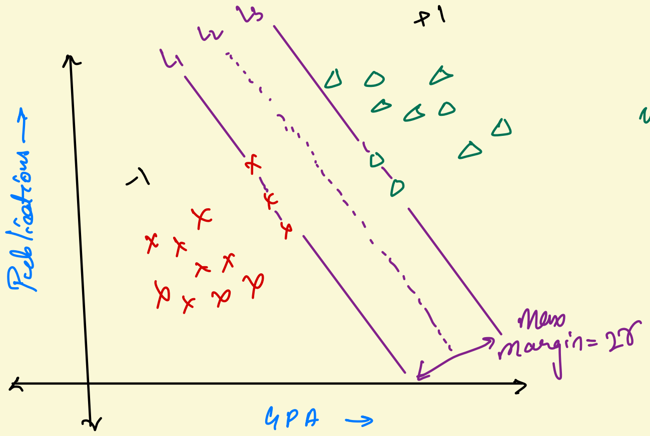
Assignment - 1

S. Soundarya

Q8

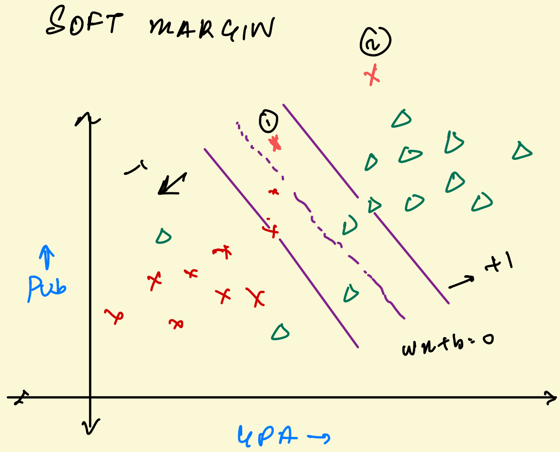


We try to find a hyperplane that is not dangerously close to both the data to avoid misclassification / we want to map. the margin. (γ)

γ: shortest distance b/w the margin & the chosen hyperplane.

* Linear SVMs

SOFT MARGIN



For data that are not linearly seperable.

Classify data so that type I is on one side of Bound & type II is on the other side but with some misclassification With a penalty for every mistake depending upon the mistake

Hinge Loss: $\max[0, 1 - y_i \underline{(wx_i+b)}]$

Incorrectly $y_i(wx_i+b) < 0$

①: $\max(0, 1 - (<0 \,\&\, > -1))$ → lesser

②: $\max(0, 1 - (<0 \,\&\, < -1))$ → greater

Soft SVMs:
Maximize margin
Minimize hinge loss

Maximizing γ = minimizing $\frac{1}{\|w\|}$ $\quad (\because \gamma = 1/\|w\|)$

L1 SVM

$$\min_{w,b,\xi_i} \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i \qquad \xi_i = \max\{0, 1 - y_i(w^T x_i + b)\} \quad 1 \le i \le N$$

PRIMAL

$$\xi_i \ge 0 \qquad \xi_i \ge 1 - y_i(w^T x_i + b)$$

↓ DUAL $\left(\text{Lagrangian } L, \; \frac{\partial L}{\partial w} = \frac{\partial L}{\partial \xi_i} = \frac{\partial L}{\partial b} = \frac{\partial L}{\partial \alpha_i} = \frac{\partial L}{\partial \beta_i} = 0 \right)$

DUAL PROBLEM

$$\underset{\alpha_i}{\text{minimize}} \; \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j - \sum_{i=1}^{N} \alpha_i \quad \equiv$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \quad \text{and} \quad 0 \le \alpha_i \le C$$

VECTORIZED

$$\max_{\alpha} \; 1^T \alpha - \frac{1}{2} \alpha^T Q \alpha$$

subject to $y^T \alpha = 0$

$\alpha \ge 0$

$$\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j = \alpha^T Q \alpha \quad \text{where} \quad Q = [Q_{ij}]_{n \times n}$$

$$[Q_{ij}]_{n\times n} = [y y^T]_{n \times n} \odot \begin{bmatrix} x_1^T x_1 & \cdots & x_1^T x_N \\ & x_i^T x_j & \\ x_n^T x_1 & \cdots & x_N^T x_N \end{bmatrix}$$

$\downarrow$ Inner product b/w x_i's

Try out with $N=2$:

$$y y^T = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} [y_1 \ y_2] = \begin{bmatrix} y_1 y_1 & y_1 y_2 \\ y_2 y_1 & y_2 y_2 \end{bmatrix}$$

np.dot(y, y.T)

$Q = y * x$ // returns element wise

$$\therefore y y^T \odot \begin{bmatrix} x_1^T x_1 & x_1^T x_2 \\ x_2^T x_1 & x_2^T x_2 \end{bmatrix} = \begin{bmatrix} y_1 y_1 x_1^T x_1 & y_1 y_2 x_1^T x_2 \\ y_2 y_1 x_2^T x_1 & y_2 y_2 x_2^T x_2 \end{bmatrix}$$

$$\underset{1\times 2}{[\alpha_1 \ \alpha_2]} \underset{2\times 2}{\begin{bmatrix} y_1 y_1 x_1^T x_1 & y_1 y_2 x_1^T x_2 \\ y_2 y_1 x_2^T x_1 & y_2 y_2 x_2^T x_2 \end{bmatrix}} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = [\alpha_1 y_1 y_1 x_1^T x_1 + \alpha_2 y_2 y_1 x_2^T x_1 \quad \alpha_1 y_1 y_2 x_1^T x_2 + \alpha_2 y_2 y_2 x_2^T x_2] \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix}$$

$$= \alpha_1^2 y_1 y_1 x_1^T x_1 + \alpha_1 \alpha_2 y_2 y_1 x_2^T x_1 + \alpha_2 \alpha_1 y_1 y_2 x_1^T x_2 + \alpha_2^2 y_2 y_2 x_2^T x_2 = \sum_{i=1}^{2}\sum_{j=1}^{2} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

- Hence $\sum_{i=1}^{2}\sum_{j=1}^{2} \alpha_i \alpha_j y_i y_j x_i^T x_j = \alpha^T Q \alpha$

- $1^T \alpha = [1 \ \ 1] \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \alpha_1 + \alpha_2 = \sum_{i=1}^{2} \alpha_i$

- $y^T \alpha = [y_1 \ y_2] \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \sum_{i=1}^{2} y_i \alpha_i = 0$

- Plane.

$$w^* = \sum_{i=1}^{N} \alpha_i^* y_i x_i$$

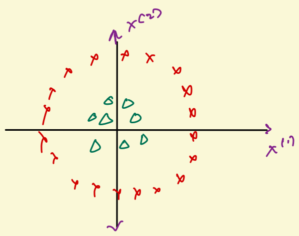
$$b^* = y_i - w^{*T} x_i$$

Plane: $w^* x + b^*$

$$= \sum_i \alpha_i y_i \boxed{< \vec{x}_i, \vec{x} >} + b$$

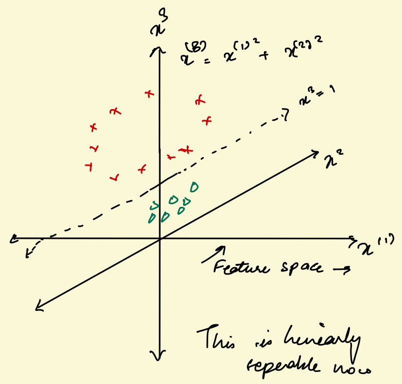
1

Non-Linear SVMs



We can't seperate this by a hyperplane.

Adding a new dimension $h(x^{(1)}, x^2)$



This is linearly seperable now

Idea: Project the data into a higher dim space so that it's linearly seperable in the higher dimension

> Project data to a higher dim where linearly seperable

> Apply linear SVM to the higher dimension

Aim: We just need the innerproducts of the transformation.

$$\max_{\alpha} \; 1^T\alpha - \frac{1}{2}\alpha^T Q \alpha$$

Sub. to $\quad y^T\alpha = 0$

$\alpha \geq 0$

$Q = [Q_{ij}]_{n\times n}$

$$[Q_{ij}]_{n\times n} = [y y^T]_{n\times n} \odot \begin{bmatrix} h(x_1)^T h(x_1) & \cdots & h(x_1)^T h(x_n) \\ \vdots & h(x_i)^T h(x_j) & \vdots \\ h(x_n)^T h(x_1) & \cdots & h(x_n)^T h(x_n) \end{bmatrix}$$

$\phi : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$

$\phi(x_i, x_j) = h(x_i^T) h(x_j)$

Feature space(higher dimensional space)

• First idea was: Learning a non-linear classifier using SVM
– Calculate h(x)for each training example – Find a linear SVM in the feature space.

• Problems:
– Feature space can be high dimensional or even have infinite dimensions, so storing h(x) can be inefficient or even impossible

So we came up with the idea of "Kernels":

-Kernels are similarity functions that return inner products between the images h(xi) of data points.
• Choosing kernel (phi) is equivalent to choosing h(x) in the feature space .

The kernel trick
• No need to know what h(x) is and what the feature space is.
• No need to explicitly map the data to the feature space.(memory constraint resolved)
• Define a kernel function phi and replace the dot produce <x,z> with a kernel function phi(x,z) in both training and testing.

- Plane: $\omega^* = \sum_{i=1}^{N} \alpha_i^* y_i h(x_i)$

$b^* = y_i - \omega^{*T} x_i$

$\omega^* x + b^*$

$$f(\vec{x}) = \sum_i \alpha_i y_i \boxed{\phi(\vec{x_i}, \vec{x})} + b$$

$\phi(x_i, x) = h(x_i)^T h(x)$

Define a function ϕ

```
def phi(x, z):
    return np.dot (x, z)
```

```
Q = np.dot (y, y.T) *
    phi(np.dot (x, x.T))
```

ⓐ Linear Kernel:

$\phi(x, z) = (\alpha x, \alpha z)$, $\alpha \in \mathbb{R}$

$\therefore \phi(x, z) = (h(x), h(z)) = (h(x))^T h(z)$

$= [\alpha x_1, \ldots \alpha x_N] \begin{bmatrix} \alpha z_1 \\ \vdots \\ \alpha z_N \end{bmatrix} = \alpha^2 x_1 z_1 + \alpha^2 x_2 z_2 + \cdots + \alpha^2 x_N z_N$

ⓑ Polynomial Kernel:

$\phi(x, z) = (x^T z + 1)^P$

$\phi(x, z) = (h(x), h(z)) = (x^T z + 1)^P$

$= (x_1 z_1 + x_2 z_2 + \cdots + x_N z_N + 1)^P$

ⓒ RBF Kernel:

$\phi(x, z) = \exp[-\gamma \|x - z\|^2) = (h(x), h(z))$

$\phi(x, z) = e^{-\gamma (\sum_{i=1}^{N} (x_i - z_i)^2)}$

# of samples = m
# of features = n

Q7 Logistic Regression for $K=2$:

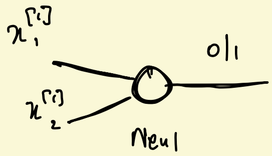
cost func

$$J(w,b) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} y^{[i]} \log(f_{w,b}(x_j^{[i]}) + (1-y^{[i]}) \log(f_{w,b}(x_j^{[i]}))$$

$\frac{\partial J}{\partial w_j}$ & $\frac{\partial J}{\partial b}$

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

BINARY CLASSIFICATION



GD Algo:

repeat

$$\{ \quad w_j = w_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{[i]}) - y^i)\, x_j^{[i]}$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^i) - y^i)$$

$\}$

$K>2$

$K = 10$

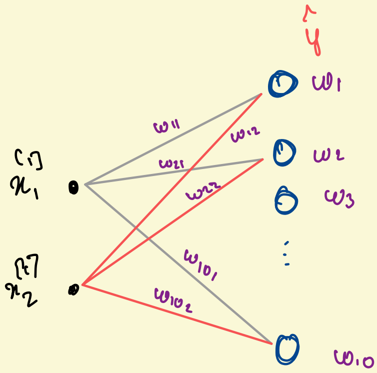
eg: for digit recognition:

8

$\hat{y} =$ Pred. o/p.

$$\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{bmatrix} 0.1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.05 \\ 0 \\ 0.8 \\ 0.05 \end{bmatrix} \qquad y = \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Replace the Sigmoid function by the softmax function



10 o/ps

One single Sample

$$w = \begin{bmatrix} w_{11} & w_{12} \\ \vdots & \\ w_{101} & w_{102} \end{bmatrix}_{10\times 2} \qquad x^{[i]} = \begin{bmatrix} x_1^{[i]} \\ x_2^{[i]} \end{bmatrix}_{2\times 1} \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{10} \end{bmatrix}$$

$$z^{[i]} = \underbrace{wx + b}_{10\times 1}$$

$$z^{[i]} = \begin{bmatrix} z_1^{[i]} \\ z_2^{[i]} \\ \vdots \\ z_{10}^{[i]} \end{bmatrix}$$

$$a^{[i]} = \text{softmax}(z^{[i]})$$

$$a_j^{[i]} = \frac{e^{z_j^{[i]}}}{\sum_{j=1}^{K} e^{z_j^{[i]}}}$$

$$a^{[i]} = \begin{bmatrix} a_1^{[i]} \\ a_2^{[i]} \\ \vdots \\ a_{10}^{[i]} \end{bmatrix}$$

Single sample

Softmax activation function

$$\begin{bmatrix}1.3\\5.1\\2.2\\0.7\\1.1\end{bmatrix} \rightarrow \boxed{\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}} \rightarrow \begin{bmatrix}0.02\\0.90\\0.05\\0.01\\0.02\end{bmatrix}$$

$$L = \text{Loss}(a_1, \ldots, a_N, y) = \begin{cases} -\log a_1, & y = 1 \\ -\log a_2, & y = 2 \\ \vdots & \vdots \\ -\log a_N, & y = N \end{cases}$$

Loss for a single sample : $\sum_{j=1}^{K} -\log(a_j^{[i]})\, y_j^{[i]}$

Cost function: $J(w,b) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{K} (-\log(a_j^{[i]}))\, y_j^{[i]}$ for m samples

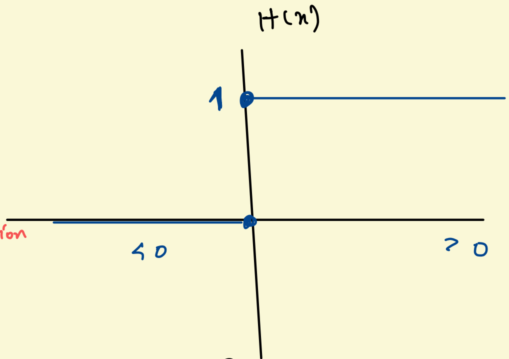
Gradient Descent for this ↑ Refer to the Neural Networks notes for it Derived it there.

MCEs Idea: is to minimize misclassification

$y_i = 0$ or 1

K=2

$$-y_i w^T x_i = \begin{cases} \geq 0, & \text{misclassification} \\ < 0, & \text{correct classification.} \end{cases}$$



$$E_0(w) = \sum_{i=1}^{m} H(-y_i w^T x_i)$$

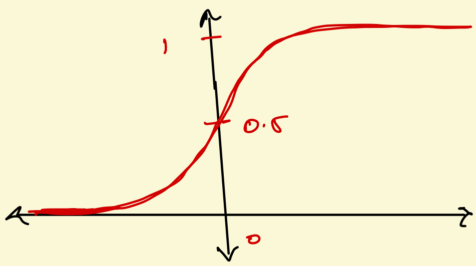
→ counts the no. of misclassification

So if we minimize $E_0(w)$ we are done, but $\nabla H = 0$ i.e except @ origin

Replace heaviside by sig

$$E_1(w) = \sum_{i=1}^{m} \sigma(-y_i w^T x_i)$$

→ Probability of misclassification.



$$E_1(w) = \sum_{i=1}^{m} \frac{1}{1 + e^{-y_i w_1 x_1^{[i]} - y_i w_2 x_2^{[i]} \cdots - y_i w_N x_N^{[i]}}}$$

N features.

$$\frac{\partial E_1}{\partial w_j} = \sum_{i=1}^{m} y_i\, \sigma(y_i w^T x_i)(1 - \sigma(y_i w^T x_i))\, x_j^{[i]}$$

Gradient Descent

repeat

{

$$w_j = w_j - \sum_{i=1}^{m} y_i\, \sigma(y_i w^T x^{[i]}) (1 - \sigma(y_i w^T x^{[i]}))\, x_j^{[i]}$$

$$b = b - \sum_{i=1}^{m} y_i\, \sigma(y_i w^T x^{[i]}) (1 - \sigma(y_i w^T x^{[i]}))$$

}

K ≥ 2

$$\hat{y} = \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{bmatrix} 0.1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.05 \\ 0 \\ 0.8 \\ 0.05 \end{bmatrix} \qquad y = \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Pred. o/p.

149

K Class

For each class k the

$$\hat{y} \cdot y \begin{cases} \geq 0.5, & \text{correctly classified} \\ < 0.5, & \text{incorrectly classified} \end{cases}$$

$E_1(w) =$

Q6 We normalize all the examples (x_i, y_i) so that $\|x_i\| \le 1 \quad \forall i$

Why Margin Perceptron Algorithm? We iterate through the data using perceptron algorithm updating the weights not only on mistakes but also on examples x that the current hypothesis gets correct by a margin $< \frac{\gamma}{2}$ (So if an ex is within margin $\frac{\gamma}{2}$ then it's incorrect)

predict +ive $\frac{w_t^T x_i}{\|w_t\|} \ge \frac{\gamma}{2}$, negative if $\frac{w_t^T x_i}{\|w_t\|} < -\frac{\gamma}{2}$

$\frac{y_i w^{(n)T} x_i}{\|w^n\|} < \frac{\gamma}{2}$ then it's a mistake

$$\|w_{t+1}\|^2 \le \|w_t\|^2 + \|x_i\|^2 + 2y_i w_t^T x_i$$

$$\|w_{t+1}\|^2 \le \|w_t\|^2 + \|x_i\|^2 + 2y_i w^T x_i$$

$$\le \|w_t\|^2 + 2w^T x_i + \|x_i\|^2$$

$$\le \|w_t\|^2 \left(1 + \frac{1}{\|w_t\|} \frac{w_t x_i}{\|w_t\|} + \frac{\|x_i\|^2}{\|w_t\|^2}\right)$$

$$\|w_{t+1}\| \le \|w_t\| \Big(1 + \underbrace{\frac{2}{\|w_t\|} \frac{\gamma}{2} + \frac{1}{\|w_t\|^2}}_{\ge 0}\Big)^{1/2}$$

(for $x > 0$ $\sqrt{1+x} < 1 + \frac{x}{2}$) $\Rightarrow$ Verify by MVT

$$\|w_{t+1}\| \le \|w_t\| \left(1 + \frac{\gamma}{2\|w_t\|} + \frac{1}{2\|w_t\|^2}\right)$$

Hence $\|w_{t+1}\| \le \|w_t\| + \frac{1}{2\|w_t\|} + \frac{\gamma}{2}$

If the classification is $\frac{y_i w_t^T x_i}{\|w_t\|} < \frac{\gamma}{2}$ it's a mistake

$$\frac{\|w_t\|}{y_i w_t^T x_i} > \frac{2}{\gamma}$$

If for some eg we take $y_i w_t^T x_i = 1$ then, $\|w_t\| > \frac{2}{\gamma}$

$\left(\frac{1}{\|w_t\|} < \frac{\gamma}{2} \right)$

$$\|w_{t+1}\| \leq \|w_t\| + \frac{\gamma}{4} + \frac{\gamma}{2} = \|w_t\| + \frac{3\gamma}{4}$$

Repeating it recursively after m mistakes

$$\|w_{M+1}\| \leq \|w_t\| + \frac{3M\gamma}{4} < \frac{2}{\gamma} + \frac{3M\gamma}{4}$$

Also for a perceptron algo recall $M\gamma \leq \|w_{M+1}\|$

So, $$M\gamma \leq \|w_{M+1}\| < \frac{2}{\gamma} + \frac{3M\gamma}{4}$$

$$M\gamma \leq \frac{2}{\gamma} + \frac{3M\gamma}{4} \Rightarrow \frac{M\gamma}{4} \leq \frac{2}{\gamma} \Rightarrow \boxed{M \leq \frac{8}{\gamma^2}}$$

Q5

$$H(x) = -\int p(x) \ln p(x)$$

$$\int p(x)\,dx = 1 \qquad \int p(x)\, x\, dx = \mu$$

$$\int p(x)\,(x-\mu)(x-\mu)^T\, dx = \Sigma$$