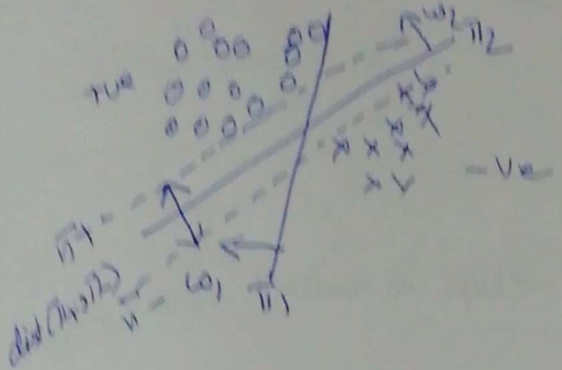
Support Vector Machines (SVM)

→ SVM can do classification & Regression.

→ we have bunch of negative & positive points



→ Key idea of SVM

we have to find the hyperplane (Π) that separates +ve &
-ve pts as widely as positive. possible.

Π_+ & Π_- are parallel to Π

Π_+ & Π_- are also parallel to Each other

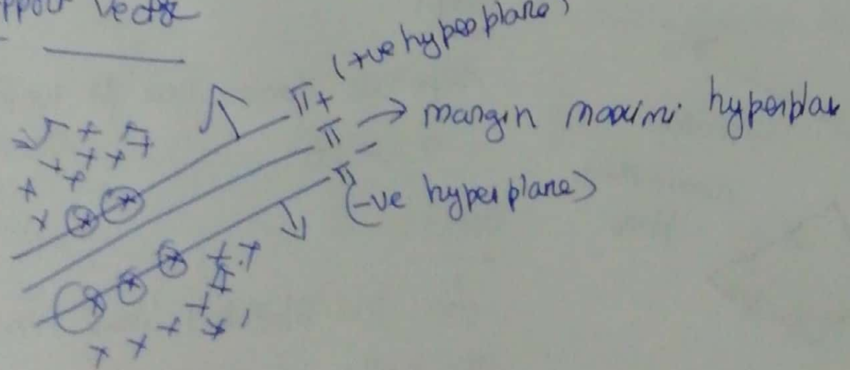
Π: margin maximizing hyper plane.

SVM:- Try to find a Π that maximizes the margin = dist(Π^+, Π^-)

margin↑ ; generalization acc ↑

↳ Error on unseen data.

Support Vectors

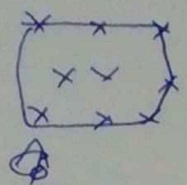


→ pts through which Π^+ & Π^- pass through are support

Vectors.

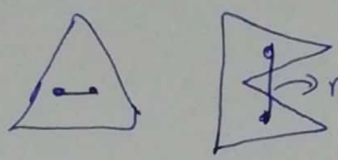
Alternative geometry intuition of SVM:

Convex-hull



Convex-polygon

An points which is passing throught the shape is called convex polygon.

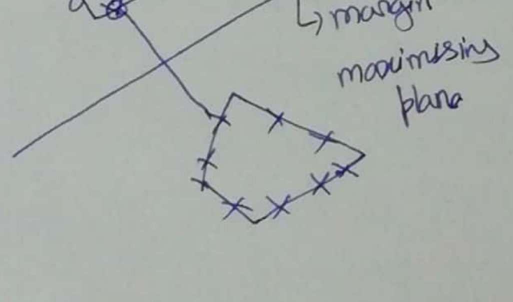


non convex polygon.

Convex polygon : If i want to connect the two points, the line connecting to both the points are inside the polygon is called convex polygon

Convex-hull

A convex hull is given a bunch of pts, If we can build the smallest convex polygon, that has every point inside the convex polygon or on ~~inside~~ the convex polygon.



Step1

separate convex hull for +ve pts & -ve pts

Step2

find the shortest lines connecting the hulls

Step3

bisect the line (Equal parts)

28.2 Mathematical derivation of SVM

π : margin-maximization.

$\pi = w^T x + b = 0$

w is perpendicular to π

$(w^*, b^*) = \underset{(w,b)}{\text{argmax}}\ 2/\|w\|$

Since π^+ & π^- are perpendicular, these are also perpendicular to w

If $\pi^+ = w^T x + b = 1$

$\pi^- = w^T x + b = -1$

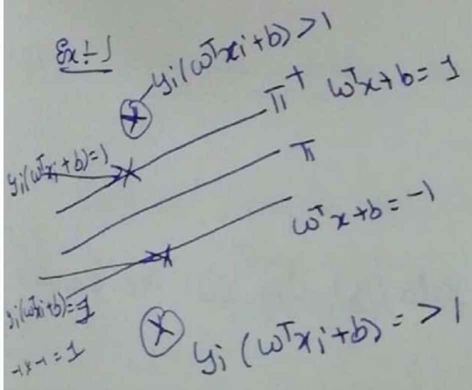
Margin :

$$d = \frac{2}{\|w\|}$$

$$w^*, b^* = \underset{(w,b)}{\text{argmax}} \frac{2}{\|w\|} = \text{margin.} \quad \text{S.t } y_i(w^T x_i + b) \geq 1 \text{ for all } x_i$$

→ hard margin SVM

Ex :- 1



$$w^* b^* = \underset{w,b}{\text{argmax}} \frac{2}{\|w\|}$$

S.t $\forall_i$ (for each) $\quad y_i(w^T x + b) \geq 1$

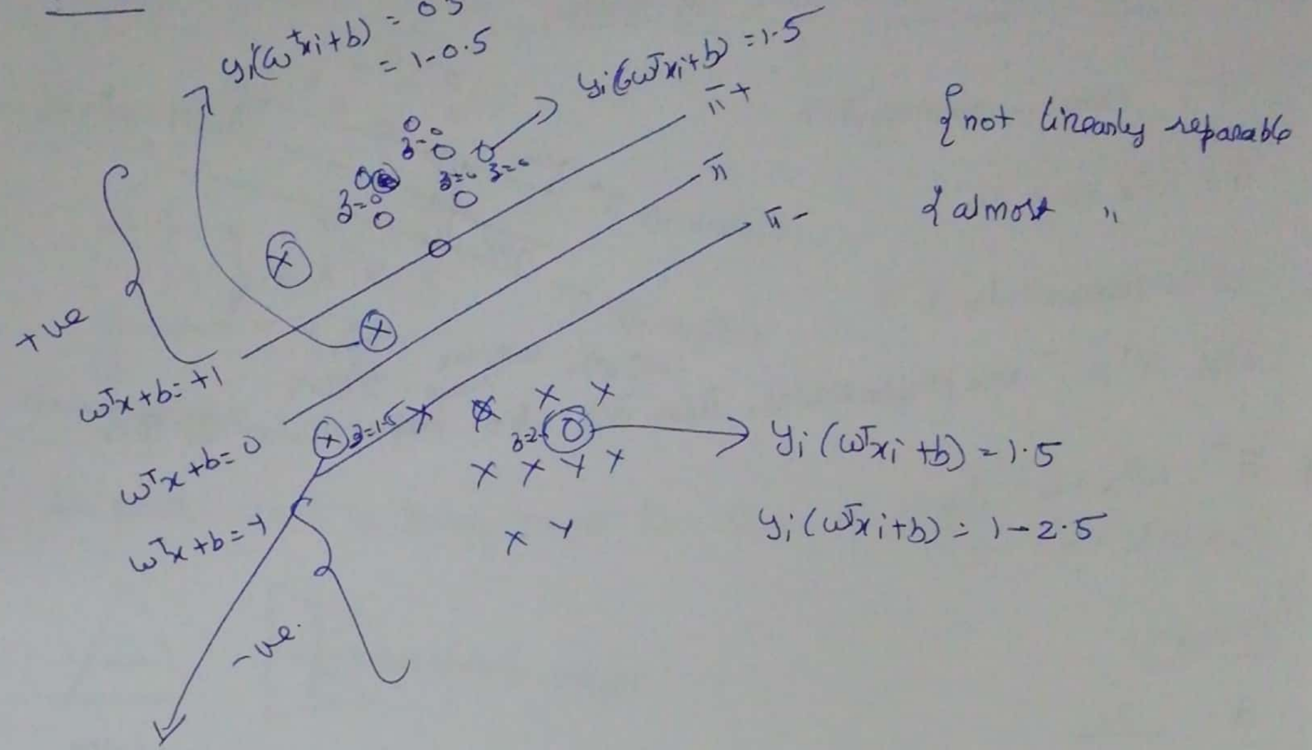
Constraint

All the +ve points should be one side & -ve points on the other side.

There should be any value in the margin area.

→ data should be linearly separable

Ex :2



half way b/w π & π^-

$$\boxed{\begin{aligned} y_i(w^T x_i + b) &= -0.5 \\ y_i(w^T x_i + b) &= (1 - \underbrace{(1.5)}_{\xi_i}) \end{aligned}}$$

(zeta)

→ we will create a new variable called zeta (ξ) for all the points

~~such that~~

$\xi_i \uparrow$, pt is furth away from correct π in the incorrect direction.

$x_i \rightarrow \xi_i$

$\xi_i = 0$ if $y_i(w^T x_i + b) \geq 1$

↑ correctly classified π^+ & π^-

$\xi_i > 0$ & it is equal to the some units of distance away from the correct hyperplane in the incorrect direction.

$$(w^*, b^*) = \underset{(w,b)}{\operatorname{argmax}} \frac{2}{\|w\|} = \underset{(w,b)}{\operatorname{argmin}} \frac{\|w\|}{2}$$

$$w^*, b^* = \underset{(w,b)}{\operatorname{argmin}} \underbrace{\frac{\|w\|}{2}}_{\text{regularization}} + \underbrace{C}_{\text{hyperparameter}} \cdot \left(\frac{1}{n} \sum_{i=1}^{n} \xi_i \right) \rightarrow \text{average distance of missclassification pt for correct } \pi\text{'s}$$

$>>1$

$$\text{s.t.} \begin{cases} y_i (w^T x_i + b) \geq 1 - \xi_i \quad \forall i \text{ (for each)} \\ \boxed{\because \xi_i > 0} \end{cases}$$

For correctly classified points $\xi_i = 0$

we have to minimize misclassification, means $\sum_{i=1}^{n} \xi_i$ should be less.

→ As $C \uparrow$, tendency to make mistake on D_{train} ↓

⇒ overfit ⇒ high variance.

→ As $C \downarrow$; tendency to underfit ⇒ high bias

→ This is called soft margin SVM.

28.3 why we take values +1 & −1 for support vector plane.

$\|w\| \neq 1$ (any vector)
need not be
unit vector

margin $= \frac{2}{\|w\|}$

whole task is to maximize the margin.

① $\pi^+ : w^T x + b = K_1$; $\pi^- : w^T x + b = K_2$ $\boxed{K > 0}$

why we taking only K_1 & $-K_2$, the π^+ & π^- should be equally separable

margin $= \frac{2K}{\|w\|}$; $\underset{(w,b)}{\operatorname{argmax}} \frac{2}{\|w\|} = \underset{w,b}{\operatorname{argmax}} \frac{2K}{\|w\|} = \frac{8}{\|w\|}$ $\rightarrow K = 4$

we took +1 & −1 for convinience.

Case 2 π^+ : $w^Tx+b = k$

$$\left(\frac{w}{k}\right)^T x + \left(\frac{b}{k}\right) = 1$$

$w \perp \pi$

$||w||$ need not be 1

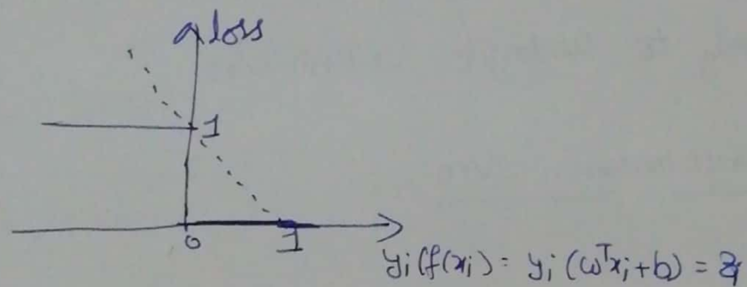
$$(w')^T x + b' = 1$$

28.4 Loss function (Hinge Loss) based interpretation

logistic regression → logistic loss + reg

lr. regression → lr. loss + reg

svm : hinge loss + reg



when $z_i > 0$: x_i correctly classified

$z_i < 0$: x_i is incorrectly classified.

hinge loss

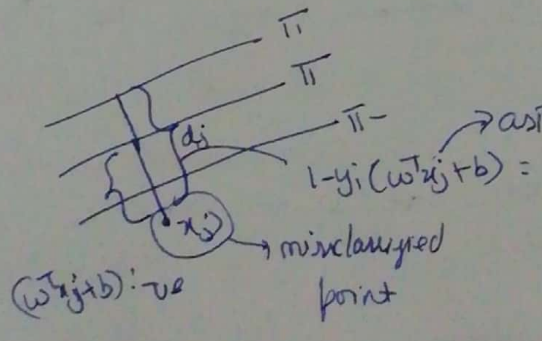
$z_i \geq 1$; hinge loss is 0

$z_i < 1$; hinge loss $= 1 - z_i$

→ $\max(0, 1-z_i)$

case 1 : $z_i \geq 1$; $1-z_i$ is a negative value $\Rightarrow \max(0, 1-z_i) = 0$

Case 2 : $z_i \leq 1$; $1 - z_i > 0 \Rightarrow \max(0, 1-z_i) = 1 - z_i$



as this is negative, negative * negative will become positive.

$d_j = 1 - y_i(w^T x_i + b) = z_j$

$\xi = $ dist of x_j to $\pi^+ = d_j = 1 - z_j$

$\xi_j = 1 - z_j \rightarrow$ when x_j is misclassified

$\max(0, 1 - z_i) = \xi_i$

Soft svm : $\min\limits_{w,b} \frac{\|w\|}{2} + \boxed{C}^{\text{loss}} \sum\limits_{i=1}^{n} \xi_i$

s.t $(1 - y_i(w^T x_i + b) \leq \xi_i$
$\xi \geq 0$

$C\uparrow =$ overfit
$C\downarrow =$ underfit

loss min : $\min\limits_{w,b} \sum\limits_{i=1}^{n} \max(0, 1 - \underbrace{y_i(w^T x_i + b)}_{z_i}) + \boxed{\lambda}^{\text{reg}} \|w\|^2$

$\|w\| \geq 0 \Rightarrow \min \frac{\|w\|}{2}$ is same as $\min \|w\|^2$

$\lambda\uparrow =$ underfit
$\lambda\downarrow =$ overfit

$\xi = 0$; for correctly classified points
$\xi > 0$; for incorrectly classified points.

28.5 Dual form of SVM formulation

dual form of SVM

$$\left\{ \begin{array}{l} \min_{w,b} \ \frac{1}{2}\|w\| + C\sum_{i=1}^{n} \xi_i \\ \text{s.t } \ y_i(w^T x_i + b) \geq 1-\xi_i \ \forall i \\ \xi_i \geq 0 \end{array} \right\} = \max_{\alpha_i} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

Primal of SVM

s.t $\alpha_i \geq 0$ → $\alpha_i = 0$ for non SVs

$\sum_{i=1}^{n} \alpha_i y_i = 0$ $\alpha_i > 0$ for ~~non~~ support

dual form

① for every x_i there is an corresponding α_i

② $x_i^T x_j$ ⇒ x_i's only occur in this form.

End of the day

x_q : $(w^T x_q + b) = f(x_q)$

③ $f(x_q) = \sum_{i=1}^{n} \alpha_i y_i x_i^T x_q + b$

④ $\alpha_i > 0$ only for support vectors

$\alpha_i = 0$ for non support vectors.

$f(x_q)$: only pts that matter are support vectors

$x_i^T x_j = x_i \cdot x_j = \text{cosine sim}(x_i, x_j)$

If $\|x_i\| = 1$; $\|x_j\| = 1$

→ We can replace $x_i^T x_j$ ~~can~~ with any similarity function $\text{sim}(x_i, x_j)$

→ This is what made SVM super popular

→ $\text{Sim}(x_i x_j) = K(x_i, x_j)$

updated Eq

$$\max_{\alpha_i} \sum_{i=1} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \, K(x_i, x_j)$$

Kernal-fn.

$$\text{s.t} \sum_{i=1}^{n} \alpha_i y_i = 0 \; ; \alpha_i \geq 0$$

28.6 Kernel Trick

→ one class of similarity function is Kernel function.

The most important idea in SVM is Kernal.

Soft svm hyperplane ≈ log-reg

↳ margin-max.

If we ~~Ki~~ did not apply Kernal trick and just leave it as $x_i^T x_j$.

Then it is called linear svm

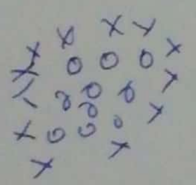
lr. svm : $x_i^T x_j$

Kernal svm : $K(x_i, x_j)$

In a linear svm we are finding margin maximizing hyperplane in the space of x_i's

log-reg we are ~~...~~ finding a hyperplane which minimizes logistic loss in the space of x_i's.

lr.svm & log.reg will in below case.



$f_1, f_2 \rightarrow f_1^2, f_2^2$

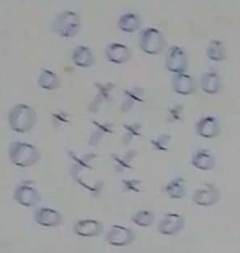
In the above case log-reg + feature transfn will succeed.

Kernal svm will be successful.

↳ It will transform $x_i \rightarrow x_i'$, It tries to find hyperplane in the space of x_i'

Kernel SVM ÷ non-lin. separable data

26.7 Kernel Polynomial



$\rightarrow (f_1, f_2) \xrightarrow{FT} (f_1^2, f_2^2)$

$\downarrow$

log reg

Let's see how Kernelization solve this problem.

$$K(x_1, x_2) = (x_1^T x_2 + c)^d$$

(e.g) $K(x_1, x_2) = \underbrace{(1 + x_1^T x_2)^2}_{\text{Quadratic Kernel}}$

$x_1 = \langle x_{11}, x_{12} \rangle$

$x_2 = \langle x_{21}, x_{22} \rangle$

$$= (1 + x_{11}x_{21} + x_{12}x_{22})^2$$

$$= 1 + x_{11}^2 x_{21}^2 + x_{12}^2 x_{22}^2 + 2x_{11}x_{21} + 2x_{12}x_{22} + 2x_{11}x_{21}x_{12}x_{22}$$

$$[1, x_{11}^2, x_{12}^2, \sqrt{2}x_{11}, \sqrt{2}x_{12}, \sqrt{2}x_{11}x_{12}] = x_1'$$

$$[1, x_{21}^2, x_{22}^2, \sqrt{2}x_{21}, \sqrt{2}x_{22}, \sqrt{2}x_{21}x_{22}] = x_2'$$

$$= (x_1')^T (x_2')$$

$$\begin{array}{ccc} x_1 & & x_2 \\ \downarrow & FT & \downarrow \\ x_1' & & x_2' \end{array} \qquad x_1^T x_2 \rightarrow x_1'^T x_2'$$

Kernalization takes data (d) $\xrightarrow[\text{internally / implicitly}]{FT}$ d' $\boxed{d' > d}$

Mercer's Theorem :-

Says what Kernal-trick is doing

$$\text{not linearly separable} \rightarrow d \longrightarrow d' \longrightarrow \text{lr. separable}$$

typically $d' > d$

SVM → Explicity FT → logi-reg

↳ finding the right kernal.

28.7 RBF - Kernel

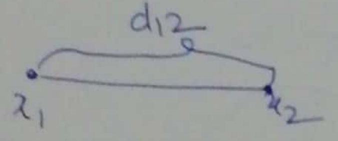
→ Radial Basis Function (RBF)

→ most popular / general purpose kernal : RBF

(x_1, x_2)

$$K_{RBF}(x_1, x_2) = Exp\left(\frac{-\|x_1 - x_2\|^2}{2\sigma^2}\right)$$

σ ← hyper paramete

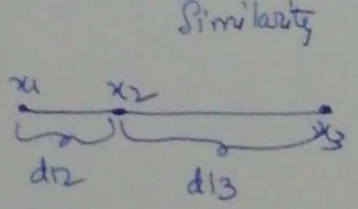


$$\|x_1 - x_2\|^2 = d_{12}^2$$

$$K_{RBF}(x_1, x_2) = Exp\left(\frac{-d_{12}^2}{2\sigma^2}\right) = \frac{1}{e^{d^2/2\sigma^2}}$$

$d_{12} \uparrow$; $\underline{K(x_1, x_2)} \downarrow$

Similarity



$K(x_1, x_2) > K(x_1, x_3)$

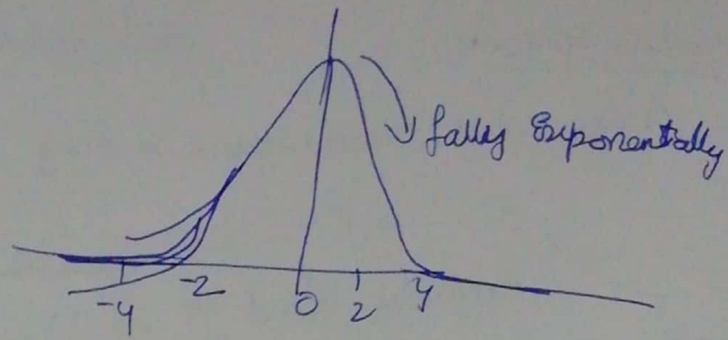
(2) Impact of 'σ'

$\sigma = 1$, $\sigma = 0.1$, $\sigma = 10$

① $d = 0$

$k = 1$

② $d \uparrow$; $k \downarrow$, Exponentially



Kernal → Similar

dist → dissimilar.

RBF ≈ gaussian PDF

Case

$\sigma = 0.1$

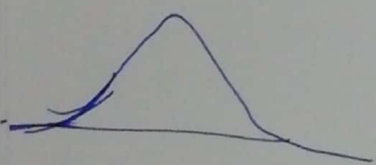
$\sigma^2 = 0.01$

$dis > 1$; $k = 0$

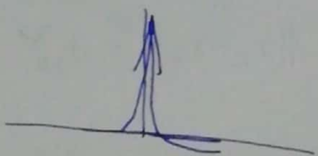
Let σ increase

$\sigma = 10$, $\sigma^2 = 100$

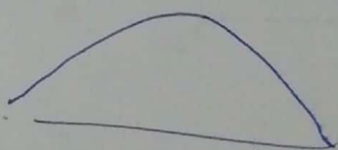
$\sigma = 1$



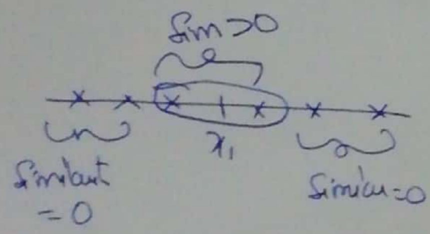
$\sigma = 0.1$



$\sigma = 10$

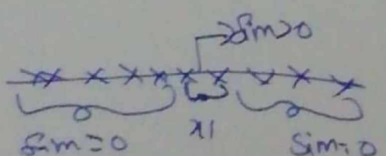


as $\sigma \uparrow$

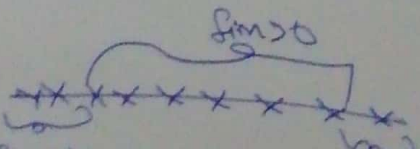


= $\sigma = 1$

$\sigma = 0.1$



$\sigma = 10$



There is a relation b/w K-NN & RBF Kernal.

$\sigma \uparrow \Rightarrow k \uparrow$ in KNN
(in RBF)

RBF-SVM $\approx$ KNN

K-NN : Store all the K pts. & LSH

RBF-SVM : We can just have SV's & its α_i's

$\#$SV's $\ll n$
(number of points)

If we don't have best kernal simply use RBF-SVM.

Soft margin : C & σ

$(C, \sigma) \rightarrow$ grid search
random search.

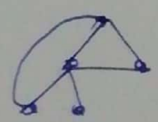
289 Domain Specific Kernels

RBF $\rightarrow$ general purpose kernel

→ String Kernels (for text classification)

→ genome Kernels

→ Graph Kernels



→ given the ~~baten~~ problem, we have to select the appropiate Kernels

For Amazon data set, String kernel will give better results than RBF

28.10 : Train & runtime Complexities

Train → SGD
↳ Specialized algo (dual) → Sequential minimal optimization (SMO)

libsvm: best libraries for training svm's.

Training Time ≈ $O(n^2)$ for kernel svm's

If n is large → $O(n^2)$ is very large.

Typically donot use svm when n is large

applications

Internet

Runtime Complexity

$$f(x_q) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x_q) + b$$

$\alpha = 0$ for non sv's

sv's = K ⟹ $O(Kd)$

of support vector

20.11 nu-SVM: Control Errors & support vectors.

C-SVM → original formulation. $(C \geq 0)$

alternative formulation of SVM :

nu-SVM : $0 \leq nu \leq 1$

nu: hyper parameter

nu ≥ fraction of Errors

SVM

Dtrain

↓

I don't want 10% Errors

↓

nu = 0.1

↓

I don't want 1% Errors

↓

nu = 0.01

Let assume

nu = 0.01 ⇒ % of Errors ≤ 1%

SV's ≥ 1% of n

→ For runtime Complexity : fewer SV's is ideal

28.13 Cases for svm's

→ Feature Engineering & FT

↳ Finding the right kernel (svm tends to work well)

→ Decision Surface

lr. svm's : hyperplane

Kernel-svm's : d x_i → non-linear surface

$\downarrow \quad \downarrow$

d' x_i' → linear surface.

$\boxed{d' > d}$

→ Imagine we are given distance or similarity fn

↳ $k(x_i, x_j)$

→ Interpretability & Feature importance

↳ no way for us to get feature importance directly

↳ we can use forward feature selection

→ Outlier : very little impact on the model

↳ SV's that matter

↳ RBF with a small σ → knn with small 'k'

↳ there too might get impacted with small σ & k

→ Bias Variance

$C \uparrow$ → overfit ↑

$C \downarrow$ → underfit the model

→ large d → v.good for svm ($d \to d'$)

↳ good kernels if we have otherwise RBF

→ Best cases :

↳ right Kernel

→ worst cases

↳ when 'n' is large → Training time is typically high

Ex: internet based application.

↳ K is large =

↳ we cannot have low latency

when n is v.v large ppl endup using Logistic Regression.

28.14 Code Sample

→ on Scikit Learn

→ Sklearn implements SVC, nu-SVC, SVR

// class. sklearn. svm. SVC (C=1.0, Kernal = 'rbf', degree=3, gamma = 'auto',

coef0 = 0.0, shrinking = True, probability = False, tol = 0.001,

cache_size = 200, class_weight = None, Verbose = False, max_iter = -1,

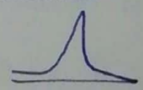
decision_function_shape = 'ovr', random_state = None)

//

→ C : bias & Variance

degree is usefull only when the kernel is polynomial

→ gamma ÷ In RBF we have sigma (σ). In SK-learn it is referred to as gamma $\boxed{\gamma = 1/\sigma}$

$\sigma\downarrow$ $\gamma\uparrow$ 

tol ÷ 0.001

→ Tolerance says when we are moving from i^{th} iteration to $i+1^{th}$ iteration

$(w_i - w_{i+1})$

If difference b/w w_i & w_{i+1} is smaller than tolerance, terminate the loop.

→ class weight = None

Imbalance data if we want to do upsampling then we have to give class weight

→ max_iter = -1

it will iterate until the tolerance is reached

→ decision function_shape

If it is multiclass then we have to provide 'ovr' (one vs rest)

```
import numpy as np
X = np.array([[-1,-1], [-2,-1], [1,1], [2,1]])
y = np.array([1,1,2,2])
from sklearn.svm import SVC
clf = SVC()
clf.fit(X,y)
-> print(clf.predict([[-0.8,-1]]))
o/p + [1]
```

*

gamma

The gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors.

gamma = 10^-1, c = 10^{-2} → g = 10^0, c = 10^{-2}, ⇒ g = 10^1, c = 10^-2

when $\sigma \downarrow$, we are overfitting $\xrightarrow{\text{when moving}}$

(or)

$\gamma \uparrow$

when k = 1, $\sigma \downarrow$ in RBF both are same, they will overfit

Similarly

when c↑ it will overfit

gamma = 10^{-1}, c = 10^{-2}

g = 10^{-1}, c = 10^{0}

g = 10^{-1}, c = 10^{-2}

Sklearn.svm.NuSVC

Only thing will change is nu (nu=0.5)

nu:

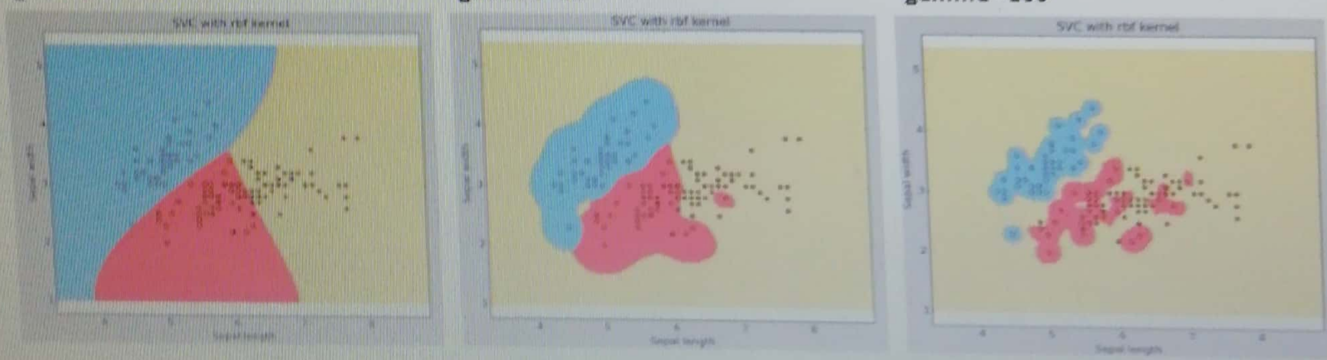
An upper bound on the fraction of training errors and a lower bound of the fraction of support vectors.
Should be in the interval of (0, 1)

gamma: Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. Higher the value of gamma, will try to exact fit the as per training data set i.e. generalization error and cause over-fitting problem.

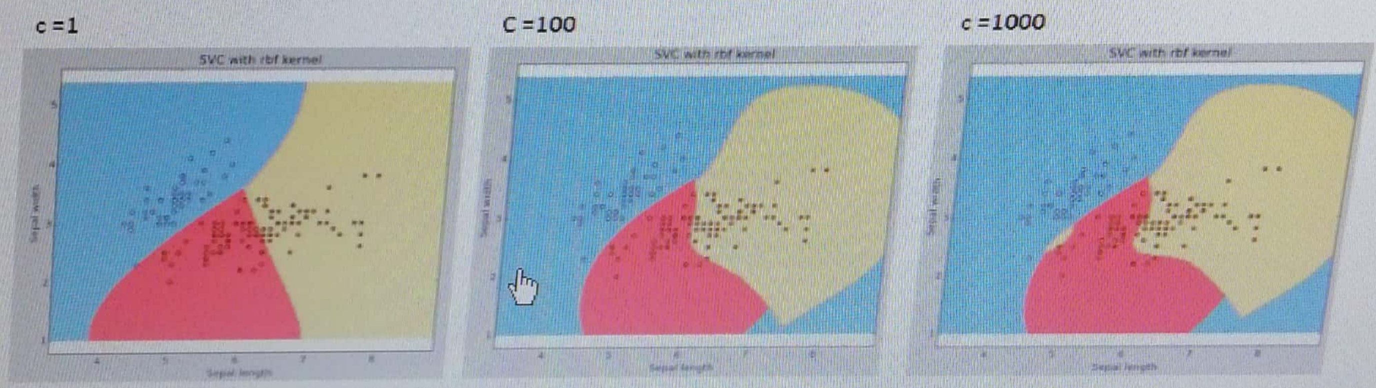
Example: Let's difference if we have gamma different gamma values like 0, 10 or 100.

```
svc = svm.SVC(kernel='rbf', C=1,gamma=0).fit(X, y)
```

gamma =0 gamma =10 gamma =100



C: Penalty parameter C of the error term. It also controls the trade off between smooth decision boundary and classifying the training points correctly.



We should always look at the cross validation score to have effective combination of these parameters and avoid over-fitting.

Pros and Cons associated with SVM

- **Pros:**
  - It works really well with clear margin of separation
  - It is effective in high dimensional spaces.
  - It is effective in cases where number of dimensions is greater than the number of samples.
  - It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

- **Cons:**
  - It doesn't perform well, when we have large data set because the required training time is higher
  - It also doesn't perform very well, when the data set has more noise i.e. target classes are overlapping
  - SVM doesn't directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. It is related SVC method of Python scikit-learn library.

US] | https://medium.com/@aneesha/svm-parameter-tuning-in-scikit-learn-using-gridsearchcv-2413c02125a0

Web Testing: Comple | 180+ Sample Test Ca | How can a Web site b | Expenditure | 500 Oracle interview | Oracle 11g PL/S

per

Medium

Sign in

```
from sklearn import svm, grid_search

def svc_param_selection(X, y, nfolds):
    Cs = [0.001, 0.01, 0.1, 1, 10]
    gammas = [0.001, 0.01, 0.1, 1]
    param_grid = {'C': Cs, 'gamma' : gammas}
    grid_search = GridSearchCV(svm.SVC(kernel='rbf'), param_grid,
cv=nfolds)
    grid_search.fit(X, y)
    grid_search.best_params_
    return grid_search.best_params_
```