MACHINE LEARNING by ambedkar@IISc

- Support Vector Machines
- Kernel Methods

Agenda

Stochastic Gradient Descent and Perceptron

Support Vector Machines

Recall SVMs

Kernel Methods

What if the data is not linearly separable?

Yes! In practice, most often the data is not linearly separable. Then

- Make linearly separable using kernel methods.
- (Or) Use multilayer perceptron.

What are all these?

- The first leads to Support Vector Machines, that rules machine learning for decades
- The second one leads to Deep Learning!

Stochastic Gradient Descent and Perceptron

Recall Gradient Decent for Logistic Regression

Given data $\{x_n, y_n\}_{n=1}^N$,

- We have the following two class classification problem

$$P(y_n = 1 | x_n, w) = \mu_n$$
$$P(y_n = 0 | x_n, w) = 1 - \mu_n$$

where μ_n is defined using logistic function as

$$\mu_n = f(x_n) = \sigma(w^T x_n) = \frac{\exp(w^T x_n)}{1 + \exp(w^T x_n)}$$

Recall Gradient Decent for Logistic Regression

- The loss function that we have incorporated in this problem is cross entropy loss defined as

$$L(w) = -\sum_{n=1}^{N} \left[y_n w^T x_n - \log(1 + \exp(w^T x_n)) \right]$$

- Gradient Decent:

$$w^{(t+1)} = w^{(t)} - \eta \underbrace{\sum_{n=1}^{N} (\mu_n^{(t)} - y_n) x_n}_{\text{Gradient at the previous value}}$$

where $\mu_n^{(t)} = \frac{1}{1+\exp\left(-w^{(t)^T} x_n\right)}$

Stochastic Gradient Decent for Logistic Regression

- Gradient decent requires all the data to calculate the gradient *at each iteration*
- A heuristic that we can apply is the following: approximate the gradient using randomly chosen (x_n, y_n) i.e.

$$w^{(t+1)} = w^{(t)} - \eta_{(t)} \left(\mu_n^{(t)} - y_n \right) x_n$$

- Also replace predicted label probability $\mu_n^{(t)}$ by predicted binary label $\hat{y}_n^{(t)}$, where

$$\hat{y}_n^{(t)} = \begin{cases} 1 \text{ if } \mu_n^{(t)} \geq 0.5 \text{ or } {w^{(t)}}^T x_n \geq 0 \\ 0 \text{ if } \mu_n^{(t)} < 0.5 \text{ or } {w^{(t)}}^T x_n < 0 \end{cases}$$

Stochastic Gradient Decent for Logistic Regression (cont. . .)

- Then the update rule becomes

$$w^{(t+1)} = w^{(t)} - \eta_{(t)}(y_n^{(t)} - y_n)x_n$$

  $w^{(t)}$ gets updated only when there is a misclassification i.e. $\hat{y}_n^{(t)} \neq y_n$
  This is mistake driven update rule.

- Assume that class labels are $+1, -1$

$$\implies \hat{y}_n^{(t)} - y_n = \begin{cases} -2y_n & \text{if } \hat{y}_n^{(t)} \neq y_n^{(t)} \\ 0 & \text{if } \hat{y}_n^{(t)} = y_n^{(t)} \end{cases}$$

Mistake driven learning (contd. . .)

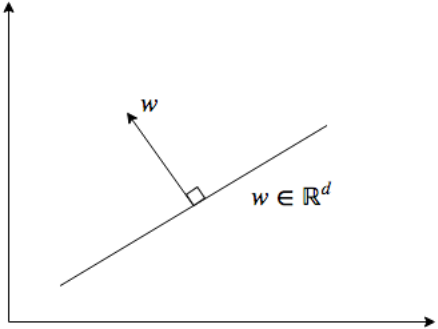
- Whenever there is a misclassification update the weights with the following update rule

$$w^{(t+1)} = w^{(t)} + 2\eta_{(t)} y_n x_n$$

Perceptron learning algorithm is a hyperplane based learning algorithm.

Hyperplanes

- Separates a d-dimensional space into two half spaces (positive and negative).
- $w \in \mathbb{R}^d$ is a normal vector pointing towards positive half.



- Equation of the hyperplane is $w^T x = 0$
- If hyperplane does not pass through origin, we add bias $b \in \mathbb{R}$

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

$$b > 0 : \text{moving it parallely along w}$$

$$b < 0 : \text{opposite direction}$$

Hyperplane based Classifiers

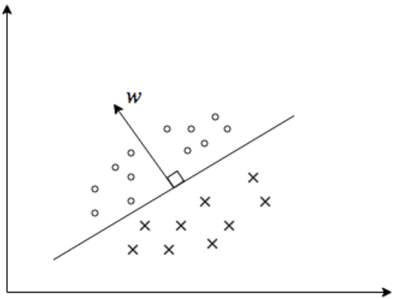
Classification rule

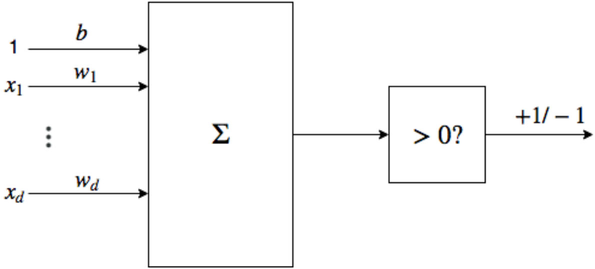
$$y = \text{sign}(w^T x + b)$$

i.e.

$$w^T x + b > 0 \implies y = +1$$

$$w^T x + b < 0 \implies y = -1$$





The Perceptron Algorithm

- Aim is to learn a linear hyperplane to separate two classes.
- Mistake drives online learning.
- Guaranteed to find a separating hyperplane if data is linearly separable.
- If data is not linearly separable
  - Make it linearly separable using kernel methods.
  - (or) Use multilayer perceptron.

What is the best hyperplane for a classification task

- Suppose we have several choices of classifiers, which is the most promising one?
  - promising…from the point view of learning
  - learning…means that has a better generalizing capacity
- Support vector machine provides an answer to this

Distance from a point to a line

- Consider a two dimensional case
- For $a, b, c \in \mathbb{R}$, $ax + by + c = 0$ defines a line in two dimensional plane.
- Let (x_0, y_0) be any point then

$$\text{Distance}(ax + by + c = 0, (x_0, y_0)) = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}$$

Margins

- Let $w^T x + b = 0$ be a hyperplane in $\mathbb{R}^d$.
- Geometric margin is a distance

$$r_n = r_n(w^T x + b = 0, x_n) = \frac{|w^T x + b|}{\|w\|}$$

Since margin is completely determined by w, we write

$$r_n = r_n(w, x_n) = \frac{|w^T x + b|}{\|w\|}$$

- Given a set of points $x_1, x_2, \ldots, x_N$, margin w.r.t. w is

$$r = \min_{1 \leq n \leq N} |r_n| = \min_{1 \leq n \leq N} \frac{|w^T x + b|}{\|w\|}$$

Margins (contd...)

- Functional margin of w on a training sample (x_n, y_n) is defined as

$$f(w, (x_n, y_n)) = y_n(w^T x + b)$$
$$= \begin{cases} \text{positive if } w \text{ predicts } y_n \text{ correctly} \\ \text{negative if } w \text{ predicts } y_n \text{ incorrectly} \end{cases}$$

Loss Function for Hyperplane based Classifiers

- The loss function for hyperplane based classifiers

$$\begin{aligned}\mathcal{L}(w,b) &= \sum_{n=1}^{N} l_n(w,b) \\ &= \sum_{n=1}^{N} \max\{0, -y_n(w^T x_n + b)\}\end{aligned}$$

  - If $y_n(w^T x_n + b) > 0$ then w, b predicts y_n correctly hence $l_n(w,b) = 0$
  - If $y_n(w^T x_n + b) < 0$ then w, b predicts y_n correctly hence $l_n(w,b) = 0$

Stochastic Gradients

- We are going to calculate gradients for l_n not $\mathcal{L}$. (Hence stochastic)

$$\frac{\partial l_n(w,b)}{\partial w} = \begin{cases} -y_n x_n \text{ when } w, b \text{ make a mistake} \\ 0 \text{ otherwise} \end{cases}$$

$$\frac{\partial l_n(w,b)}{\partial w} = \begin{cases} -y_n \text{ when } w, b \text{ make a mistake} \\ 0 \text{ otherwise} \end{cases}$$

- For every mistake, update rule is

$$w = w + y_n x_n$$

$$b = b + y_n$$

(Assuming the learning rate is 1.)

Perceptron Algorithm

Given training data : $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
Initialize $w_{old} = \{0, \ldots, 0\}, b_{old} = 0$
Repeat until convergence

- For a random $(x_n, y_n) \in \mathcal{D}$
- If $y_n(w^T x_n + b) \leq 0$ (or $sign(w^T x_n + b) \neq y_n$, i.e. mistake mode)

$$w_{new} = w_{old} + y_n x_n$$
$$b_{new} = b_{old} + y_n$$

Perceptron Algorithm : In Working

Case 1: Misclassified positive example ($y_n = +1$)

- That is we are in a mistake mode and the perceptron wrongly predicts that

$$
\begin{aligned}
& w_{old}^T x_n + b_{old} < 0 \\
\implies & y_n(w_{old}^T x_n + b_{old}) < 0
\end{aligned}
$$

- Update

$$
\begin{aligned}
w_{new} &= w_{old} + y_n x_n = w_{old} + x_n \text{ (since } y_n = +1) \\
b_{new} &= b_{old} + y_n \quad\;\; = b_{old} + 1
\end{aligned}
$$

- Then

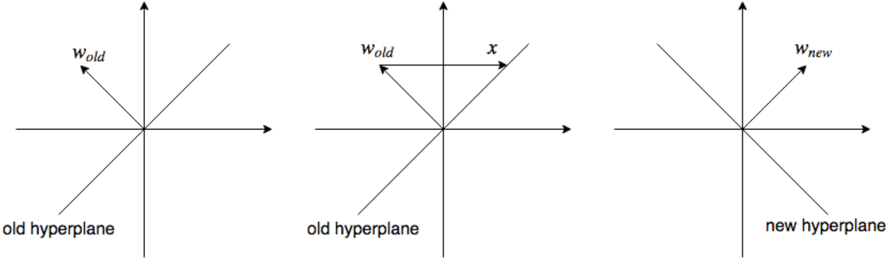
$$
\begin{aligned}
w_{new}^T x_n + b_{new} &= (w_{old} + x_n)^T x_n + b_{old} + 1 \\
&= (w_{old}^T x_n + b_{old}) + x_n^T x_n + 1
\end{aligned}
$$

Perceptron Algorithm : In Working (contd...)

Case 1 (contd...) : Misclassified positive example ($y_n = +1$)

$\Longrightarrow$ $w_{new}^T x_n + b_{new}$ is less negative than $w_{old}^T x_n + b_{old}$

$\Longrightarrow$ Hence, hyperplane gets adjusted in a right direction.



Perceptron Algorithm : In Working (contd. . .)

Case 2: Misclassified negative example ($y_n = -1$)

- Again we are in a mistake mode and perceptron wrongly predicts that

$$w_{old}^T x_n + b_{old} > 0$$
$$i.e. y_n(w_{old}^T x_n + b_{old} < 0$$

- Update

$$w_{new} = w_{old} + y_n x_n = w_old - x_n \text{ (since } y_n = -1)$$
$$b_{new} = b_{old} + y_n = b_{old} - 1$$

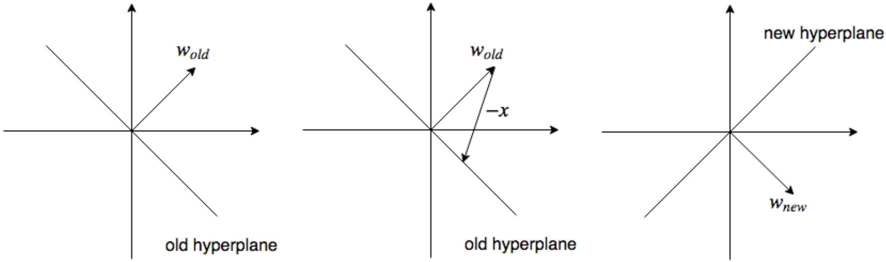
- Then

$$w_{new}^T x_n + b_{new} = (w_{old} - x_n)^T x_n + b_{old} - 1$$
$$= (w_{old} x_n + b_{old}) - (x_n^T x_n + 1)$$

Case 2 (contd...) : Misclassified negative example ($y_n = -1$)

$\implies$ $w_{new}^T x_n + b_{new}$ is less positive than $w_{old}^T x_n + b_{old}$

$\implies$ Hence, hyperplane gets adjusted in a right direction.



Perceptron Convergence Theorem: (Block & Novikoff)

If the training data is linearly separable with margin r by a unit norm hyperplane $w_*(||w_*|| = 1)$ with $b = 0$, then perceptron converges after $\frac{R^2}{r^2}$ mistakes during the training.

Some Final Remarks

- If exists, perceptron finds one of many hyperplanes.
- Of many choices which is the best? : Hyperplane having maximum margin?
- Large margin leads to good generalization on the data.

Support Vector Machines

A bit of history[1]

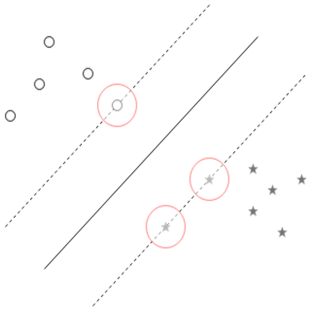
- Pre 1980
  - Almost all learning methods learned linear decision surfaces
  - Linear learning methods have nice theoretical properties
- 1980's
  - Decision trees and Neural Networks allowed efficient learning of non linear decision surfaces
  - Little theoretical basis and all suffer from local minima
- 1990's
  - Efficient learning algorithms for nonlinear functions based on computational learning theory
  - Nice theoretical properties

[1] Slide credit R. Berwick

Introduction (cont...)

- SVM is a hyperplane based classifier
  - That means that our model is linear
  - Later we see how cleverly we can bring in nonlinearity
- Prediction rule $y = \text{sign}(w^T x + b)$
- **Aim:** Given training data $\{(x_1, y_1), \ldots (x_n, y_n)\}$, build a "good" classifier
- **Trick:** Learn w and b such that *achieves maximum margin*

The points in the red circles are called support vectors.



Objective

- Let us consider two class classification problem with class labels $+1$ and -1
- We have the following perceptron objective

$$w^T x_n + b \geq 0 \implies y_n = +1$$

$$w^T x_n + b \leq 0 \implies y_n = -1$$

- We slightly modify our objective

$$w^T x_n + b \geq 1 \implies y_n = +1$$

$$w^T x_n + b \leq -1 \implies y_n = -1$$

Objective (cont...)

One can see that

$$w^T x_n + b \geq 1 \Longrightarrow y_n = +1$$
$$w^T x_n + b \leq -1 \Longrightarrow y_n = -1$$

$$\Downarrow$$

$$y_n(w^T x_n + b) \geq 1$$
$$\Rightarrow \min_{1 \leq n \leq N} |w^T x_n + b| = 1$$

Margin

- Given a set of points $x_1, x_2, \ldots, x_N$, margin w.r.t. w is

$$\gamma(w,b) = \min_{1\leq n\leq N} |r_n| = \min_{1\leq n\leq N} \frac{|w^T x + b|}{\|w\|}$$

- Now since we have

$$\min_{1\leq n\leq N} |w^T x_n + b| = 1$$

- We get

$$\gamma(w,b) = \min_{1\leq n\leq N} \frac{|w^T x_n + b|}{||w||} = \frac{1}{||w||}$$

Optimization Problem

Maximizing the margin

$$\gamma(w, b) = \frac{1}{||w||}$$

$$\Downarrow$$

$$\text{Minimizing } ||w||$$

Optimization Problems:

$$\text{minimize } f(w, b) = \frac{||w||^2}{2}$$

$$\text{subject to } y_n(w^T x_n + b) \geq 1$$

which is a quadratic program with N linearity constraints.

Optimization Problem (cont...)

Data: $\{(x_1, y_1), \ldots (x_N, y_N)\}$

Modal: $w^T x + b = 0$

Parameters: w a d-dimensional vector and b a number

Optimization Problem:

$$\text{minimize } f(w, b) = \frac{||w||^2}{2}$$

$$\text{subject to } y_n(w^T x_n + b) \geq 1$$

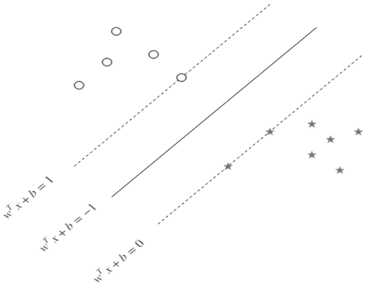
which is a quadratic program with N linearity constraints.

Why a large margin implies good generalization?

- In SVM we have $\gamma \propto \frac{1}{||w||}$
- Large margin $\Rightarrow$ small $||w||$ $i.e$ small l_2 norm.
- Small ||w|| $\Rightarrow$ regularized solution i.e w_i does not become weighing.
- Generalizes very well on the test data.

Hard SVM

Assumption: Every training example need to fulfill the margin condition i.e $y_n(w^T x_n + b) \geq 1$



Objective:

$$\min_{w,b} f(w,b) = \frac{||w||^2}{2}$$

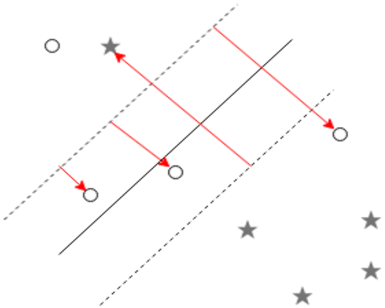
$$\text{subject to } y_n(w^T x_n + b) \geq 1, \quad n = 1, 2, \ldots N$$

Soft Margin

Allow some training examples

- fall within the margin
- misclassified (i.e fall on the wrong side)

ζ : slack : Distance by which it violates the margin



Case 1 : $\zeta_n < 1$: x_n violates the margin but on the right side.
Case 2: $\zeta_n > 0$: x_n not only violates the margin but totally on the wrong side.

Soft SVM (contd ...)

In the case data satisfies

$$y_n(w^T x_n + b) \geq 1 - \zeta_n, \qquad \zeta_n > 0$$

Goal: Not only maximize margins but also minimize the sum of slacks.

Objective: The principle objective is

$$\min_{w,b,\zeta} f(w, b, \zeta) = \frac{||w||^2}{2} + c \sum_{n=1}^{N} \zeta_n$$

subject to $y_n(w^T x_n + b) \geq 1 - \zeta_n, \qquad \zeta_n \geq 0$

This is also convex objective function which is a quadratic program (QP) with $2N$ inequality constraints.

Diversion: Solving constrained optimization problems

Constrained Optimization Problem: Consider

$$\min_w f(w)$$

$$\text{such that } g_n(w) \leq 0, \qquad n = 1, 2, \ldots, N$$

$$h_m(w) = 0, \qquad m = 1, 2, \ldots, M$$

- Constrained optimization problems are difficult to solve
- So we will introduce non-negative lagrange multipliers

  $$\alpha = \{\alpha_n\}_{n=1}^{N} \text{and } \beta = \{\beta_n\}_{n=1}^{M}$$

  one for each constraints
- Lagrangian:
  $\mathscr{L}(w, \alpha, \beta) = f(w) + \sum_{n=1}^{N} \alpha_n g_n(x) + \sum_{m=1}^{M} \beta_m h_m(x)$

Diversion: Solving constrained optimization problem (contd...

Let $\mathscr{L}_p(w) = \max_{\alpha,\beta} \mathscr{L}(w, \alpha, \beta)$

- $\mathscr{L}_p(w) = \infty$ if w violates any of the constraints
- $\mathscr{L}_p(w) = f(w)$ if w satisfies all the constraints

$$\Rightarrow \min_w \mathscr{L}_p(w) = \min_w \max_{\alpha,\beta} \mathscr{L}(w, \alpha, \beta)$$

Further if f, g, h are convex then

$$\min_w \max_{\alpha,\beta} \mathscr{L}(w, \alpha, \beta) = \max_{\alpha,\beta} \min_w \mathscr{L}(w, \alpha, \beta)$$

KKT Condition: At optimal solution

$$\alpha_n g_n(w) = 0 \text{ and } \beta_m h_m(w) = 0$$

Solving hard margin SVM

- We have the following hard margin SVM

$$\min_{w,b} f(w,b) = \frac{||w||^2}{2}$$

$$\text{subject to } 1 - y_n(w^T x_n + b) \leq 0, n = 1, 2, \ldots, N$$

- Lagrangian can be written as

$$\min_{w,b} \max_{\alpha \geq 0} \mathscr{L}(w, b, \alpha)$$

$$= \frac{||w||^2}{2} + \sum_{n=1}^{N} \alpha_n (1 - y_n(w^T x_n + b))$$

- We can solve this by solving the dual problem (Eliminate w and b and solve for dual variables)

Solving hard margin SVM (contd...)

- Derivative of lagragian w.r.t w

$$\frac{\delta \mathscr{L}}{\delta w} = w - \sum_{n=1}^{N} \alpha_n y_n x_n = 0$$

$$\Rightarrow w = \sum_{n=1}^{N} \alpha_n y_n x_n$$

- Derivative of lagragian w.r.t b

$$\frac{\delta \mathscr{L}}{\delta b} = \sum_{n=1}^{N} \alpha_n y_n = 0$$

- Now we substitute $w = \sum_{n=1}^{N} \alpha_n y_n x_n$ in lagragian and also we use $\sum_{n=1}^{N} \alpha_n y_n = 0$

Solving hard margin SVM (contd. . .)

$$\max_{\alpha \geq 0} \mathscr{L}_D(\alpha) = \frac{1}{2}(\sum_{n=1}^{N} \alpha_n y_n x_n)^T (\sum_{n=1}^{N} \alpha_n y_n x_n)$$

$$+ \sum_{n=1}^{N} \alpha_n [1 - y_n (\sum_{m=1}^{N} \alpha_m y_m x_m)^T x_n + b y_n]$$

$$= \frac{1}{2}(\sum_{n=1}^{N} \alpha_n y_n x_n^T)(\sum_{m=1}^{N} \alpha_m y_m x_m)$$

$$+ \sum_{n=1}^{N} \alpha_n - \sum_{n=1}^{N} \alpha_n y_n (\sum_{m=1}^{N} \alpha_m y_m x_m^T) x_n$$

$$+ b \sum_{n=1}^{N} \alpha_n y_n$$

Solving hard margin SVM (contd...)

$$\max_{\alpha \geq 0} \mathscr{L}_D(\alpha) = \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m x_n^T x_m + \sum_{n=1}^{N} \alpha_n$$

$$- \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m x_n^T x_m$$

$$= \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m x_n^T x_m$$

$$\text{such that } \sum_{n=1}^{N} \alpha_n y_n = 0$$

Let $G_{mn} = y_m y_n x_m^T x_n$ a $n \times n$ matrix

Then the optimization problem is :

$$\max_{\alpha \geq 0} \mathscr{L}_D(\alpha) = \alpha^T 1 - \frac{1}{2}\alpha^T G \alpha \qquad s.t \sum_{n=1}^{N} \alpha_n y_n = 0$$

Solving hard margin SVM (contd...)

- We have a maximization of a concave function. (because Hessian of G is p.s.d)
- Note that the original primal SVM objective is also convex
- The input x appear as inner product have one can apply something called "kernel trick".
- On solving dual optimization problem We can treat the objective on a quadratic program and by running QP solver like quadprog, CPLE etc.

Solving hard margin SVM (contd...)

- once we solve for α_n, w and b can be computed :

$$w = \sum_{n=1}^{N} \alpha_n y_n x_n$$

$$b = -\frac{1}{2}\left(\min_{x:y_n=\pm 1} w^T x_n + \max_{x:y_n=-1} w^T x_n\right)$$

- most $\alpha_n{}'s$ will be zero.
  - $\alpha_n \neq 0$ only if x_n lies on one of the two margin boundaries

$$\text{i.e } y_n(w^T x_n + b) = 1$$

  - These one called support vectors.

Solving soft margin SVM

- Optimization problems:

$$\min_{w,b,\zeta} f(w,b,\zeta) = \frac{||w||^2}{2} + c\sum_{n=1}^{N}\zeta_n$$

$$\text{subject to } 1 \le y_n(w^T x_n + b) + \zeta_n, \qquad \zeta_n \ge 0$$

$$n = 1, 2, \ldots, N$$

- By introducing lagrange multiplier

$$\min_{w,b,\zeta} \max_{\alpha\ge 0,\beta\ge 0} \mathscr{L}(w,b,\zeta,\alpha,\beta)$$

$$= \frac{||w||^2}{2} + c\sum_{n=1}^{N}\zeta_n + \sum_{n=1}^{N}\alpha_n(1 - y_n(w^T x_n + b) - \zeta_n) - \sum_{n=1}^{N}\beta_n\zeta_n$$

Solving soft margin SVM (contd...)

- Next step is to eliminate the primal variables w, b, ζ to get dual problem containing dual variable

$$\frac{\delta \mathscr{L}}{\delta w} = 0 \Rightarrow w = \sum_{n=1}^{N} \alpha_n y_n x_n$$

$$\frac{\delta \mathscr{L}}{\delta b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0$$

$$\frac{\delta \mathscr{L}}{\delta \zeta_n} = 0 \Rightarrow c - \alpha_n - \beta_n = 0$$

Solving soft margin SVM (contd ...)

- This gives

$$\max_{\alpha \leq C, \beta \geq 0} \mathscr{L}_D(\alpha, \beta) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m,n=1}^{N} \alpha_m \alpha_n y_m y_n (x_m^T x_n)$$

$$\text{such that} \sum_{n=1}^{N} \alpha_n y_n = 0$$

(Note dual variable β does not appear)

$$\Rightarrow \max_{\alpha \leq C} \mathscr{L}_D(\alpha) = \alpha^T 1 - \frac{1}{2} \alpha^T G \alpha \qquad s.t \sum_{n=1}^{N} \alpha_n y_n = 0$$

where $G_{mn} = y_m y_n x_m^T x_n$ a NxN matrix

- **Note:**
  - $\alpha' s$ are again sparse
  - Nonzero $\alpha_n' s$ corresponds to the support vector.

The Nature of support vectors

- Hard Margin SVM : It has only one type of support vectors.
  - Lying on the margin boundaries

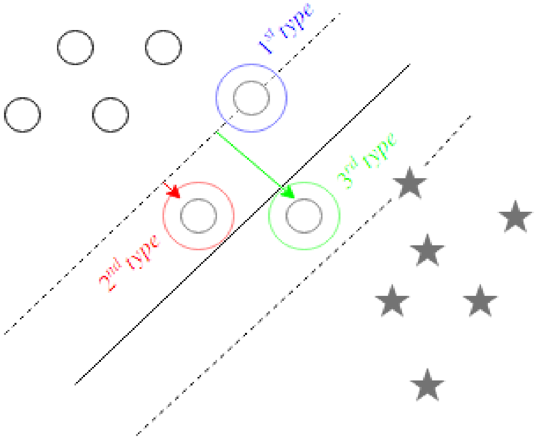
$$w^T x + b = -1 \text{and } w^T x + b = +1$$

- Soft Margin SVM : Three types of support vectors
  - Lying on the margin boundaries

    $$w^T x + b = -1 \text{ and } w^T x + b = +1 (\zeta = 0)$$

  - Lying within the margin region $(0 < \zeta_n < 1)$ but still on the correct side.
  - Lying on the wrong side of the hyperplane $(\zeta_n \geq 1)$

The nature of support types



The nature of support types

SVM via Dual Formulation

Hard Margin SVM

$$\max_{\alpha \geq 0} \mathscr{L}_D(\alpha) = \alpha^T 1 - \frac{1}{2}\alpha^T G \alpha \qquad \text{s.t } \sum_{n=1}^{N} \alpha_n y_n = 0$$

Soft margin SVM

$$\max_{\alpha \leq C} \mathscr{L}_D(\alpha) = \alpha^T 1 - \frac{1}{2}\alpha^T G \alpha \qquad \text{s.t } \sum_{n=1}^{N} \alpha_n y_n = 0$$

Advantages of Dual Formulation:

- The dual problem has only one constraint that is non trivial ($\sum_{n=1}^{N} \alpha_n y_n = 0$)
  The original primal formulation of SVM has many more (N- number of training examples)
- Allow non linear separator by replacing the linear product by kernalized similarities.

SVM via Dual Formulation

Drawbacks of Dual Formulation

- Dual formulation can be expensive if N (The size of the data) is very large $\Rightarrow$ Have to solve for N variables $\alpha = [\alpha_1, \ldots, \alpha_N]$

- Need to store an $N \times N$ matrix G

Loss functions in hyperplane based classifier

- Perceptron Loss: $l(w, b) = \sum_{n=1}^{N} l_n(w, b)$

$$= \sum_{n=1}^{N} \max\{0, -y_n(w^T x_n + b)\}$$

- SVM Loss: For each training sample we need

$$y_n(w^T x_n + b) \geq 1 - \zeta_n$$

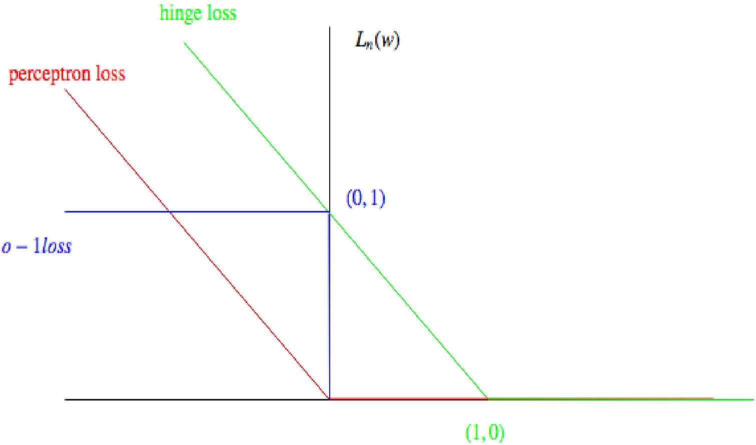
$$Loss = \text{Sum of slacks}$$

$$= \sum_{n=1}^{N} l_n(w, b)$$

$$= \sum_{n=1}^{N} \zeta_n$$

$$= \sum_{n=1}^{N} \max\{0, 1 - y_n(w^T x_n + b)\}$$

Loss Functions in hyperplane based classifier



Loss functions

Recall SVMs

Objective

- Let us consider two class classification problem with class labels +1 and −1
- We have the following perceptron objective

$$w^T x_n + b \geq 0 \implies y_n = +1$$

$$w^T x_n + b \leq 0 \implies y_n = -1$$

- We slightly modify our objective

$$w^T x_n + b \geq 1 \implies y_n = +1$$

$$w^T x_n + b \leq -1 \implies y_n = -1$$

Optimization Problem (cont...)

Data: $\{(x_1, y_1), \ldots (x_N, y_N)\}$

Modal: $w^T x + b = 0$

Parameters: w a d-dimensional vector and b a number

Optimization Problem:

$$\text{minimize } f(w, b) = \frac{||w||^2}{2}$$

$$\text{subject to } y_n(w^T x_n + b) \geq 1$$

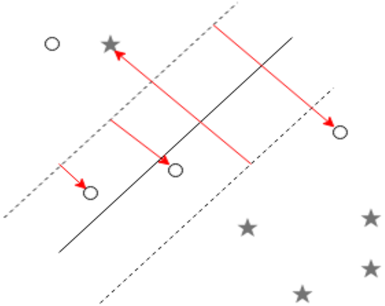
which is a quadratic program with N linearity constraints.

Soft Margin

Allow some training examples

- fall within the margin
- misclassified (i.e fall on the wrong side)

ζ : slack : Distance by which it violates the margin



Case 1 : $\zeta_n < 1$: x_n violates the margin but on the right side.
Case 2: $\zeta_n > 0$: x_n not only violates the margin but totally on the wrong side.

Soft SVM (contd . . .)

In the case data satisfies

$$y_n(w^T x_n + b) \geq 1 - \zeta_n, \qquad \zeta_n > 0$$

Goal: Not only maximize margins but also minimize the sum of slacks.

Objective: The principle objective is

$$\min_{w,b,\zeta} f(w, b, \zeta) = \frac{||w||^2}{2} + c \sum_{n=1}^{N} \zeta_n$$

subject to $y_n(w^T x_n + b) \geq 1 - \zeta_n, \qquad \zeta_n \geq 0$

This is also convex objective function which is a quadratic program (QP) with $2N$ inequality constraints.

Solving soft margin SVM (contd . . .)

- This gives

$$\max_{\alpha \leq C, \beta \geq 0} \mathscr{L}_D(\alpha, \beta) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m,n=1}^{N} \alpha_m \alpha_n y_m y_n (x_m^T x_n)$$

$$\text{such that} \sum_{n=1}^{N} \alpha_n y_n = 0$$

(Note dual variable β does not appear)

$$\Rightarrow \max_{\alpha \leq C} \mathscr{L}_D(\alpha) = \alpha^T 1 - \frac{1}{2} \alpha^T G \alpha \qquad s.t \sum_{n=1}^{N} \alpha_n y_n = 0$$

where $G_{mn} = y_m y_n x_m^T x_n$ a NxN matrix

- **Note:**
  - $\alpha' s$ are again sparse
  - Nonzero $\alpha_n' s$ corresponds to the support vector.

Kernel Methods

The notion of Similarity and Distance

- Consider a d dimensional real space $\mathbb{R}^d$
- Consider two points $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$
- When do we say the point x is similar to point y or how do we measure the similarity between x and y
- What is the distance between x and y

Linear models depend on "linear" notion of similarity and distance

$$\text{similarity}(x_n, x_m) = x_n^T x_m$$

$$\text{Distance}(x_n, x_m) = (x_n - x_m)^T (x_n - x_m)$$

Going from one space to another

Use feature mapping function ϕ to map data to new space (usually high dimensional) where the original learning problem becomes easy i.e

$$\phi : \mathbb{X} \rightarrow \mathbb{F}$$

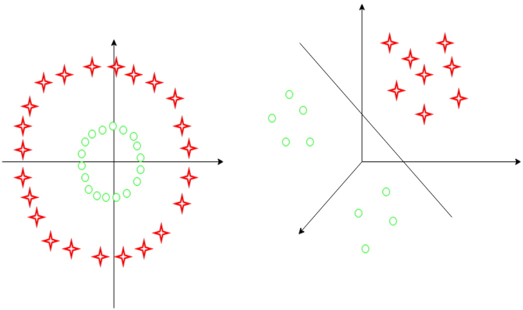
$$\mathbb{X} : \text{space that the original data lies}$$

$$\mathbb{F} : \text{some high dimensional space}$$

Consider the following mapping

$$\phi : \mathbb{R}^2 \to \mathbb{R}^3$$

$$(x_1, x_2) \to (x_1^2, \sqrt{2}x_1x_2, x_2^2) = (z_1, z_2, z_3)$$



Cover's Theorem on the Seperability of Patterns

By Thomas Cover, 1965

A complex pattern-classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly seperable than in a low-dimensional space, provided that the space is not densely populated

- This motivates use of nonlinear kernels in various machine learning methods.
- Kernel methods dominated ML for many years.

Thomas Cover was an information theoretist

What could be the problem with the mappings?

- Constructing these mappings can be expensive, specially when the new space is high dimension.
- Storing and using the mappings in later computation can be way expensive.
- Kernels side-step these issues by defining on "implicit" feature map.

Kernel : Example

Consider $x = (x_1, x_2) \in \mathbb{R}^2$, $z = (z_1, z_2) \in \mathbb{R}^2$

Define a function

$$\begin{aligned}
K : \mathbb{R}^2 \times \mathbb{R}^2 &\to \mathbb{R} \\
K(x, z) &= (x^T z)^2 \\
&= (x_1 z_1 + x_2 z_2)^2 \\
&= x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1 x_2 z_1 z_2 \\
&= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)(z_1^2, \sqrt{2} z_1 z_2, z_2^2) \\
&= \phi(x)^T \phi(z)
\end{aligned}$$

Kernel : Example (contd. . .)

We have

$$\begin{aligned} K : \mathbb{R}^2 \times \mathbb{R}^2 &\rightarrow \mathbb{R} \\ K(x, z) &= (x^T z)^2 \\ &= \phi(x)^T \phi(z) \end{aligned}$$

K implicitly defines a mappings ϕ to a higher dimensional space $\phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$ and computes inner product based similarity $\phi(x)^T\phi(x)$ in that space

Kernels : Examples (contd . . .)

- We did not need to predefine/compute the mapping ϕ to compute $K(x, z)$

- The function K is known as the kernel function

- Evaluating K is almost as fast as computing inner product.

- Any kernel function K implicitly defines an associated feature mapping ϕ

Kernel : Definition

Feature mapping:

$$\phi : \mathcal{X} \to \mathcal{F}$$

Kernel function:

$$K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

$$(x, z) \to \phi(x)^T \phi(z)$$

Note: Not every K with $K(x, z) = \phi(x)^T \phi(z)$, for some ϕ is not a kernel. K needs to satisfy Mercer's condition

Mercer Condition

- K is symmetric and positive semidefinite

$$\Downarrow$$

  K must define a dot product for some higher space $\mathcal{F}$

- The function K is p.s.d if

$$\int\int f(x)K(x,z)f(z)\mathrm{d}x\mathrm{d}z \geq 0$$

  for every function f that is square integral i.e

$$\int f(x)\mathrm{d}x < \infty$$

Algebraic operations on Kernels

$$
\begin{aligned}
K(x,z) &= K_1(x,z) + K_2(x,z) \\
K(x,z) &= \alpha K_1(x,z) \\
K(x,z) &= K_1(x,z) K_2(x,z)
\end{aligned}
$$

Examples of Kernels

- Linear kernel : $K(x,z) = x^T z$
- Quadratic kernel : $K(x,z) = (x^T z)^2$ or $(1 + x^T z)^2$
- Polynomial kernel : $K(x,z) = (x^T z)^d$ or $(1 + x^T z)^d$
- Radial basis function(RBF) : $K(x,z) = \exp(-r||x - z||^2)$

Kernel Matrix

Given the data $\{x_1, x_2, \dots, x_N\}$, where $x_n \in \mathcal{X}$, $n = 1, 2, \dots N$, kernel K is a function

$$K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$
$$K(x_i, x_j) \mapsto \phi(x_i)^T \phi(x_j)$$

that defines a $N \times N$ matrix K as

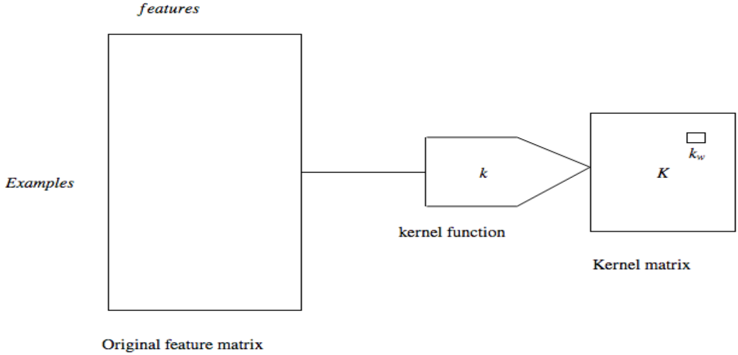
$$K_{ij} = K(x_i, x_j)$$

which gives similarity between i^{th} and j^{th} example in the feature space $\mathcal{F}$.

Important Properties of Kernel Matrix

- The matrix K is
  - Symmetric i.e. $K = K^T$
  - Positive definite i.e $z^T K z > 0, \qquad \forall z \in \mathbb{R}^N$
    $\Rightarrow$ all eigenvalues are positive.

Kernel Matrix (contd. . .)



Kernel matrix

On using kernels

- Kernels can turn linear models to nonlinear models. In any model during training and test if input appear as $x_i^T x_j$ then these models can be kernalised by replacing $x_i^T x_j$ with $\phi(x_i^T)\phi(x_j) = K(x_i, x_j)$
- The following learning algorithm can be kernalized
  - Distance based methods, Perceptron, SVM, linear regression.
  - Many unsupervised learning algorithms like k-means clustering, PCA.

Kernalized SVM training

- The soft margin SVM dual problem is

$$\max_{\alpha \leq C} \mathscr{L}_D(\alpha) = \alpha^T 1 - \frac{1}{2}\alpha^T G \alpha \qquad s.t \sum_{n=1}^{N} \alpha_n y_n = 0$$

  -

$$G_{mm} = y_m y_n x_m^T x_n = y_m y_n K_{mn}$$

  - we can replace the inner product with a kernel function as

$$K_{mn} = K(x_m, x_n) = \phi(x_m)^T \phi(x_n)$$

- Now SVM learn a linear separator in the kernel induced feature space $\mathbb{F}$, which is a nonlinear separators in the original space.

Kernalized SVM training (contd...)

- For a new test sample x

$$
\begin{aligned}
y = \operatorname{sign}(w^T x) &= \operatorname{sign}\left(\sum_{n=1}^{N} \alpha_n y_n x_n^T x\right) \\
&= \operatorname{sign}\left(\sum_{n=1}^{N} \alpha_n y_n K(x_n, x)\right)
\end{aligned}
$$

- The SVM weight vectors is

$$
w = \sum_{n=1}^{N} \alpha_n y_n \phi(x_n) = \sum_{n=1}^{N} \alpha_n y_n K(x_n, .)
$$

- Note w can be explicitly computed and stored only if the feature map ϕ of K can be explicitly written i.e K can be written as

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

which is not always possible.

kernel Ridge regression

- Ridge repgression problem

$$w = arg\min_{w} \sum_{n=1}^{N} (y_n - w^T x_n)^2 + \lambda w^T w$$

- The solution is

$$w = \left(\sum_{n=1}^{N} x_n x_n^T + \lambda I_d \right) \left(\sum_{n=1}^{N} y_n x_n \right) = (X^T X + \lambda I_d)^{-1} X^T Y$$

Matrix Identity: We use the following identity from the matrix algebra

$$(B^T R^{-1} B + P^{-1})^{-1} B^T R^{-1} = P B^T (B P B^T + R)^{-1}$$

Substitute the following

$$R = I_N$$

$$B = X$$

$$P = I_D$$

Kernel Ridge regression (contd...)

- We get

$$w = X^T(XX^T + \lambda I_n)^{-1}y$$
$$= X^T\alpha = \sum_{n=1}^{N} \alpha_n x_n$$
$$\text{where } \alpha = (XX^T + \lambda I_n)^{-1}y = (K + \lambda I_N)^{-1}y$$
$$K_{nm} = x_n^T x_m \Rightarrow K = XX^T$$

Here α is a $Nx1$ vector of dual variables.

- Now we kernalize the model.

$$w = \sum_{n=1}^{N} \alpha_n \phi(x_n) = \sum_{n=1}^{N} \alpha_n L(x_n, .)$$
$$\text{where } \alpha = (K + \lambda I_N)^{-1}y$$
$$K_{nm} = \phi(x_n)^T \phi(x_m)$$
$$= K(x_n, x_m)$$

Kernel Ridge regression (contd ...)

For a test input x, predict the output y as

$$
\begin{aligned}
y = w^T \phi(x) &= \sum_{n=1}^{N} \alpha_n \phi(x_n)^T \phi(x) \\
&= \sum_{n=1}^{N} \alpha_n K(x_n, x)
\end{aligned}
$$

Learning from kernels: Some remarks

- RBF kernel works well in practice.
- Hyperparameters of the kernel may need to be tuned via cross validation
- There are approaches that use multiple kernel which called "Multiple kernel learning".

On kernels and Feature learning

Let $x_1, x_2, \ldots, x_N$ be given data in $\mathbb{R}^D$. Then Gram matrix is defined as

$$K = \begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \ldots & K(x_1, x_N) \\ K(x_2, x_1) & K(x_2, x_2) & \ldots & K(x_2, x_N) \\ & & & \\ & & & \\ K(x_N, x_1) & K(x_N, x_2) & \ldots & K(x_N, x_N) \end{bmatrix}$$

For any x_n define the following N-dim vectors:
$\psi(x_n) = K(n, .) = [\ K(x_n, x_1)\ K(x_n, x_2)\ , \ \ldots\ K(x_n, x_N)]$

- $\psi(x_n)$ can be considered as the new feature representation of x_n
- Each feature represents similarity of x_n with other inputs.