x;\theta)$$
$$= \prod_{i=1}^{m} p(y^i\,|\,x^i;\theta)$$
$$= \prod_{i=1}^{m} h_\theta(x^i)^{y^i} (1-h_\theta(x^i))^{1-y^i}$$

and,

$$\ell(\theta) = \log \mathcal{L}(\theta)$$
$$= \sum_{i=1}^{m} y^i \log h_\theta(x^i) + (1-y^i)\log(1-h_\theta(x^i))$$

Chose θ to maximise $\ell(\theta)$,

Batch gradient Ascent:-

$$\theta_j := \theta_j + \alpha \frac{\partial}{\partial \theta_j} \ell(\theta)$$

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

Newton's Method :-

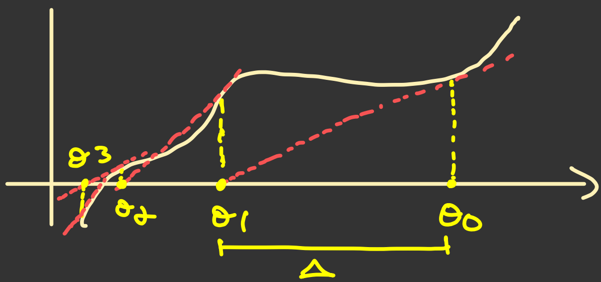
Much faster than gradient ascent for optimising the value of θ.

We have f

We want to find θ such that:-

$f(\theta)=0$

[want to maximise $L(\theta)$
i.e. want $l'(\theta)=0$]



$$\theta_1=\theta_0-\Delta$$

$$f'(\theta_0)=\frac{f(\theta_0)}{\Delta}$$

$$\therefore \Delta=\frac{f(\theta_0)}{f'(\theta_0)}$$

$$\therefore \theta_{t+1}=\theta_t-\frac{f(\theta_t)}{f'(\theta_t)}$$

Now,
Let $f(\theta)=l'(\theta)$

So,

$$\theta_{(t+1)}=\theta_t-\frac{l'(\theta_t)}{l''(\theta_t)}$$

when θ is a vector

$$\theta_{t+1}=\theta_t+H^{-1}\underbrace{\nabla_\theta l}_{\text{vector } \mathbb{R}^{n+1}}$$

where H is Hessian matrix

$H^{-1}\to\mathbb{R}^{n+1\times n+1}$

$$H_{ij}=\frac{\partial^2 l}{\partial\theta_i\partial\theta_j}$$

Lecture - 4

We saw that logistic regression used sigmoid function:-

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

The perceptron function is the hard version of sigmoid:-

$$g(z)=\begin{cases}1 & z\geq 0\\ 0 & z<0\end{cases}$$

$$h_\theta(x)=g(\theta^T x)$$

The gradient ascent is the same but the $h_\theta(x)^i$ changes in perceptron function:-

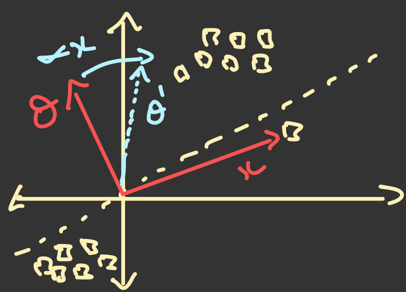
$$\theta_j:=\theta_j+\alpha(y^i-h_\theta(x)^i)x_j^i$$

Note that, $(y^i-h_\theta(x)^i)$ is a scalar. So, the values will be

$0\to$ algo got it right

$\pm 1\to 1$ if wrong $y^i=1$

-1 if wrong $y^i=0$



$\theta\approx x\mid y=1$

$\theta\not\approx x\mid y=0$

Exponential families:-

The one where PDF:-

$$P(y,\eta)=b(y)\exp[\eta^T T(y)-a(\eta)]$$

y - data

η - natural parameter

$T(y)$ - sufficient statistics

$b(y)$ - Base measure

$a(\eta)$ — log-partition

Note that the dimensions of η and $T(y)$ has to match

*) Bernoulli dist (To map binary data)

ϕ = Probability of event

$$P(y;\phi) = \phi^{y}(1-\phi)^{(1-y)}$$

$$= \exp\left[\log\left(\phi^{y}(1-\phi)^{(1-y)}\right)\right]$$

$$= \exp\left[\log\left(\frac{\phi}{1-\phi}\right)y + \log(1-\phi)\right]$$

Here,

$$b(y) = 1$$

$$\eta = \log\left(\frac{\phi}{1-\phi}\right) \Rightarrow \phi = \frac{1}{1+e^{-\eta}}$$

$$T(y) = y$$

$$a(\eta) = -\log(1-\phi)$$

$$= -\log\left(1-\frac{1}{1+e^{-\eta}}\right) = \log(1+e^{\eta})$$

This proves that Bernoulli is a member of exponential family

* Gaussian Dist (with fix variance)

Assume $\sigma^2 = 1$

$$P(y;\mu) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{(y-\mu)^2}{2}\right)$$

$$= \frac{1}{\sqrt{2\pi}}e^{-y^2/2}\exp\left(\mu y - \frac{1}{2}\mu^2\right)$$

So, $b(y) = \frac{1}{\sqrt{2\pi}}\exp\left(\frac{-y^2}{2}\right)$

$T(y) = y$, $\eta = \mu$

$$a(\eta) = \frac{\mu^2}{2} = \frac{\eta^2}{2}$$

This proves that Gaussian is also a member of exponential family.

Properties of Exponential families:-

ⓐ MLE wrt $\eta \Rightarrow$ concave

negative log likelihood $\Rightarrow$ convex

ⓑ $E[y;\eta] = \frac{\partial}{\partial\eta}a(\eta)$

ⓒ $V[y;\eta] = \frac{\partial^2}{\partial\eta^2}a(\eta)$

GLM :-

<Generalized Linear Models>

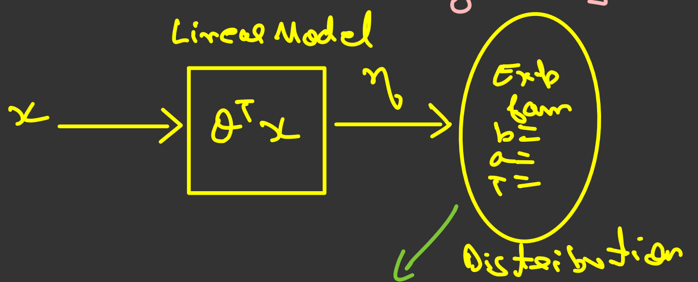
Assumptions / Design Choices :-

(i) $y|x;\theta \sim$ Exponential Family

ii) $\eta = \theta^T x$ $\theta \in \mathbb{R}^n, x \in \mathbb{R}^n$

iii) Test time: Output $E[y|x;\theta]$

$\Rightarrow h_\theta(x) = E[y|x;\theta]$



$$E[y|\eta] = E[y|\theta^T x]$$
$$= h_\theta(x)$$

It is very clear here that we are training θ to predict the params of exponential family dist. whose mean is the prediction we are going to make for y.

All this is during Test time.
During Train time:—

The param we are learning is θ using gradient descent.

During learning we perform the MLE over θ:—

$$\text{Max} \log P(y^{(i)}; \theta^T x^{(i)})$$

GLM Training:—

Learning update Rule:—

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

We can also use Newtons method to learn θ; as long as the dimensionality of our features is less than a few thousand.

Data Type	Dist. used to map
Real	Gaussian
Binary	Bernoulli
Positive Integers	Poisson
R^+	Gamma, Exponential

Terminology:—

η − Natural parameter

$\mu = E[y; \eta] = g(\eta) \rightarrow$ Canonical Response function

$\eta = g^{-1}(\mu) \rightarrow$ Canonical Link function

$$g(\eta) = \frac{\partial a(\eta)}{\partial \eta}$$

We have 3 - parameterizations:—

Model Param	Natural Param	Canonical Param
θ	η	ϕ − Bern
		μ, σ^2 − Gaus
		λ − Poisson

$\uparrow$ — $\theta^T x \longrightarrow$ η — $g \rightarrow$ Canonical Param

Learn — Design choice — $\leftarrow g^{-1}$ —

So, now it becomes evident that, for example, if we use bern to be the dist of our output then the logistic regression becomes:—

$$h_\theta(x) = E[y|x;\theta]$$
$$= \phi$$
$$= \frac{1}{1+e^{-\eta}}$$
$$= \left.\frac{1}{1+e^{-\theta^T x}}\right\} \text{sigmoid func.}$$

Softmax Regression:—

Cross Entropy interpretation

Here we are talking about multi class classification.

Goal − Learn a model which when given a new datapoint can make a prediction to which class the new point belongs to.

K - # classes

$x^{(i)} \in \mathbb{R}^n$

Label $y = [\{0,1\}^K]$ eg $[0,0,1,0]$

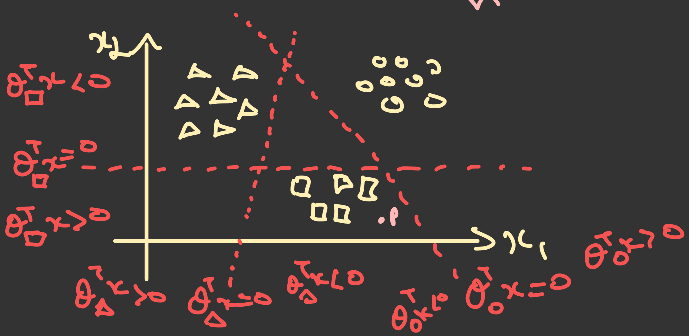
$\theta_{class} \in \mathbb{R}^n$

class $\in \{\Delta, 0, \square \dots\}$

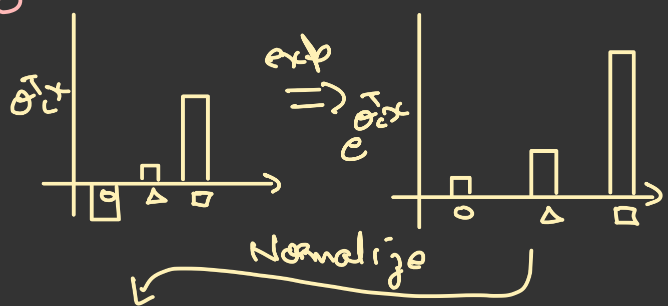
We have K such θs, one for each class

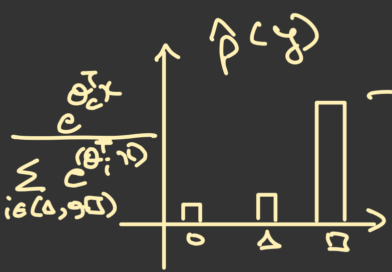
We can also represent it as a matrix :-

$$\begin{array}{cc} & \\ K & \begin{bmatrix} - & \theta_{c_1} & - \\ - & \theta_{c_2} & - \\ & \vdots & \end{bmatrix} \\ & n \end{array}$$

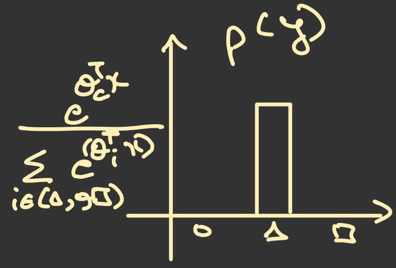


So, for a new point P we might get:-





Let's assume that the new point is Δ. In that case $P(y)$ would look like:-



So, the learning approach we need to do is to minimize the distance between $\hat{P}(y)$ and $P(y)$

Technically the term is to minimize the cross entropy between $\hat{P}(y)$ and $P(y)$

$$\text{CrossEnt}(P, \hat{P}) = \sum_{y \in \{0, \Delta, \square\}} P(y) \log \hat{P}(y)$$

$$= -\log \hat{P}(y_\Delta)$$

$$= -\log \frac{e^{\theta_\Delta^T x}}{\sum_{c \in \{0, \Delta, \square\}} e^{\theta_c^T x}} \quad \text{— (a)}$$

We treat (a) as loss and do gradient decent wrt the parameters.

Lecture - 5

All algos studied so far are called discriminative learning algos.

Now we will look into Discriminative learning algos.

A discriminative learning algo learns $p(y|x)$

or, learns $h_\theta(x) = \{^0_1$ directly

where $h_\theta(x)$ is a mapping from x to y

A generative learning algo learns

$$\underbrace{P(x}_{\text{feature}} | \underbrace{Y)}_{\text{class}}$$

It also learns $p(y)$ which is called class prior.

So by using Bayes rule:-

$$P(y=1|x) = \frac{P(x|y=1)\,P(y=1)}{p(x)}$$

where,

$$p(x) = p(x|y=1)\,p(y=1) + p(x|y=0)\,p(y=0)$$

*Gaussian Discriminant Analysis:-

Suppose $x \in \mathbb{R}^n$ (drop $x_0 = 1$ conversion)

Assume $p(x|y)$ is Gaussian

$$z \sim \mathcal{N}(\underbrace{\vec{\mu}}_{\mathbb{R}^n}, \underbrace{\Sigma}_{\mathbb{R}^{n\times n}}) \qquad z \in \mathbb{R}^n$$

$$E[z] = \mu$$

$$\mathrm{Cov}(z) = E\left[(z-\mu)(z-\mu)^T\right]$$
$$= Ezz^T - (Ez)(Ez)^T$$

I am writing $E[z]$ as Ez for easier writing

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

GDA Model:

$$p(x|y=0) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_0)^T \Sigma^{-1}(x-\mu_0)\right)$$

$$p(x|y=1) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_1)^T \Sigma^{-1}(x-\mu_1)\right)$$

Parameters are - $\underbrace{\mu_0, \mu_1}_{\mathbb{R}^n}, \underbrace{\Sigma}_{\mathbb{R}^{n\times n}}, \phi - \mathbb{R}$

Note that we are using same cov matrix Σ for both classes but diff μ - μ_1 and μ_0

$$p(y) = \phi^y (1-\phi)^{1-y}$$

y is a bern random variable as it takes values 0 and 1. That's why we have used bern dist for $p(y)$

* How to fit the parameters?

Training set - $\{x^i, y^i\}_{i=1}^m$

In Generative algos, we define Joint likelihood here,

$$\mathcal{L}(\phi, \mu_1, \mu_2, \Sigma) = \prod_{i=1}^m P(x^i, y^i; \phi, \mu_1, \mu_2, \Sigma)$$
$$= \prod_{i=1}^m p(x^i|y^i)\,p(y^i)$$

Note that here we are maximing the joint likelihood.

In case of discriminative algo we maxing the conditional likelihood

$$\mathcal{L}(\theta) = \prod_{i=1}^{m} p(y^i \mid x^i ; \theta)$$

Now in order to learn the params which maximise $l(\phi, \mu_0, \mu_1, \Sigma)$ we use max likelihood estimation we get :-

$$\phi = \frac{\sum_{i=1}^{m} y^{(i)}}{m} = \frac{\sum_{i=1}^{m} 1\{y^{(i)}=1\}}{m}$$

1 is called indicator notation where,

$$1\{true\} = 1$$
$$1\{false\} = 0$$

$$\mu_0 = \frac{\sum_{i=1}^{m} 1\{y^{(i)}=0\} x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)}=0\}}$$

Numerator is the sum of feature vectors for examples with $y=0$

Denominator is no. of examples with $y=0$

Similarly for μ_1

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

How to make prediction :-

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

→ what is the argmin / argmax notation?

$\min (z-5)^2 = 0$

$\arg\min (z-5)^2 = 5$ as 5 is the value of z we need to plug in to get $\min (z-5)^2 = 0$

Same logic for argmax

$$\arg\max_y p(x|y) = \arg\max_y \frac{p(x|y)}{p(y)}$$

Hence, we need to output the value of y which maximise $p(y|x)$

→ If we plot $p(x|y=0)$ and $p(x|y=1)$ we will get 2 gaussian curves for all $x^{(i)}$

If we try to plot $p(y=1|x)$ for all $x^{(i)}$ we will see that it gives sigmoid function plot similar to the logistic linear regression.

GDA and Logistic LR side-to-side :-

GDA assumes

$$\left.\begin{aligned} x|y=0 &\sim \mathcal{N}(\mu_0, \Sigma) \\ x|y=1 &\sim \mathcal{N}(\mu_1, \Sigma) \\ y &\sim \text{Bern}(\phi) \end{aligned}\right\} \Rightarrow P(y=1|x) = \frac{1}{1+e^{-\theta^T x}} \quad (\because x_0 = 1)$$

Logistic Regr.

$\Leftarrow\!\!\not\;$ logistic

$\therefore$ GDA makes stronger assumption

Logistic LR makes weaker assumptions

If

$$\left.\begin{aligned} x|y=1 &\sim \text{Poisson}(\lambda_1) \\ x|y=2 &\sim \text{Poisson}(\lambda_0) \\ y &\sim \text{Bern}(\phi) \end{aligned}\right\} \Rightarrow p(y=1|x) \text{ is logistic}$$

Naive Bayes :-

Another generative learning algo

(Segway into Natural Language Processing!!)

Step 1 -> Take your sentence and first map it to the feature vector x

Let's say I make a dictionary which contain top 10,000 words which appear in my email.
Then if I get a new email then I can map it to a feature vector x where $x \in \mathbb{R}^n$ $(n = 10{,}000)$
$x[i] = 1$ if email contains ith word of dictionary. (and $x[i] = 0$ otherwise.

We want to model $p(x|y), p(y)$

$2^{10,000}$ possible values of x!

So we need $2^{10,000} - 1$ params to learn!!!

Assume x_i's are conditionally independent given y.

$$p(x_1, \dots, x_{10000}|y) =$$

$$p(x_1|y)p(x_2|x_1,y)p(x_3|x_1,x_2,y) \dots p(x_{10000}|x_1 \dots x_{9999},y)$$

$$\overset{\text{assume}}{=} p(x_1|y)p(x_2|y)p(x_3|y) \dots p(x_{10000}|y)$$

$$= \prod_{i=1}^{n} p(x_i|y)$$

Parameters of this model are:-

$$\phi_{j|y=1} = p(x_j = 1 | y = 1)$$

$$\phi_{j|y=0} = p(x_j = 0 | y = 1)$$

$$\phi_y = p(y=1)$$

Joint likelyhood:

$$\mathcal{L}(\phi_y, \phi_{j|y}) = \prod_{i=1}^{m} p(x^{(i)}, y^{(i)}, \phi_y, \phi_{j|y})$$

MLE:

$$\phi_y = \frac{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} = 1\}}{m}$$

$$\phi_{j|y=1} = \frac{\sum_{i=1}^{m} \mathbb{1}\{x_j^{(i)} = 1, y^{(i)} = 1\}}{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} = 1\}}$$