Applications of Convex Optimization in Machine Learning

Igor Vorona
Anh-Huy Phan

Skoltech

Dec 2023

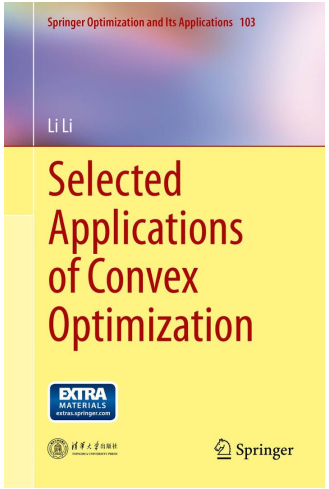
Useful materials

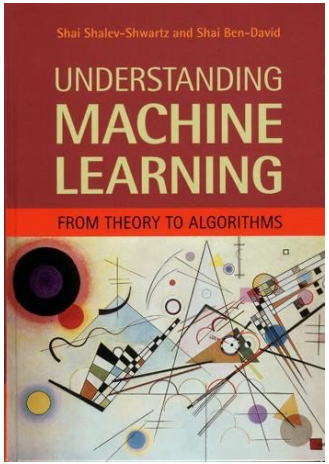
Some lectures with similar topics

- “Introduction to Convex Optimization for Machine Learning” by John Duchi
- “Optimization for Machine Learning” by Suvrit Sra

Books:

- “Selected Applications of Convex Optimization” by Li Li
- “Theory of Convex Optimization for Machine Learning” by Sebastien Bubeck
- “Understanding Machine Learning: From Theory to Algorithms” by Shai Shalev-Shwartz and Shai Ben-David. [Also, exist Russian translation]





Outline

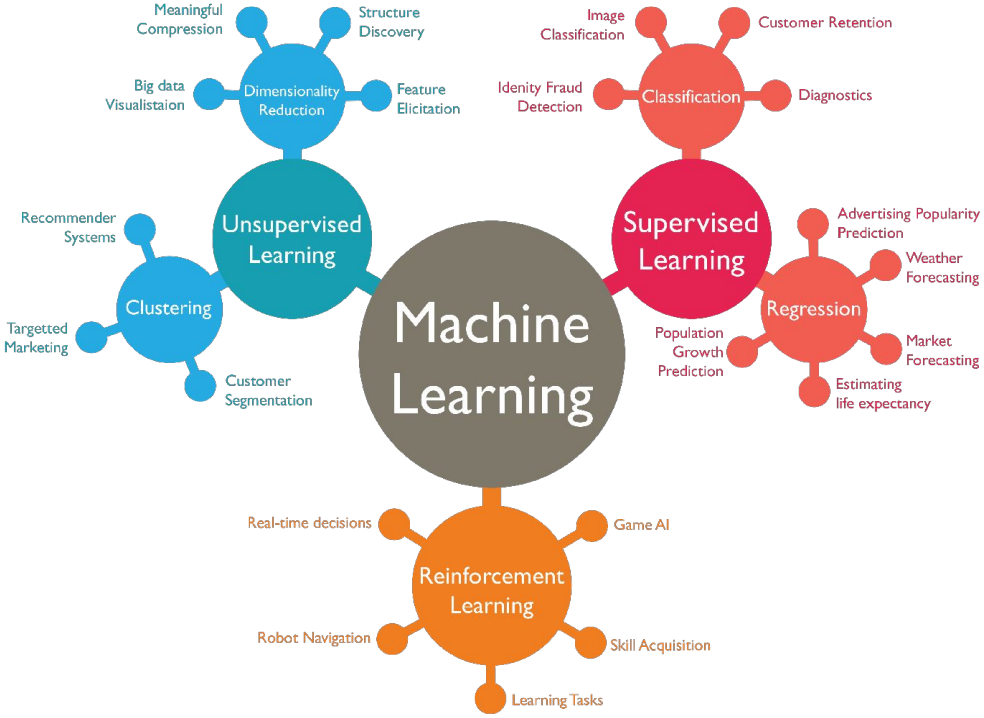
- Intro
- Supervised learning (Classification & Regression)
- Unsupervised learning (Clustering)
- Practise

Intro

Main Message

Machine Learning

- Sometimes when we can't solve the problem exactly, we can approximate its solution through previous experience. ML is about it.
- We can see any ML problem as some generalization of function approximation problem to the stochastic setting.
- 3 main ML task categories
    - Supervised
    - Unsupervised
    - Reinforcement



Formal model of statistical learning

- **Domain set:** $\mathcal{X}$ **-** it's a space of training data which we want to label
- **Label set:** $\mathcal{Y}$ - it's a space of labels (outcomes of our algorithm)
- **Training data:** $S = ((x_1, y_1), \ldots, (x_m, y_m)) \in \mathcal{X} \times \mathcal{Y}$
- **The learner's output:** $h : \mathcal{X} \to \mathcal{Y}$ (h - is our ML method h = hypothesis)
- **"Correct" labeling function:** $f : \mathcal{X} \to \mathcal{Y}$ f is the function, which we want to approximate
- **Data-generation model:** $\mathcal{D}$ **-** is the probability distribution from which data a generated.
- **Measures of success:** $L_{\mathcal{D},f}(h) = \mathbb{P}_{x \sim \mathcal{D}}[h(x) \neq f(x)] = \mathcal{D}(\{x : h(x) \neq f(x)\})$

Empirical Risk Minimization

Training error (empirical risk)

$$L_S(h) = L_{\mathcal{D}(\text{uniform over S})}(h) = \frac{|\{i \in [m] : h(x_i) \neq y_i\}|}{m},$$

Where $[m] = \{1, \ldots, m\}$

(we work under i.i.d assumption) $L_{\mathcal{D}}(h) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[\ell(h,(x,y))]$

To

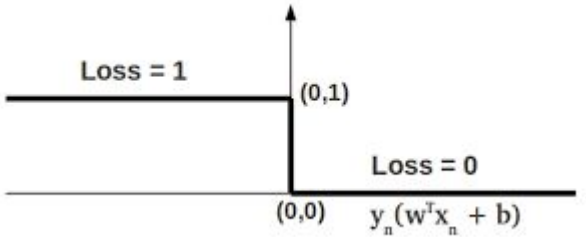
$$L_S(h) = \frac{1}{m}\sum_{i=1}^{m} \ell(h,(x,y)_i)$$

Where $\ell(h,(x,y)) : \mathcal{H} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_+$ is a *loss function*, which measure discrepancy between prediction and answer. Our task minimize this function.

Examples of loss functions

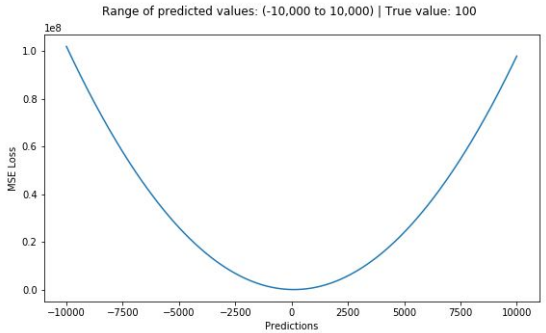
- 0 - 1 loss (popular for binary classification)

$$\ell_{0-1}(h,(x,y)) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } h(x) = y \\ 1 & \text{if } h(x) \neq y \end{cases}$$



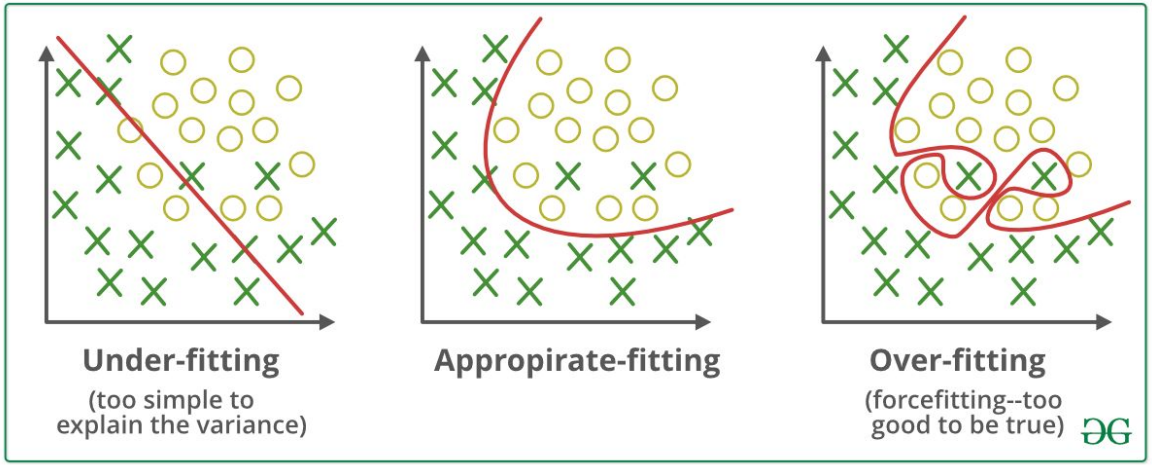
- Square loss (popular for regression task)

$$\ell_{\text{sq}}(h,(x,y)) \stackrel{\text{def}}{=} (h(x) - y)^2.$$



Not so easy... problems

Many ML problems are ill-posed due to existence of multiple (possibly infinite) solutions.



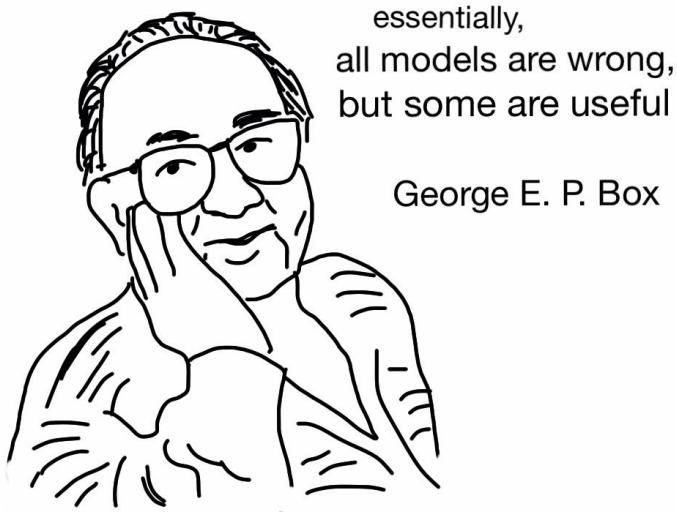
On this figure crosses and circles are geometrical interpretation of data points from training set

Inductive bias

- “No Free Lunch Theorems” => it doesn’t exist universal model and universal algorithm for all problems
- We need some reweighting procedure for our hypothesis h(x)
- Generally, we prefer hypothesis based on

+ Prior Knowledge about data

+ Principle of simplicity (“Occam’s razor”)



Convex ML Problems

- From this point of view the Convex Optimization Theory give us toolbox to encode our prior knowledge in a **simple** and **efficient** way. It give us efficient access to global optima's, in other words model became **efficiently learnable**.

- Theorem. If x is a local minimizer of a convex optimization problem, it is a global minimizer.

Convex surrogate loss

General idea to make the ML problem convex:

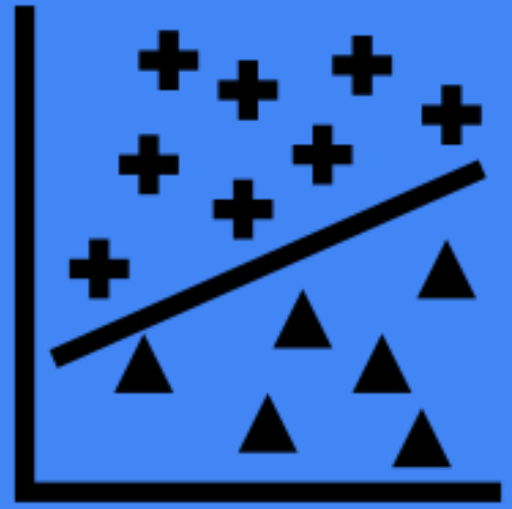
Take your loss function and majorize it via convex surrogate in following way:

And if $$\ell_{original}(h,(x,y)) \leq \ell_{convex}(h,(x,y))$$

Then $$\min_h \ell_{original}(h,(x,y)) = \min_h \ell_{convex}(h,(x,y))$$

$$\{h|h = \arg\min_h \ell_{convex}(h,(x,y))\} \subseteq \{h|h = \arg\min_h \ell_{original}(h,(x,y))\}$$

Supervised



General setting

- Supervised, because we have training dataset with correct (x, y) pairs

- For simplicity, we start from two categories:

$\mathcal{X} \in \mathbb{R}$ $\mathcal{Y} = \{-1, +1\}$

$$\ell_{0-1}(h, (x, y)) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } h(x) = y \\ 1 & \text{if } h(x) \neq y \end{cases}$$

0-1 loss encode our understanding of misclassification in the most proper way

Linear classifiers

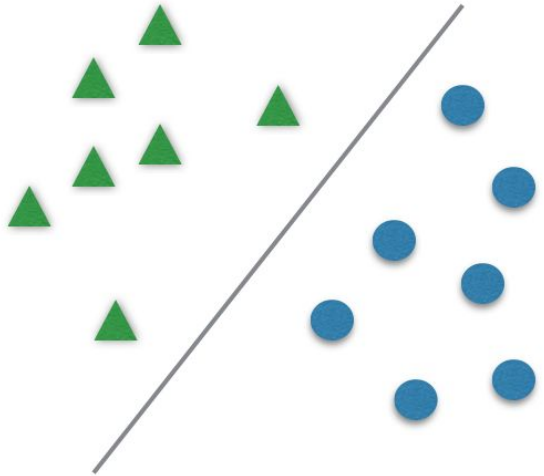
Main idea:

Classify by weighted sum of features

Weighted sum induced hyperplane in $\mathbb{R}$

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

Decision by $h(x) \geq 0$



Our problem 0-1 loss reformulated as optimization problem:

$$\text{minimize}_{w,b} \frac{1}{m} \sum_{i=1}^{m} \max(0, -y_i(w^T x_i + b))$$

Disadvantage of this formulation: loss-function is non-convex and also model allow non-unique solution.

Upper bounds of 0-1 loss

A straightforward way is to use mean squared error loss:

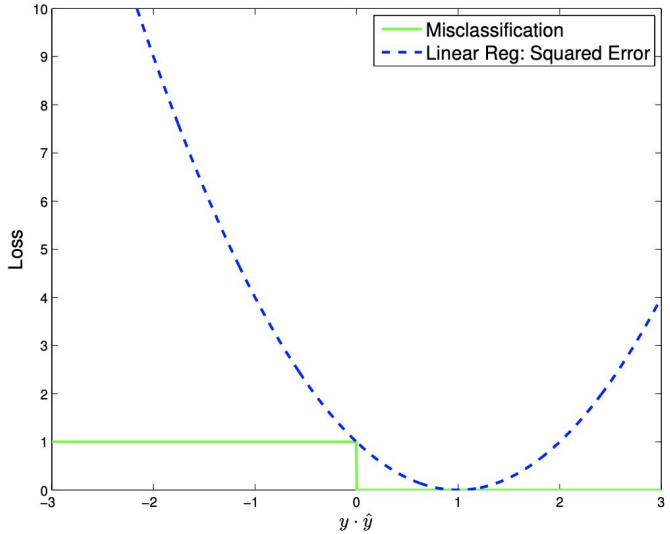
$$\ell_{\text{sq}}(h,(x,y)) \overset{\text{def}}{=} (h(x)-y)^2.$$

Where

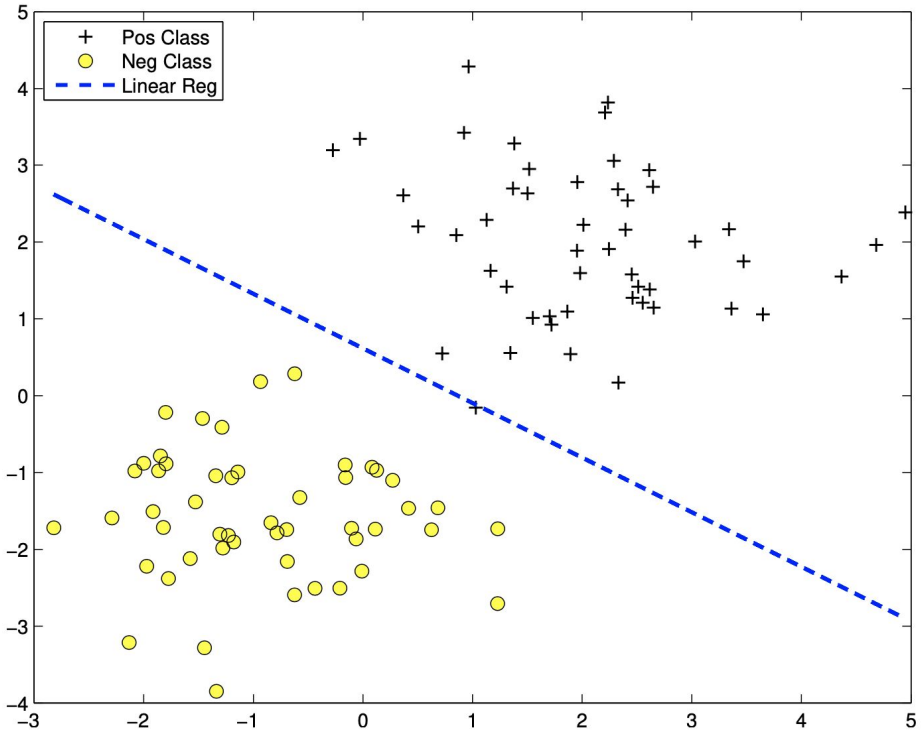
$$\hat{y} = h(x) = w^T x + b$$

Notice, that

$$|\hat{y}_i - y_i|^2 = |\hat{y}_i - y_i|^2 \cdot y_i^2 = (1 - y_i\hat{y}_i)^2$$



Decision boundary of squared loss



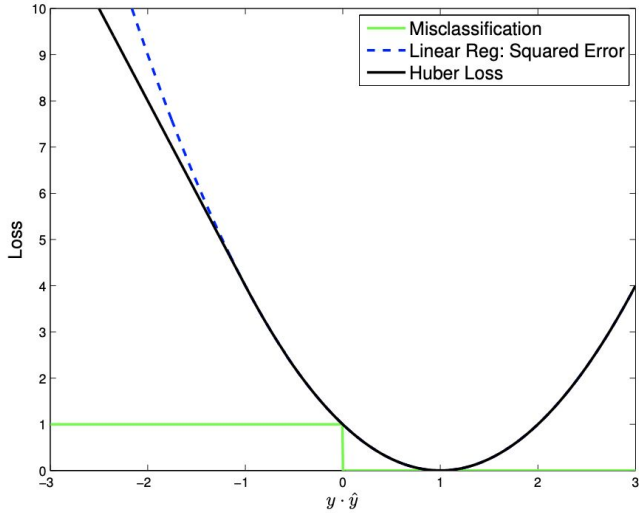
Not enough good (x in the center on the decision boundary). Can we find closer upper bound?

Huber loss

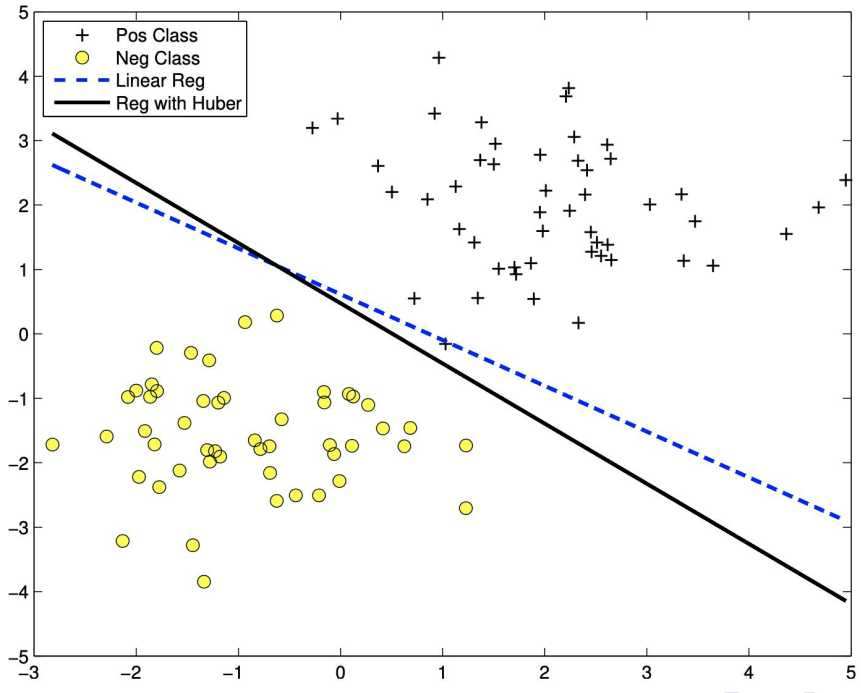
Huber Loss (with parameter M):

$$\phi_{hub}(x) = \begin{cases} |x|^2 & |x| < M \\ M(2|x| - M) & |x| \geq M \end{cases}$$

Select $M = 2$, define loss as $\phi_{hub}(1 - y\hat{y})$:



Decision boundary of Huber loss



Good! But can we do better?

Logistic loss

Logistic loss is the best way to majorize loss from class of **smooth** and **convex** functions

It combines Maximum Likelihood Estimation with Sigmoid function

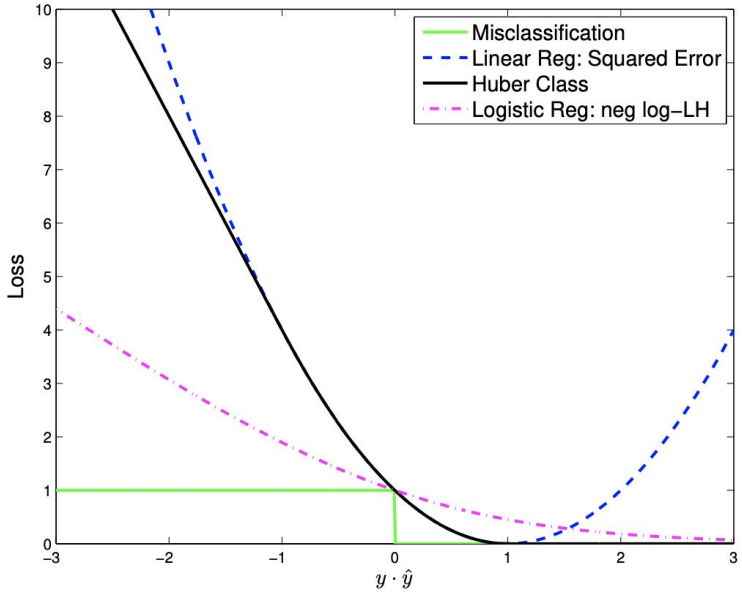
$$\ell_{MLE}(h,(x,y)) = -\log(h(x))$$

$$h(x) = \sigma(x) = \frac{1}{1+\exp(-(w^T x+b))}$$

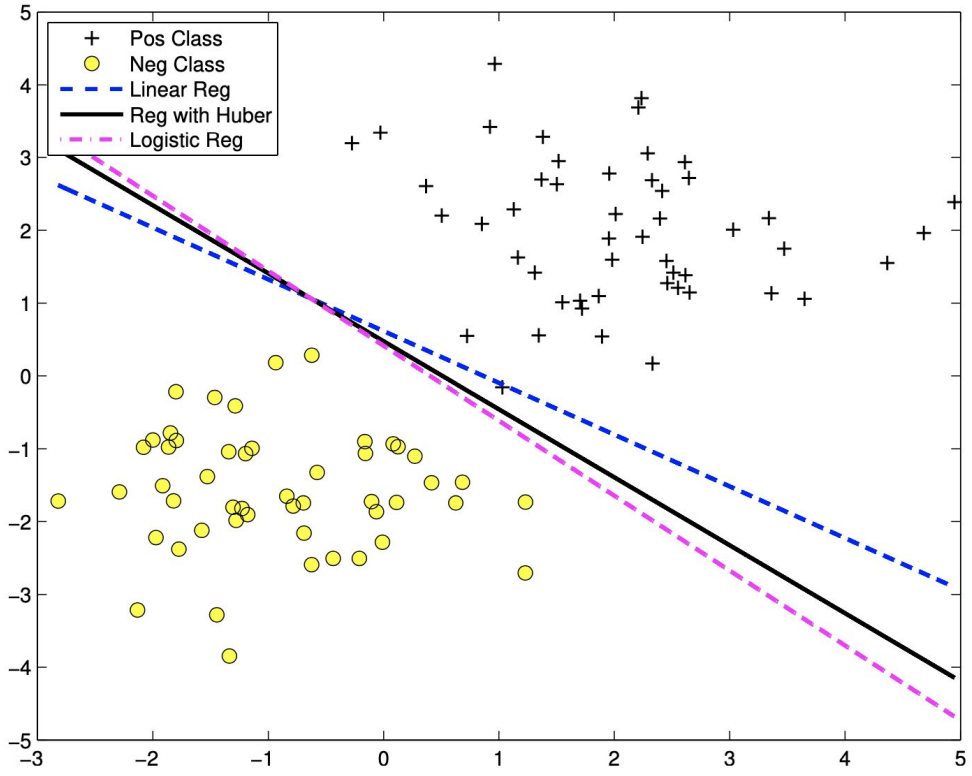
$h(x)$ generate probabilistic outcomes

$$\mathbf{P}(y=+1|w,x) = h(x)$$

$$\mathbf{P}(y=-1|w,x) = 1-h(x)$$



Decision boundary of logistic loss

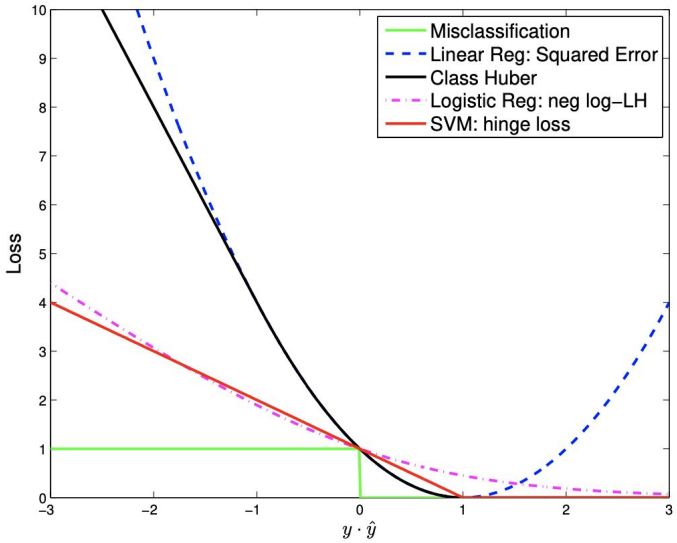


Hinge loss

$$\max(0, -y_i(w^T x_i + b)) \leq \max(0, 1 - y_i(w^T x_i + b))$$

Best way to majorize 0-1 loss via **convex** function is piecewise linear function called hinge loss.

Due to convexity of hinge loss problem is much easier, but still not unique.

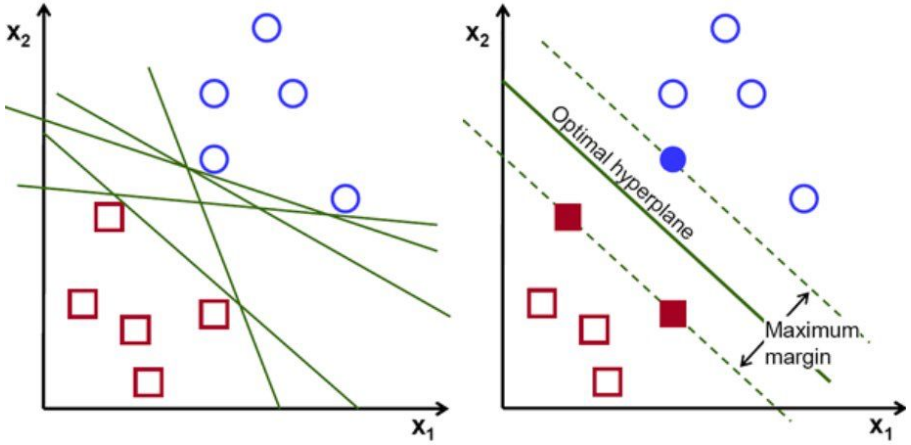


Optimal separating hyperplane

There are many hyperplanes, which can separate two classes.

Optimal Separating Hyperplane should:

1. Separate the two classes
2. Maximize the distance to the closest point from either class.



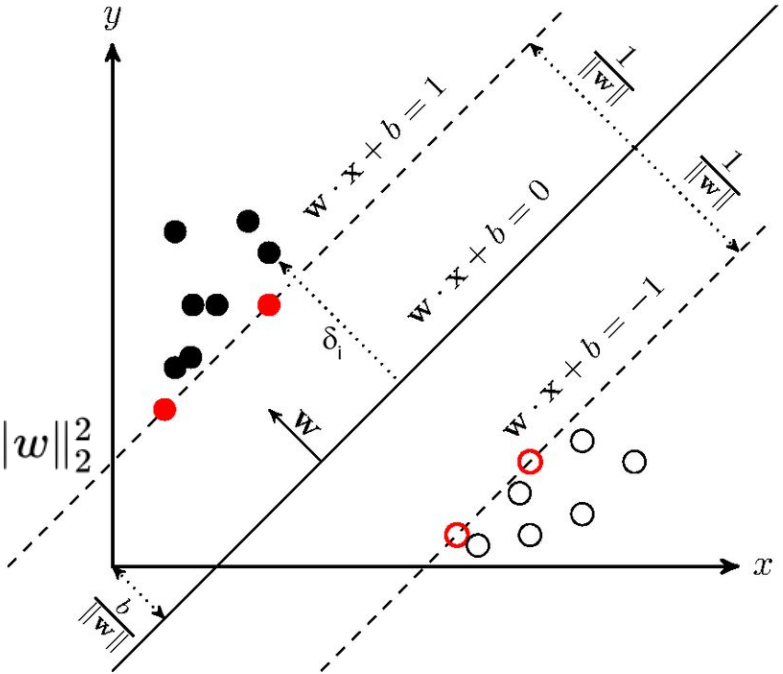
Support Vector Machine

SVM loss = Hinge Loss + Euclidean norm of

weights.

And we go to this problem:

$$\text{minimize}_{w,b} \frac{1}{m} \sum_{i=1}^{m} \max(0, 1 - y_i(w^T x_i + b)) + \lambda \|w\|_2^2$$



In this case we reweight our problem solutions.

We prefer solution with maximum margin (margin is equal to 1, but margin unit is inverse proportional to ||w||) between two classes.

SVM optimization methods

- We can optimize SVM objective with subgradient descent (GD on differentiable areas of loss function), which is better for huge datasets.

- But for small dataset is good to reformulate task as constrained quadratic optimization as in original SVM paper.

- We can use hard constraints reformulation : $\max(0, 1 - y_i(w^T x_i + b)) \rightarrow y_i(w^T x_i + b) \geq 1$

  Then we have following QP:

$$\text{minimize}_{w,b} \frac{1}{2} ||w||_2^2$$

subject to

$$y_i(w^T x_i + b) \geq 1, i = 1, \ldots, m$$

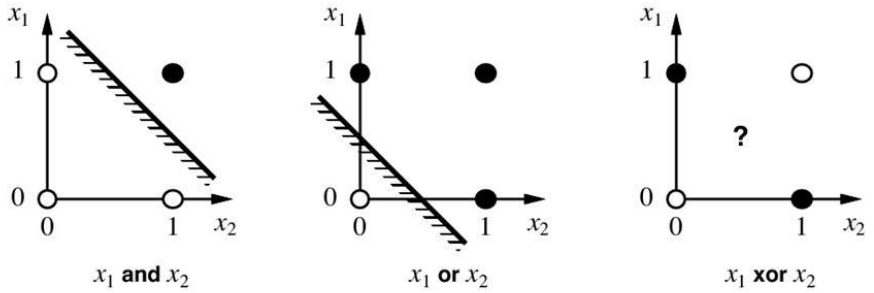
Hard-SVM

Linear separability

$$\text{minimize}_{w,b}\ \frac{1}{2}\|w\|_2^2$$

subject to

$$y_i(w^T x_i + b) \geq 1, i = 1, \ldots, m$$



Hard SVM only work for linear separable problems

Example of non-separability: xor function (we put binary variables in real space).

Soft-SVM

Add slack nonnegative variables allow to violate some restrictions when we optimize Soft-SVM and allow somehow overcome separability problem.

Using $\max(0, 1 - y_i(w^T x_i + b)) \rightarrow y_i(w^T x_i + b) \geq 1 - \xi$

We have

$$\text{minimize}_{w,b} \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{m} \xi_i \qquad C \geq 0$$

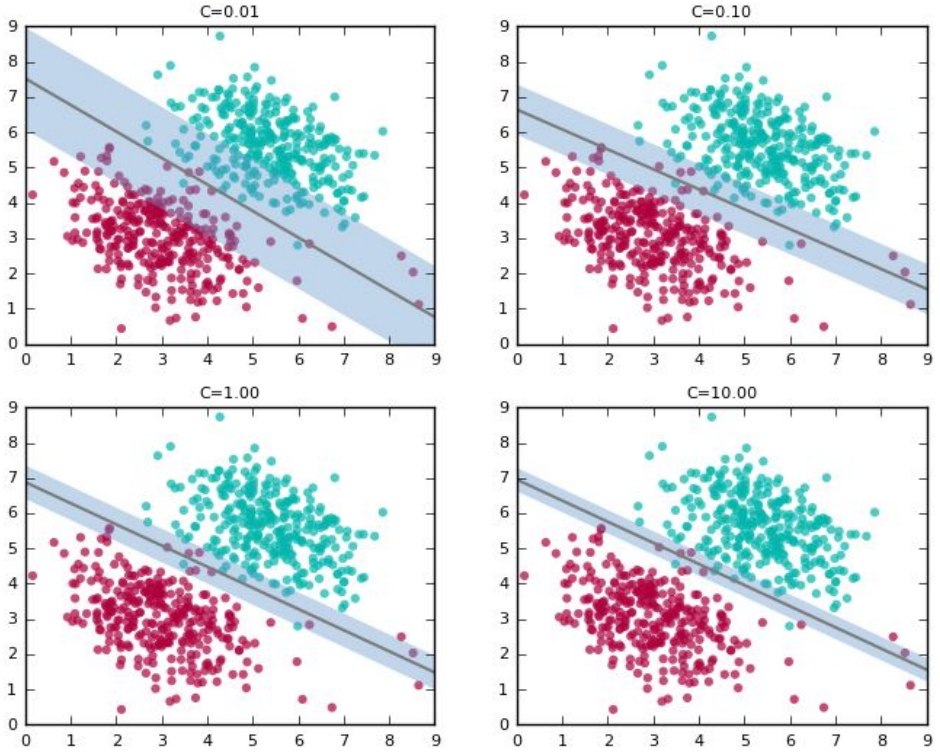
subject to

$$y_i(w^T x_i + b) \geq 1 - \xi_i, i = 1, \ldots, m \quad \xi \geq 0$$

Soft-SVM

When we increase C more ξ to 0's,

SVM becomes more sensitive to data



Reminder: Lagrange Duality

From standard formulation of nonlinear programming problem:

$$\text{minimize } f_0(x)$$

$$\text{subject to } f_i(x) \leq 0, i \in \{1, \ldots, m\}$$

$$h_i(x) = 0, i \in \{1, \ldots, p\}$$

We go to *Lagrangian* form using Karush-Kuhn-Tucker conditions

$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x)$

$\lambda \geq 0$

$\lambda \in \mathbb{R}$ and $\nu \in \mathbb{R}$ are called the dual variables or Lagrange multipliers

Reminder: Lagrange Duality

Form a *dual function*

$$g(\lambda, \nu) = \inf_x \mathcal{L}(x, \lambda, \nu) = \inf_x \left\{ f_0(x) + \sum_{i=1}^{k} \lambda_i f_i(x) + \sum_{j=1}^{l} \nu_j h_j(x) \right\}$$

Then *original* problem is equivalent to

$$\underset{x}{\text{minimize}} \left[\sup_{\lambda \succeq 0, \nu} \mathcal{L}(x, \lambda, \nu) \right]$$

And *dual* problem is switching the min and max:

$$\underset{\lambda \succeq 0, \nu}{\text{maximize}} \left[\inf_x \mathcal{L}(x, \lambda, \nu) \right]$$

Reminder: Lagrange Duality

Dual function properties:

- Concave, regardless of convexity of the primal
- Lower bound on primal

Dual function often used for:

- Creating more advanced algorithms (e.g., primal-dual algorithms)
- Certificate of optimality (**duality gap**)

Reminder: Lagrange Duality

Lemma (Weak Duality)

If $\lambda \geq 0$, then $g(\lambda, \nu) \leq f_0(x^*)$.

Proof.

We have $g(\lambda, \nu) = \inf_x \mathcal{L}(x, \lambda, \nu) \leq \mathcal{L}(x^*, \lambda, \nu) \leq \sup_{\lambda \geq 0, \nu} \mathcal{L}(x^*, \lambda, \nu)$ □

The difference between primal infimum and dual supremum called ***duality gap***.

If duality gap is equal to 0 than holds **strong duality** (e.g., SVM problem).

Soft SVM via Lagrangian function

We can formulate our constrained problem via Lagrange multipliers

$$L(w, b, \xi_i, \lambda, \nu) = \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^m \xi_i - \sum_{i=1}^m \lambda_i [y_i(w^T x_i + b) - 1 + \xi_i] - \sum_{i=1}^m \nu_i \xi_i$$

where $\lambda \in \mathbb{R}_+^m$ and $\nu \in \mathbb{R}_+^m$ are associated Lagrange multipliers

$$\frac{\partial L(w,b,\xi_i,\lambda,\nu)}{\partial w} = 0 \Longrightarrow w = \sum_{i=1}^m \lambda_i y_i x_i$$

$$\frac{\partial L(w,b,\xi_i,\lambda,\nu)}{\partial b} = 0 \Longrightarrow \sum_{i=1}^m \lambda_i y_i = 0$$

$$\frac{\partial L(w,b,\xi_i,\lambda,\nu)}{\partial \xi_i} = 0 \Longrightarrow C - \lambda_i - \nu_i = 0$$

Soft SVM in Dual Setting

After some algebraic manipulation we get dual problem:

$$\text{maximize}_{\lambda} \sum_{i=1}^{m} \lambda_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \lambda_i \lambda_j x_i^T x_j$$

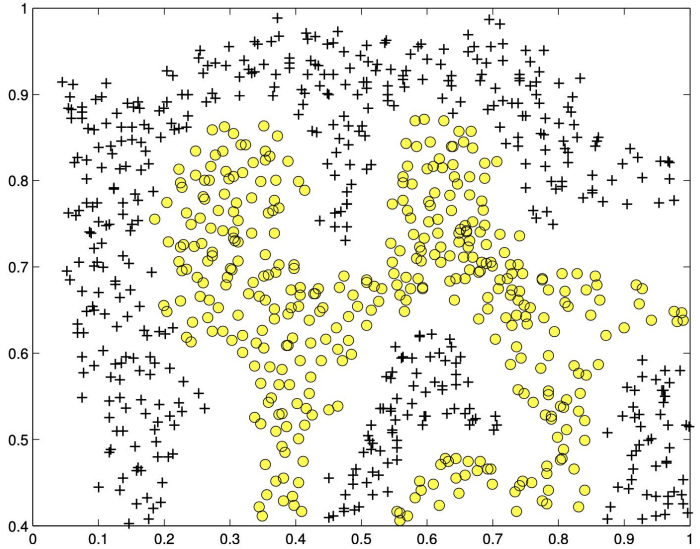
$$\text{subject to } 0 \leq \lambda_i \leq C$$

$$\sum_{i=1}^{m} \lambda_i y_i = 0$$

From Complementary Slackness (KKT) we get when $0 < \lambda_i < C$ then $w^T x_i + b = y_i$ and x_i called *support vector*. Support vector are points in which classifier achieve their worst quality, thereby, SVM depends only on hardest points for classification. Number of this points significantly less than overall number of data points.

Generalization of SVM: Nonlinear approximation via Kernels

- Unfortunately, SVM efficiently works with linear separable data. If our dataset generate from nonlinear function then quality can be not enough. One way to avoid this problem is to use Neural Network. However, NN can not be efficiently learnable.



Generalization of SVM: Nonlinear approximation via Kernels

Another way is to use special map from data space to infinite dimensional Hilbert space. Following map can be done through nonlinear dot products (Kernels):

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

Main property is positive semidefiniteness:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} K(x_i, x_j) \lambda_i \lambda_j \geq 0$$

General +: Kernels depends only on data points x so it doesn't affect optimization.

SVM +: Due to dependence of SVM on support vectors, number of useful x can be lower than full dataset

Examples of kernels

General form

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

$K(x_i, x_j) = x_i^T x_j$ (Linear)

$K(x_i, x_j) = (a x_i^T x_j + b)^d$ (Polynomial)

$K(x_i, x_j) = \exp\left(\frac{-\|x_i - x_j\|_2^2}{\sigma^2}\right)$ (Radial-basis function)

Kernel trick for SVM

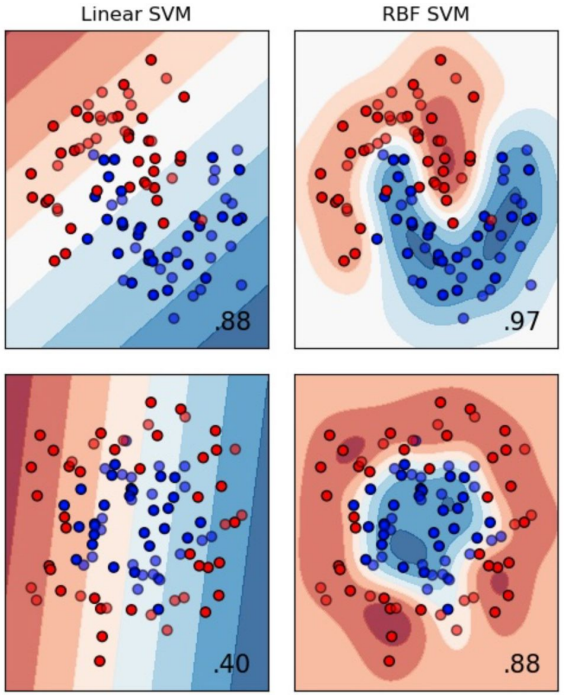
We using dual form of SVM

$$\sum_{i=1}^{m} \lambda_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \lambda_i \lambda_j \underline{x_i^T x_j}$$

And simple plug kernel

$$\sum_{i=1}^{m} \lambda_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \lambda_i \lambda_j \underline{K(x_i, x_j)}$$

Reminder: We optimize only by λ , so, our problem doesn't depends on complexity of K and we optimize only by support vectors!



Multiclass SVM

SVM is inherently binary classifier.

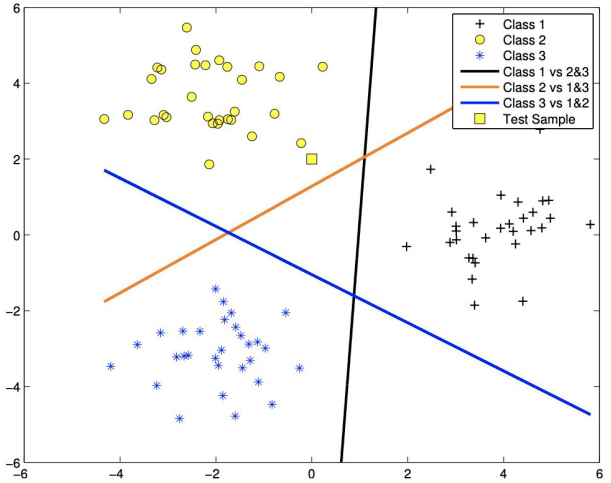
Two ways to avoid this problem:

First way:

- Decompose the multiclass problem to multiple binary classification problems

- Use the majority voting principle or a combined decision from a committee to predict the label

- One-vs-Rest or One-vs-One



Multiclass SVM via pairwise difference

Second way to write SVM objective function for multiclass classification via pairwise difference between SVM weights (Useful for ranking)

$$\text{minimize}_{w,\xi} \ \frac{1}{2}\sum_{k=1}^{K} \|w_k\|_2 + C\sum_{i=1}^{m} \xi_i$$

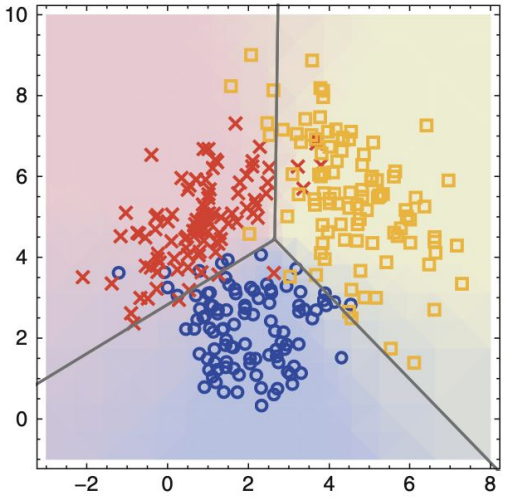
$$\text{subject to}_{w,\xi} \ w_{y_i}^T x_i - w_k^T x_i \geq e_i^k - \xi_i, i = 1, \ldots, m, \ k = 1, \ldots, K$$

where

$$e_i^k = \begin{cases} 0 & \text{for } y_i = k \\ 1 & \text{for } y_i \neq k \end{cases}$$

Classification function

$$\arg\max_k \ w_k^T x$$



Multiclass SVM: Deriving dual formulation

$$\text{minimize}_{w,\xi}\ L(w,\xi) = \frac{1}{2}\sum_{k=1}^{K}\|w_k\|_2 + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\sum_{k=1}^{K}\lambda_{i,k}(w_{yi}^T x_i - w_k x_i - e_i^k + \xi)$$

$$\text{subject to}_{w,\xi}\ \lambda_{i,k} \geq 0, \forall i,k$$

$$e_i^k = \begin{cases} 0 & \text{for } y_i = k \\ 1 & \text{for } y_i \neq k \end{cases}$$

$$\frac{\partial L(w,\xi,\lambda)}{\partial \xi} = C - \sum_{k=1}^{K}\lambda_{i,k} = 0 \Longrightarrow \sum_{k=1}^{K}\lambda_{i,k} = C$$

$$\frac{\partial L(w,\xi,\lambda)}{\partial w_k} = w_k - \sum_{i=1}^{m}\sum_{k=1}^{K}\lambda_{i,k}(1-e_i^k)x_i + \sum_{i=1}^{m}\lambda_{i,k}x_i = x_k - C\sum_{i=1}^{m}(1-e_i^k)x_i + \sum_{i=1}^{m}\lambda_{i,k}x_i = 0$$

Dual multiclass SVM

Finally, after some algebraic combinations we have:

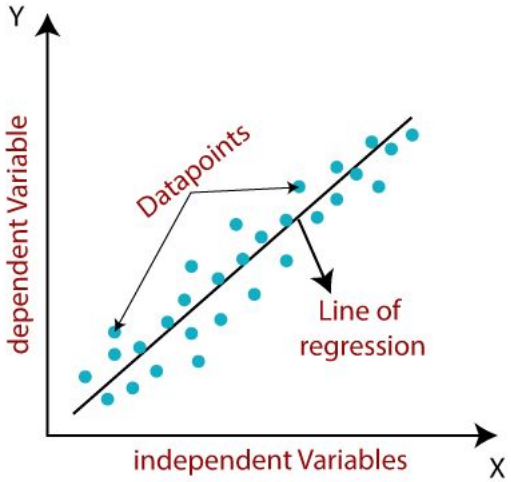
$$\text{maximize}_\lambda \; -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} x_j^T x_i \sum_{k=1}^{K}(C - Ce_i^k - \lambda_{i,k})(C - Ce_j^k - \lambda_{j,k}) + \sum_{i=1}^{m}\sum_{k=1}^{K}\lambda_{i,k}e_i^k$$

$$\text{subject to } \lambda_{i,k} \geq 0, \forall i, k$$

Regression

- We need to fit curve (in our lecture simple line) to real valued outcomes using several statistical assumption about data.

- In this case both x and y are defined in continuous space.



$$\ell_{\mathrm{sq}}(h,(x,y)) \stackrel{\mathrm{def}}{=} (h(x)-y)^2.$$

$$\mathcal{X} \in \mathbb{R} \qquad \mathcal{Y} \in \mathbb{R}$$

Ordinary Least Squares

Minimize

$$\ell(h, (x, y)) = (y - h(x))^2 = (y - Ax)^2$$

It is strictly convex optimization problem (A is weight matrix)

Normal equation

$$A^T Ax = A^T y$$

If A is full column rank matrix, we have:

$$x^* = (A^T A)^{-1} A^T y$$

Support Vector Regression

SVR defined a special ϵ—Insensitive Loss Function to avoid problem with ϵ— noisy observations.

The problem can be written as a convex optimization problem

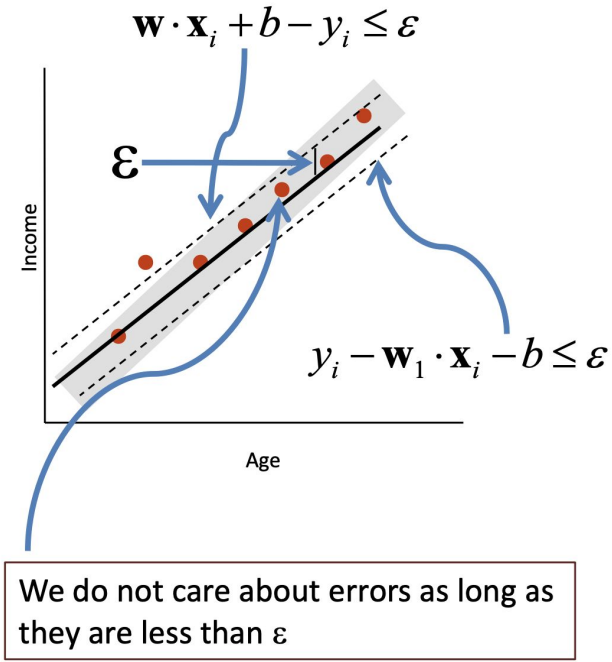
$$\text{minimize}_w \frac{1}{2}\|w\|_2^2$$

$$\text{subject to}$$

$$y_i - w^T x - b \leq \epsilon$$

$$w^T x + b - y_i \leq \epsilon$$

$$\ell_\epsilon(h,(x,y)) = \max(|y - w^T x - b| - \epsilon, 0)$$



Support Vector Regression via QP

- Primal problem

$$\text{minimize}_w \ \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^m (\xi + \xi^*)$$

$$\text{subject to}$$

$$y_i - w^T x - b \leq \epsilon + \xi_i$$

$$w^T x + b - y_i \leq \epsilon + \xi_i^*$$

$$\xi_i, \xi_i^* \geq 0, i = 1, \ldots, m$$

- Dual problem

$$\text{maximize } \frac{1}{2}\sum_{i,j=1}^m (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) x_i^T x_j - \epsilon\sum_{i=1}^m (\alpha_i - \alpha_i^*) + \sum_{i=1}^m y_i(\alpha_i - \alpha_i^*)$$

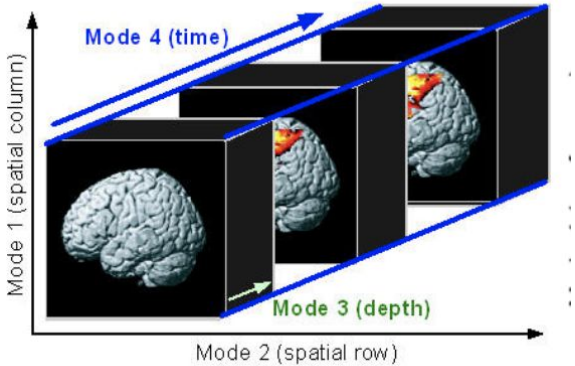
$$\text{subject to}$$

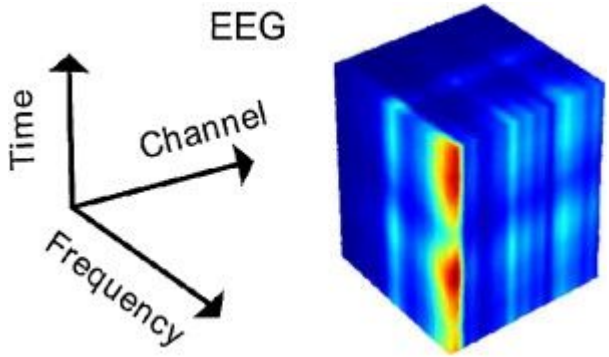
$$\sum_{i=1}^m (\alpha_i - \alpha_i^*) = 0, \ 0 \leq \alpha_i, \alpha_i^* \leq C$$

From vectors to arbitrary tensors

- Practical application from biomedicine
  - Classification of fMRI (time x 3D volume indexed by cartesian coordinate) and EEG (time x frequency x electrodes)

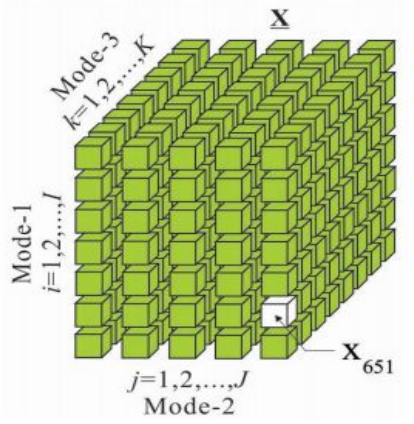
  - Ordinary SVM for this type of data can be very restrictive. We should somehow use it tensor structure (multidimensional structure)





Brief overview of tensors

Tensor is multilinear form. Higher-order generalization of vector and matrices.



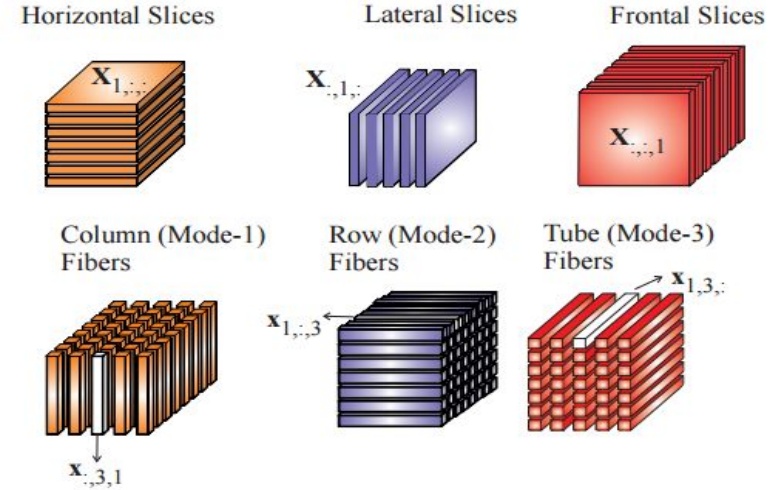


Figure 2: A 3rd-order tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I \times J \times K}$, with entries $x_{i,j,k} =$

CPD format and Tensor Rank

Basic tensor operation is tensor or outer product. Suppose $\underline{\mathbf{X}}$ is n-order tensor

$$\underline{\mathbf{X}} = a_{i_1}^{(1)} \circ a_{i_2}^{(2)} \cdots \circ a_{i_N}^{(N)}$$

The symbol "∘" represents the vector outer product. This means that each element of the tensor is the product of the corresponding vector elements:

$$x_{i_1 i_2 \cdots i_N} = a_{i_1}^{(1)} a_{i_2}^{(2)} \cdots a_{i_N}^{(N)} \quad \text{for all } 1 \leq i_n \leq I_n.$$

CPD or Canonical Polyadic Decomposition (generalization of SVD) of

$$\underline{\mathbf{X}} = \sum_{r=1}^{R} \lambda_r u_1^{(1)} \circ u_2^{(2)} \circ \cdots \circ u_r^{(n)}$$

Where R - rank of Tensor

Basic operation

N-mode matricization of n-order tensor is reshaping of tensor to matrix by n-th dimension $\mathbf{X}_{(n)}$

n - Mode product of Tensor and vector:

$$\underline{\mathbf{X}} \bar{\times}_n \mathbf{v} = \mathbf{X}_{(n)} \mathbf{v}$$

Suppose $\underline{\mathbf{X}} = \sum_{r=1}^{R} \lambda_r u_1^{(1)} \circ u_2^{(2)} \circ \cdots \circ u_r^{(n)}$ than following holds

$$\mathbf{X}_{(n)} = \mathbf{U}^{(n)} (\mathbf{U}^{(N)} \odot \cdots \odot \mathbf{U}^{(n+1)} \odot \mathbf{U}^{(n-1)} \odot \cdots \odot \mathbf{U}^{(1)})^T$$

Where by $\odot$ we denote Khatri-Rao product

General idea of tensor methods

Main idea is to impose low-rank structure in the model. It's well known result that lower convex envelope for a matrix rank is nuclear norm

$$\|X\|_* = \sup\{g(\mathbf{X}) | g \text{ is convex and } g(\mathbf{X}) \leq \text{rank}(\mathbf{X})\}$$

Despite it's a lower bound of a rank, the nuclear norm and rank is significantly related thus

$$\min \text{rank}(\mathbf{X}) \approx \min \|\mathbf{X}\|_*$$

where

$$||\mathbf{X}||_* = \sum_{r=1}^{R} \sigma_r$$

Support Matrix Machine

SMM - **Support Matrix Machines**

This method suitable for matrix data like EEG, ...

For matrices we use frobenius inner product:

$$\langle \mathrm{vec}(\mathbf{W}), \mathrm{vec}(\mathbf{W}) \rangle = \mathrm{tr}\left(\mathbf{W}^T \mathbf{W}\right)$$

Final problem formulation equivalent to original soft-margin SVM + nuclear norm of weights:

$$\begin{aligned} &\arg\min_{\mathbf{W},b,\xi_m} \ \frac{1}{2}\mathrm{tr}(\mathbf{W}^{\mathrm{T}}\mathbf{W}) + \tau\|\mathbf{W}\|_* + C\sum_{m=1}^{m}\xi_m \\ &\text{s.t.} \quad y_m(\mathrm{tr}(\mathbf{W}^{\mathrm{T}}\mathbf{X}_m) + b) \geqslant 1 - \xi_m. \end{aligned}$$

Support Tensor Machine

Our first generalization SVM to Tensor models

Problem formulation:

$$\min_{\mathbf{w}_n, b, \boldsymbol{\xi}} \; J(\mathbf{w}_n, b, \boldsymbol{\xi}) = \frac{1}{2} \left\| \otimes_{n=1}^{N} \mathbf{w}_n \right\|^2 + C \sum_{m=1}^{M} \xi_m$$

$$\text{s.t.} \quad y_m \left(\underline{\mathbf{X}}_m \bar{\times}_1 \mathbf{w}_1 \cdots \bar{\times}_N \mathbf{w}_N + b \right) \geqslant 1 - \xi_m, \quad \boldsymbol{\xi} \geqslant 0,$$
$$m = 1, \ldots, M.$$

Geometrically STM tries to find N different hyperplanes in N mode spaces

STM Fitting

The STM problem fitted using similar alternating strategy as for other tensor model.

We decompose our original problem by N subproblem and solve it sequentially one by one.

The n-th QP sub-problem formulated, as follows

$$\min_{\mathbf{w}_n, b, \boldsymbol{\xi}} \ \frac{1}{2} \|\mathbf{w}_n\|^2 \prod_{1 \leqslant i \leqslant N}^{i \neq n} (\|\mathbf{w}_i\|^2) + C \sum_{m=1}^{M} \xi_m$$

$$\text{s.t.} \quad y_m \left(\mathbf{w}_n^{\mathrm{T}} \left(\underline{\mathbf{X}}_m \bar{\times}_{i \neq n} \mathbf{w}_i \right) + b \right) \geqslant 1 - \xi_m, \quad \boldsymbol{\xi} \geqslant 0,$$

$$m = 1, \ldots, M.$$

HRSTM - Higher Rank STM

Higher Rank STM aim to estimate weights of SVM in CPD format. Construct hyperplane in the tensor space.

Problem formulation:

$$\min_{\underline{\mathbf{W}},b,\boldsymbol{\xi}} \ \frac{1}{2}\langle \underline{\mathbf{W}}, \underline{\mathbf{W}} \rangle + C \sum_{m=1}^{M} \xi_m,$$

$$\text{s.t.} \quad y_m \left(\langle \underline{\mathbf{W}}, \underline{\mathbf{X}}_m \rangle + b \right) \geqslant 1 - \xi_m, \quad \xi_m \geqslant 0, \quad m = 1, \ldots, M,$$

Where

$$\underline{\mathbf{W}} = \sum_{r=1}^{R} \mathbf{u}_r^{(1)} \circ \mathbf{u}_r^{(2)} \circ \cdots \circ \mathbf{u}_r^{(N)}$$

And

$$\langle \underline{\mathbf{W}}, \underline{\mathbf{W}} \rangle = \operatorname{tr}[\mathbf{W}_{(n)} \mathbf{W}_{(n)}^{\mathrm{T}}] = \operatorname{vec}(\mathbf{W}_{(n)})^{\mathrm{T}} \operatorname{vec}(\mathbf{W}_{(n)})$$

HRSTM Fitting via SMM

We use again alternating strategy: At each iteration we solve only for the parameters that are associated with the n-th mode of the parameter tensor, while keeping all the other parameters fixed.

We use following property

$$\mathbf{W}_{(n)} = \mathbf{U}^{(n)}(\mathbf{U}^{(N)} \odot \cdots \odot \mathbf{U}^{(n+1)} \odot \mathbf{U}^{(n-1)} \odot \cdots \odot \mathbf{U}^{(1)})^{\mathrm{T}} = \mathbf{U}^{(n)}(\mathbf{U}^{(-n)})^{\mathrm{T}}$$

Then formulate problem in the SMM way and then each of N subproblems solved via QP:

$$\min_{\mathbf{U}^{(n)},b,\boldsymbol{\xi}} \frac{1}{2}\mathrm{tr}(\mathbf{U}^{(n)}(\mathbf{U}^{(-n)})^{\mathrm{T}}(\mathbf{U}^{(-n)})(\mathbf{U}^{(n)})^{\mathrm{T}}) + C\sum_{m=1}^{M}\xi_m,$$

$$\text{s.t.} \quad y_m\left(\mathrm{tr}\left(\mathbf{U}^{(n)}(\mathbf{U}^{(-n)})^{\mathrm{T}}\mathbf{X}_{m(n)}^{\mathrm{T}}\right) + b\right) \geqslant 1 - \xi_m, \quad \xi_m \geqslant 0,$$

$$m = 1, \ldots, M, \quad n = 1, \ldots, N.$$

HRSTM Fitting via SVM

Additionally we can use SVM (instead of SMM) formulation of Higher Rank STM problem.

For thus we use the following trick

Let

$$\mathbf{A} = \mathbf{U}^{(-n)^{\mathrm{T}}}\mathbf{U}^{(-n)}$$

$$\widetilde{\mathbf{U}}^{(n)} = \mathbf{U}^{(n)}\mathbf{A}^{\frac{1}{2}}$$

Then

$$\begin{aligned}\mathrm{tr}[\mathbf{U}^{(n)}(\mathbf{U}^{(-n)})^{\mathrm{T}}(\mathbf{U}^{(-n)})(\mathbf{U}^{(n)})^{\mathrm{T}}] &= \mathrm{tr}[\widetilde{\mathbf{U}}^{(n)}(\widetilde{\mathbf{U}}^{(n)})^{\mathrm{T}}] \\ &= \mathrm{vec}(\widetilde{\mathbf{U}}^{(n)})^{\mathrm{T}}\mathrm{vec}(\widetilde{\mathbf{U}}^{(n)}).\end{aligned}$$

HRSTM Fitting via SVM

Finally we solve following sequence of SVM tasks for each $\tilde{\mathbf{U}}^{(n)}$

$$\begin{aligned} &\min_{\mathbf{U}^{(n)},b,\boldsymbol{\xi}} \frac{1}{2}\mathrm{vec}(\widetilde{\mathbf{U}}^{(n)})^{\mathrm{T}}\mathrm{vec}(\widetilde{\mathbf{U}}^{(n)}) + C\sum_{m=1}^{M}\xi_m, \\ \text{s.t.} \quad & y_m\left(\mathrm{vec}(\widetilde{\mathbf{U}}^{(n)})^{\mathrm{T}}\mathrm{vec}(\widetilde{\mathbf{X}}_{m(n)}) + b\right) \geqslant 1-\xi_m, \quad \xi_m \geqslant 0, \\ & m = 1,\ldots,M, \quad n = 1,\ldots,N. \end{aligned}$$

And after each $\tilde{\mathbf{U}}^{(n)}$ update we go to original parameters via inverse transform

$$\mathbf{U}^{(n)} = \widetilde{\mathbf{U}}^{(n)}\mathbf{A}^{-\frac{1}{2}}$$

Kernel function for tensors

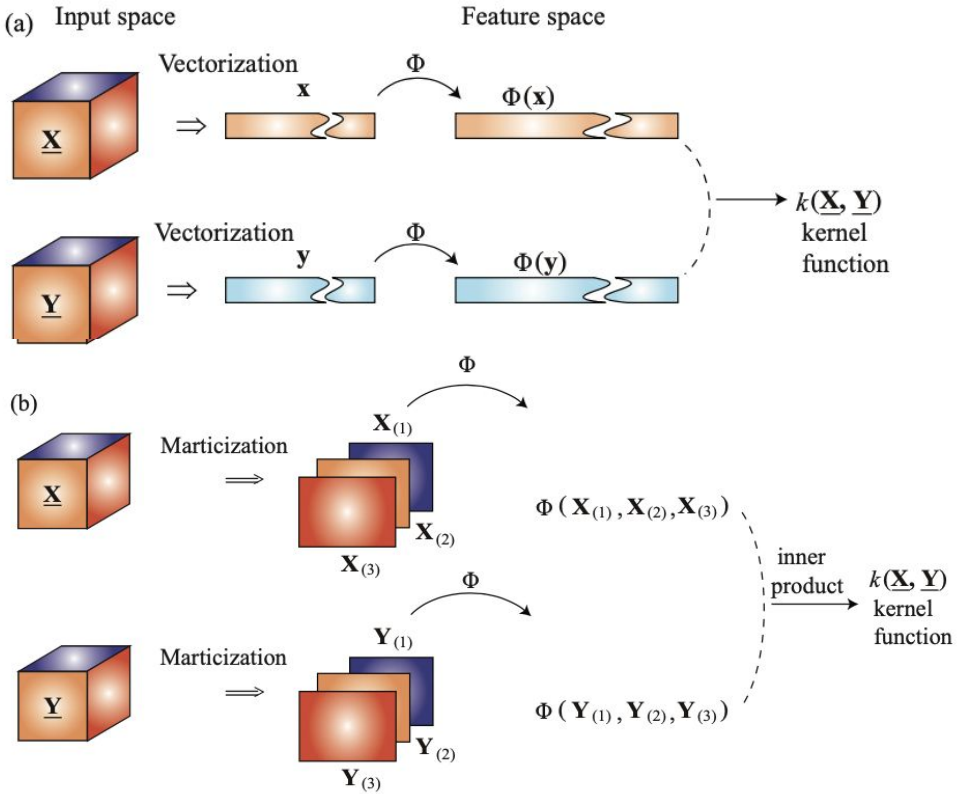
Similar to SVM in SMM and STM

we can use kernel functions:

kernel functions $k : \underline{\mathbf{X}} \times \underline{\mathbf{X}} \to \mathbb{R}$, given by

Linear kernel: $k(\underline{\mathbf{X}}, \underline{\mathbf{X}}') = \langle \text{vec}(\underline{\mathbf{X}}), \text{vec}(\underline{\mathbf{X}}') \rangle$,

Gaussian-RBF kernel: $k(\underline{\mathbf{X}}, \underline{\mathbf{X}}') = \exp\left(-\frac{1}{2\beta^2}\|\underline{\mathbf{X}} - \underline{\mathbf{X}}'\|_F^2\right)$



Unsupervised

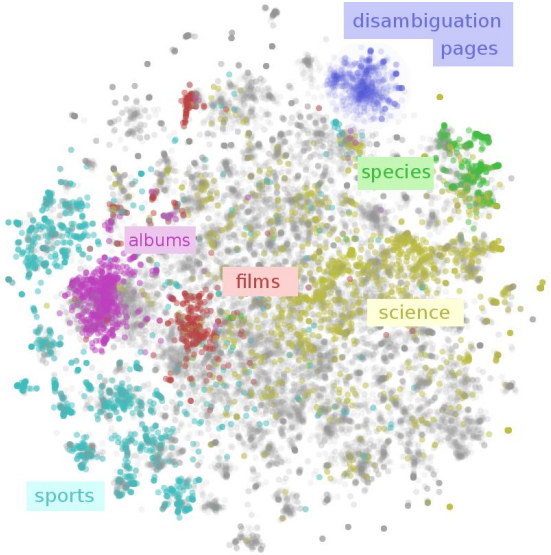
Unsupervised learning

Unsupervised learning generally about *inverse* problems.

We fit our models to predict data by labels which is inverse problem to supervised.

Examples:

- Clustering
- Probability density estimation
- Representation Learning
- Dimensional reduction
- Factor analysis



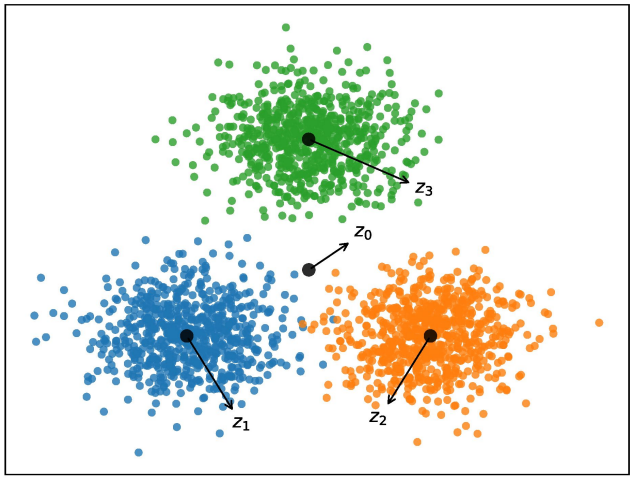
Clustering

Main idea of the task (non-formal):

Group set of objects in a such way that objects

in the same group (called a cluster) are more similar

to each than to those in other groups (clusters)



k-Means Clustering

k-means clustering:

- randomly select k cluster centroids $\mu_1, \ldots, \mu_k \in \mathbb{R}^n$
- iterate between the following two steps:
  1. assign data to different clusters: for each $j = 1, \ldots, k$,

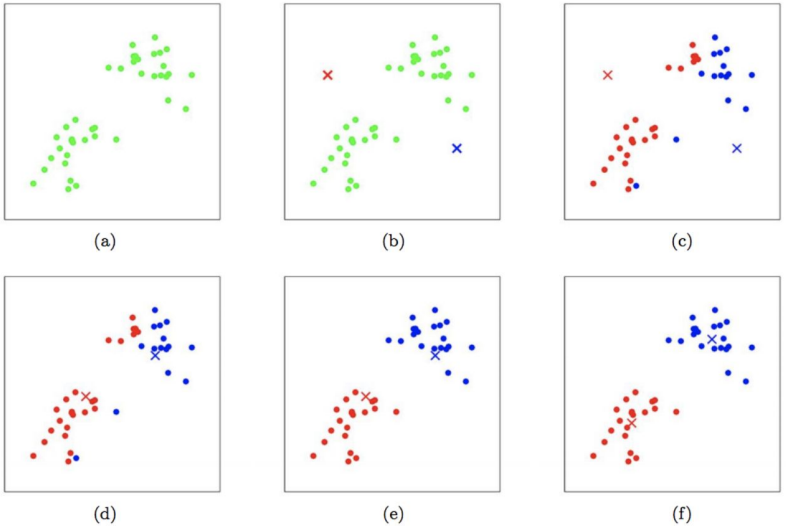
$$C_j = \left\{ i : \|a^{(i)} - \mu_j\| \leq \|a^{(i)} - \mu_{j'}\|, \ \forall j' \right\}$$

  2. update the centers of clusters: for each $j = 1, \ldots, k$,

$$\text{minimize} \quad \sum_{i \in C_j} \|a^{(i)} - \mu_j\|$$

with optimization variable μ_j

k-Means Clustering



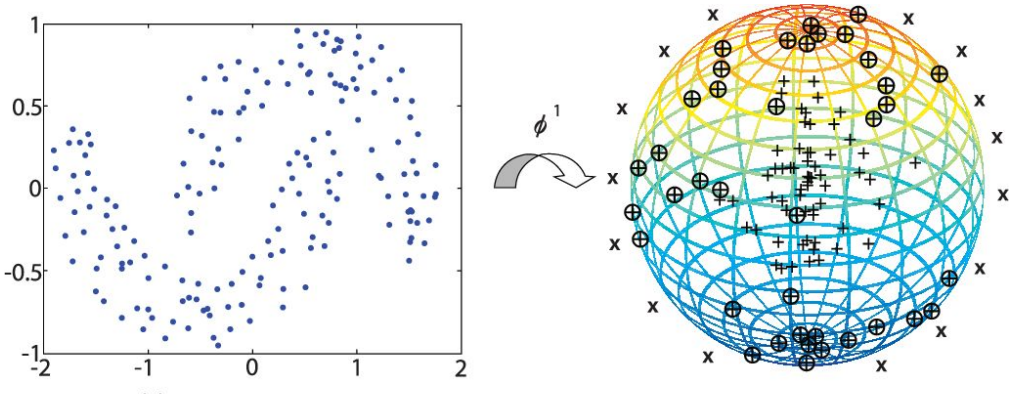
- (a), (b): data and initial guess
- (c), (d): first iteration
- (e), (f): second iteration

Support Vector Clustering

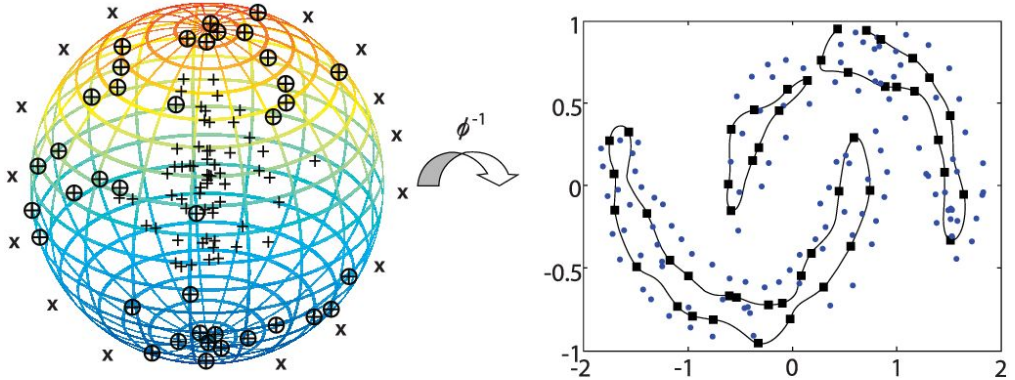
Find radius (R) and center of sphere (a)

Given this constraints:

$$\|\Phi(x_j) - a\|_2^2 \leq R^2, \forall j$$



(a) Projection from Original Data Space to Higher-Dimensional Space



(b) Inverse Projection from Enclosing Sphere to Cluster Contours

Lagrangian of Support Vector Clustering

$$L(R, a, \xi, \beta, \mu) = R^2 - \sum_j (R^2 + \xi_j - \|\Phi(x_j) - a\|^2)\beta_j - \sum_j \xi_j \mu_j + C \sum_j \xi_j$$

$$\text{subject to } \beta_j \geq 0, \mu_j \geq 0, \xi \geq 0$$

Setting to zero the derivative of L with respect to R a and ξ_j, respectively, leads to

$$\sum_j \beta_j = 1$$

$$a = \sum_j \beta_j \Phi(x_j)$$

$$\beta_j = C - \mu_j$$

Lagrangian of Support Vector Clustering

And also we have KKT complementarity condition result in

$$\xi_j \mu_j = 0,$$
$$(R^2 + \xi_j - ||\Phi(\mathbf{x}_j) - \mathbf{a}||^2)\beta_j = 0.$$

The image of a point $\boldsymbol{x_i}$ with $\xi_i > 0$ and $\beta_i > 0$ lies outside the feature space and called a *bounded support vector* or BSV.

If $\xi_i = 0$ and $0 < \beta_i < C$ then $\boldsymbol{x_i}$ is mapped to the surface of the feature space sphere and called *support vector.*

Other points lie inside the sphere.

Kernel of Support Vector Clustering

Using previous relations we may eliminate the variables R, a and μ_j

And we can turning the Lagrangian into the Wolfe dual form that is a function of the variables β_j :

$$\sum_j \Phi(x_j)^2 \beta_j - \sum_{i,j} \beta_i \beta_j \Phi(x_i)^T \Phi(x_j)$$

Using kernels we transform Lagrangian to this:

$$\sum_j K(\mathbf{x}_j, \mathbf{x}_j)\beta_j - \sum_{i,j} \beta_i \beta_j K(\mathbf{x}_i, \mathbf{x}_j).$$

And using Gaussian kernel $K(x_i, x_j) = \exp(-q\|x_i - x_j\|^2)$

Support Vector Clustering

Let's define the distance equation from center of the sphere to the point:

$$R^2(\mathbf{x}) = K(\mathbf{x}, \mathbf{x}) - 2\sum_j \beta_j K(\mathbf{x}_j, \mathbf{x}) + \sum_{i,j} \beta_i \beta_j K(\mathbf{x}_i, \mathbf{x}_j)$$

Radius of the sphere is

$$R = \{R(\mathbf{x}_i) \mid \mathbf{x}_i \text{ is a support vector } \}$$

Because it is the same for all support vectors.

And we have to 2 hyperparameters q inside gaussian kernel K and C

Cluster Assignment

Observation: “given a pair of data points that belong to different components (clusters), any path that connects them must exit from the sphere in feature space”

Calculate adjacency matrix

$$A_{ij} = \begin{cases} 1 & \text{if, for all } \mathbf{y} \text{ on the line segment connecting } \mathbf{x}_i \text{ and } \mathbf{x}_j,\ R(\mathbf{y}) \leq R \\ 0 & \text{otherwise.} \end{cases}$$

Checking the line segment is implemented by sampling a number of points (20 points used in original paper).

Calculate connected components of the graph induced by A.

Examples

SVC is divisive “clustering” algorithm.

We start from small q and then increasing it.

Initial q can be:

$$q = \frac{1}{\max_{i,j} ||\mathbf{x}_i - \mathbf{x}_j||^2}$$

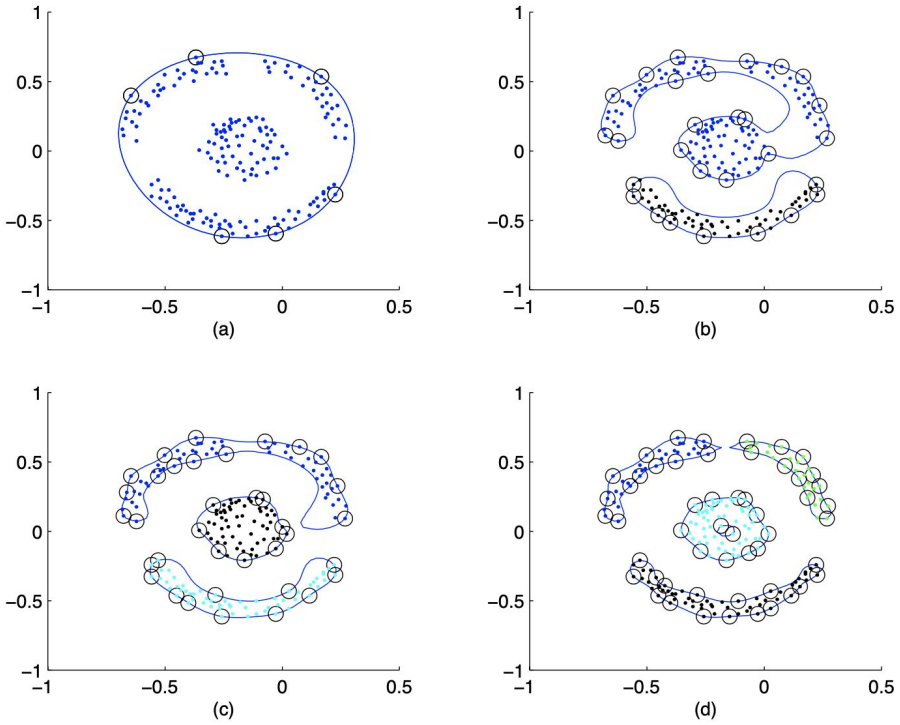


Figure 1: Clustering of a data set containing 183 points using SVC with $C = 1$. Support vectors are designated by small circles, and cluster assignments are represented by different grey scales of the data points. (a): $q = 1$ (b): $q = 20$ (c): $q = 24$ (d): $q = 48$.

Conclusion

- Convex optimization is used in ML for deriving efficient learning algorithms
- General trick:

  if we have some non-convex loss function then we upperbound it via its convex surrogate (if it exist)

Let’s practise

https://colab.research.google.com/drive/1-uXYb11yw-D7UBj3qnmhDRMBd5dqUPRA

Shorten for view:
https://bit.ly/2PjDMst

Shorten for edit:
https://bit.ly/2Ypk42s

Thank you for your attention! :)