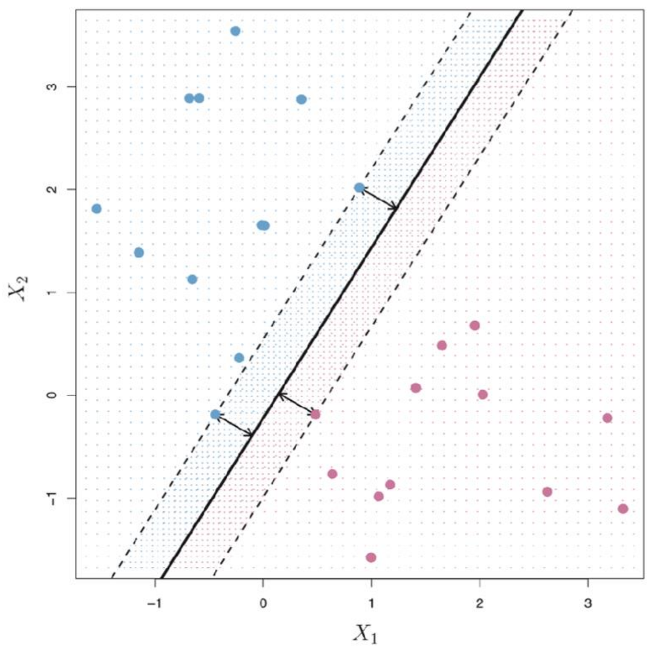
CSCE 421: Machine Learning

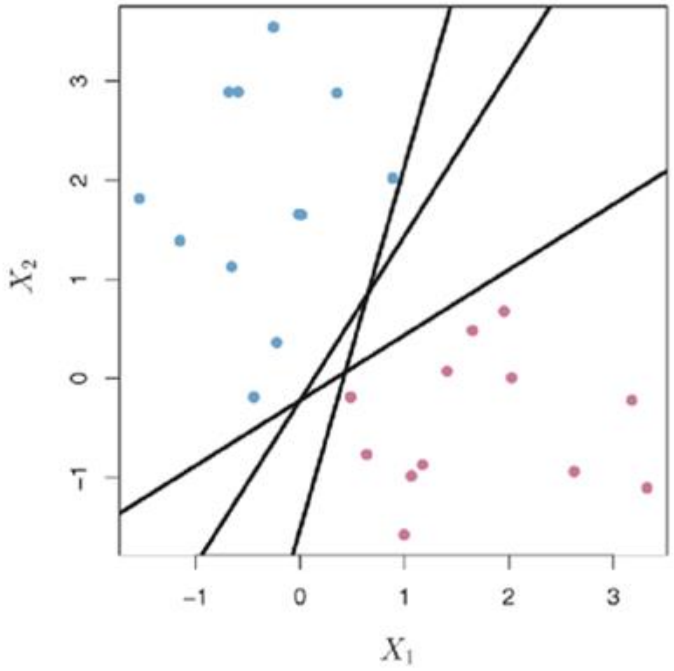
Lecture 14: SVM Optimization

Texas A&M University
CSCE 421
Bobak Mortazavi
Ryan King
Zhale Nowroozilarki

Review

1. Which of the decision boundaries would you use from the plot on the right and why?
2. In the plot on the left, what are the dashed lines called?
3. In the plot of the left what are the dots on the dashed line called?
4. How can we modify the Maximum Margin Classifier for data that isn't linearly separable?





Understanding the Loss Function

- Consider the margin boundary $y_i f(x) = y_i(w_0 + \sum_{j=1}^{D} w_j\, \phi(x_{ij}))$

Understanding the Loss Function

- Consider the margin boundary $y_i f(x) = y_i(w_0 + \sum_{j=1}^{D} w_j \, \phi(x_{ij}))$
- Why do we multiply y_i by $f(x)$?

Understanding the Loss Function

- Consider the margin boundary $y_i f(x) = y_i(w_0 + \sum_{j=1}^{D} w_j \, \phi(x_{ij}))$
- Why do we multiply y_i by $f(x)$?
- Can we somehow relate SVM's margin boundary to Regression's Loss Functions?

Understanding the Loss Function

- Consider the margin boundary $y_i f(x) = y_i(w_0 + \sum_{j=1}^{D} w_j\, \phi(x_{ij}))$
- Why do we multiply y_i by $f(x)$?
- Can we somehow relate SVM's margin boundary to Regression's Loss Functions?
- Recall $y - \hat{y} = y - f(x)$ is our residual (error)

Classification Rule

- The classification rule for SVM is $G(x) = \mathrm{sign}(f(x))$

Classification Rule

- The classification rule for SVM is $G(x) = \text{sign}(f(x))$
- Now, how do we classify error?

Classification Rule

- The classification rule for SVM is $G(x) = \text{sign}(f(x))$
- Now, how do we classify error?
- Recall $y_i \in \{-1, +1\}$

Classification Rule

- The classification rule for SVM is $G(x) = \text{sign}(f(x))$
- Now, how do we classify error?
- Recall $y_i \in \{-1, +1\}$
- So, $y_i G(x_i) > 0$ if samples are classified correctly

0-1 Loss

- The decision boundary as $f(x) = 0$

0-1 Loss

- The decision boundary as $f(x) = 0$
- $L(y, f(x))$ is called the 0-1 loss in this case

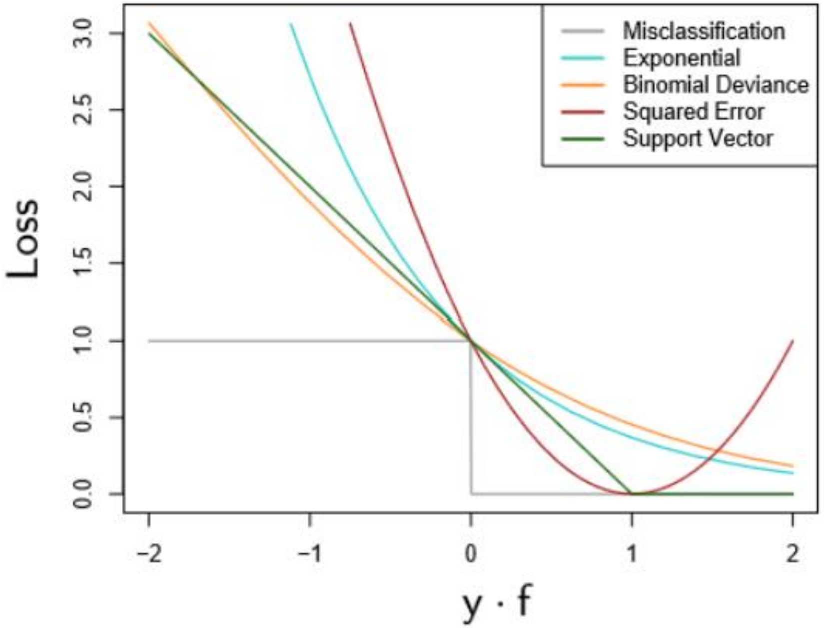
0-1 Loss

- The decision boundary as $f(x) = 0$
- $L(y, f(x))$ is called the 0-1 loss in this case
- $L\big(y, f(x)\big) = \sum_{i=1}^{N} I(y_i f(x_i) < 0)$

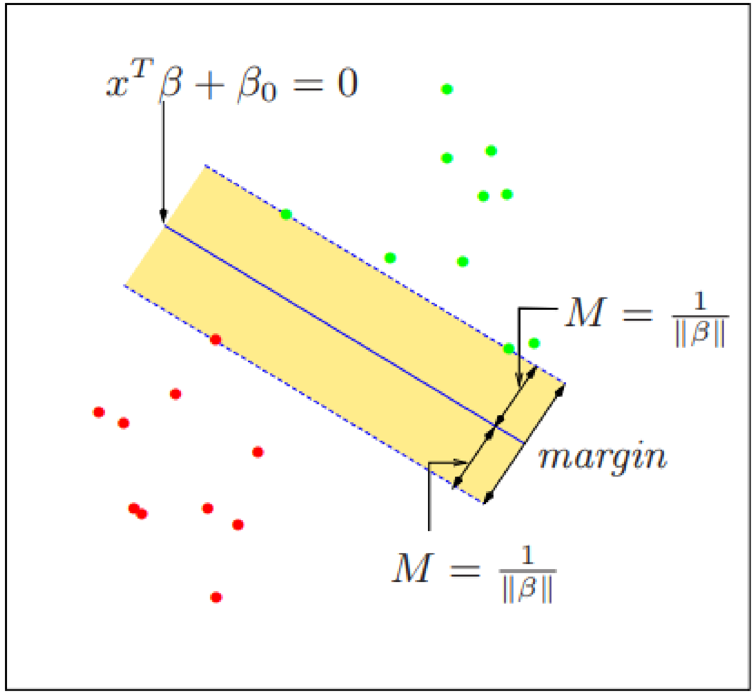
Support Vector Machine: Hinge Loss

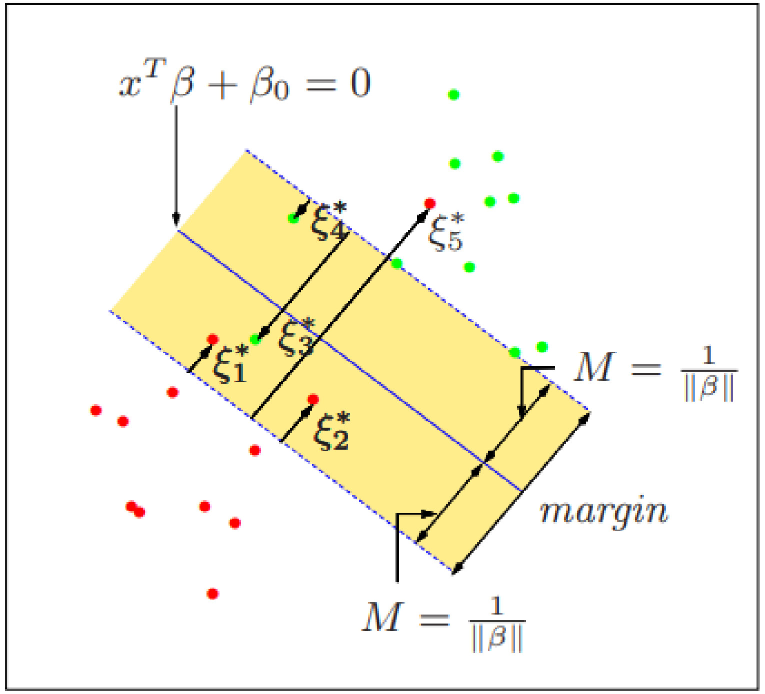
$$L(x, y, w) = \sum_{i=1}^{N} \max(0, 1 - y_i(w_0 + w_1 x_{i1} + \cdots + w_D x_{iD}))$$

- Instead of the common loss for logistic regression



Optimal Hyperplane and Support Vectors





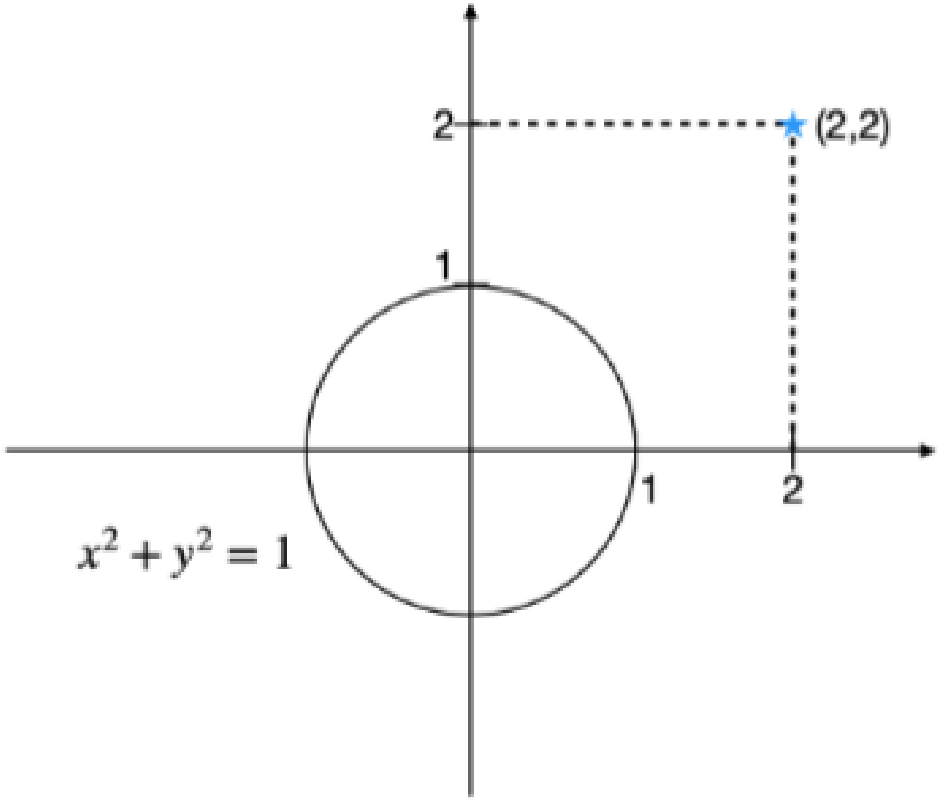
- Margin of Separation M: distance between the separating hyperplane and the closest input point
- Support Vectors: input points closest to the separating hyperplane

Mathematical Aside: Lagrange Multipliers

- Turn a constrained optimization problem into an unconstrained optimization problem by absorbing the constraints into a cost function, weighted by the Lagrange multipliers
- Example: Find point on the circle $x^2 + y^2 = 1$ closest to the point (2,2)
  – Minimize $F(x, y) = (x - 2)^2 + (y - 2)^2$
  – Subject to the constraint $x^2 + y^2 - 1 = 0$
  – Absorb the constraint into the cost function, after multiplying the Lagrange multiplier α:

$$F(x, y, \alpha) = (x - 2)^2 + (y - 2)^2 + \alpha(x^2 + y^2 - 1)$$

Mathematical Aside: Langrange Multipliers



Mathematical Aside: Lagrange Multipliers

- Formulate Lagrangian (primal problem):
- $F(x, y, \alpha) = (x-2)^2 + (y-2)^2 + \alpha(x^2 + y^2 - 1)$
- The optimization problem becomes:

$$\frac{\partial F}{\partial x} = 2(x-2) + 2\alpha \mathrm{x} = 0 \rightarrow x = \frac{2}{1+\alpha}$$

$$\frac{\partial F}{\partial y} = 2(y-2) + 2\alpha \mathrm{y} = 0 \rightarrow y = \frac{2}{1+\alpha}$$

- We substitute x,y in the Lagrangian and express it in terms of its dual form wrt α and maximize it

$$\frac{\partial F}{\partial x} = x^2 + y^2 - 1 = 0 \rightarrow \left(\frac{2}{1+\alpha}\right)^2 + \left(\frac{2}{1+\alpha}\right)^2 = 1 \rightarrow \alpha = 2\sqrt{2} - 1$$

- Recover the solution: $(x, y) = \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$

Mathematical Aside: Lagrange Multipliers

- Exercise
  - Find point on the circle $x^2 + y^2 = 1$ closest to the point (-3,3)

Primal Problem: Constrained Optimization

- For the training set $D^{train} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$ find w and w_0 such that they minimize the inverse separation margin $\left(\frac{1}{M} = \frac{\|w\|}{2}\right)$ while satisfying a constraint (all examples are correctly classified):
  - Cost function $\Phi(w) = \frac{1}{2} w^T w$
  - Constraint: $y_i(w^T x_i + w_0) \geq 1$ for $i = 1,2,\cdots,N$

$$\min_{w} \frac{1}{2} w^T w, \text{ such that (s.t.) } y_i(w^T x_i + w_0) \geq 1 \text{ for } i = 1,2,\cdots,N$$

- This problem can be solved using the method of Lagrange multipliers (see next two slides)

Support Vector Machines: Linearly separable case

$$\min_{w} \frac{1}{2} w^T w, \text{ such that (s.t.) } y_i(w^T x_i + w_0) \geq 1 \text{ for } i = 1,2,\cdots,N$$

1. Formulate Lagrangian function (primal problem)

$$L = \frac{1}{2}\|\boldsymbol{w}\|_2^2 - \sum_{i=1}^{N} \alpha_i\big(y_i(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}_i + w_0) - 1\big)$$

2. Minimize Lagrangian to solve for primal variables w and w_0

$$\frac{\partial L}{\partial \boldsymbol{w}} = 0 \rightarrow \boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i$$

$$\frac{\partial L}{\partial w_0} = 0 \rightarrow 0 = \sum_{i=1}^{N} \alpha_i y_i$$

3. Substitute the primal variables w and w_0 into the Lagrangian and express in terms of dual variables α_i

$$L = \frac{1}{2}\|\boldsymbol{w}\|_2^2 - \boldsymbol{w}^{\boldsymbol{T}} \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i - w_0 \sum_{i=1}^{N} \alpha_i y_i + \sum_{i=1}^{N} \alpha_i$$

$$= -\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \sum_{i=1}^{N} \alpha_i = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} \boldsymbol{x}_i^T \boldsymbol{x}_{i'}$$

Support Vector Machines: Linearly separable case

$$\min_{w} \frac{1}{2} w^T w, \text{ such that (s.t.) } y_i(w^T x_i + w_0) \geq 1 \text{ for } i = 1,2,\cdots,N$$

4. Maximize the Lagrangian with respect to dual variables (dual problem)

$$\max_{\alpha_i} L = \max_{\alpha_i} \left\{ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} \boldsymbol{x}_i^T \boldsymbol{x}_{i'} \right\}$$

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

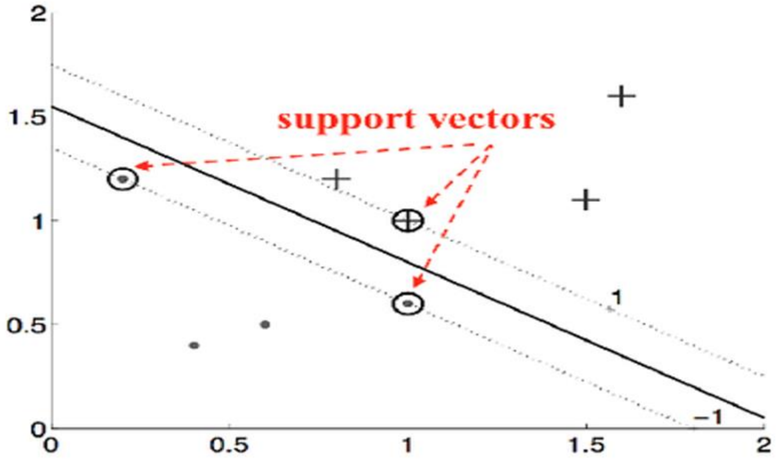
- Solved numerically using quadratic optimization methods
- The dual depends on data size N and no on the data dimensionality D
- Most of the α_i will vanish with $\alpha_i = 0$ only a small percentage
- The set of x_i whose $\alpha_i \neq 0$ are the support vectors

Support Vector Machines: Linearly separable case

$$\min_{w} \frac{1}{2} w^T w, \text{ such that (s.t.) } y_i(w^T x_i + w_0) \geq 1 \text{ for } i = 1,2,\cdots,N$$

5. Recover the solution (for the primal variables) from the dual variables
   - Find w: Substitute α_i from (4) to $w = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i$
   - Find w_0:
     - From $\boldsymbol{w}^T \boldsymbol{x}_i + w_0 = y_i$, where $\boldsymbol{x}_i$ is a support vector, calculate $w_0 = y_i - \boldsymbol{w}^T \boldsymbol{x}_i$
     - For numerical stability, average w_0 values estimated from all support vectors

Support Vector Machines: Linearly separable cases



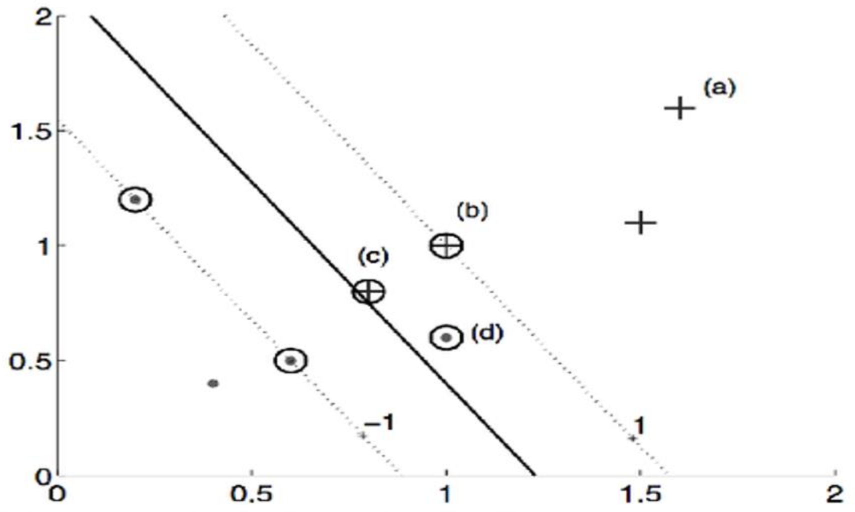
- Sample $\boldsymbol{x}_i$ for which $\alpha_i = 0$
  - Majority of samples
  - Lie away from the hyperplane: $\mathrm{y_i}\left(\mathbf{w}^{\mathrm{T}}\boldsymbol{x}_i + w_0\right) > 1$
  - Have no effect on the hyperplane
- Sample $\boldsymbol{x}_i$ for which $\alpha_i \neq 0$
  - Support Vectors
  - Lie close to the hyperplane: $\mathrm{y_i}\left(\mathbf{w}^{\mathrm{T}}\boldsymbol{x}_i + w_0\right) = 1$
  - Determine the hyperplane

Support Vector Machines: Non-separable case

- If two classes are not linearly separable, we look for the hyperplane that yields the least error
- We define slack variables $\epsilon_i \geq 0$ which represent the deviation from the margin

$$\mathrm{y_i}\left(\mathbf{w}^{\mathrm{T}}\boldsymbol{x}_i + w_0\right) \geq 1 - \epsilon_i$$

- Case (a): Far away from the margin, $\epsilon_i = 0$
- Case (b): On the right side and far from margin, $\epsilon_i = 0$
- Case (c): On the right side, but in the margin, $\epsilon_i > 0$
- Case (d): On the wrong size, $\epsilon_i \geq 1$



Support Vector Machines: Linearly separable case

$$\min_{w} \frac{1}{2} w^T w + C \sum_{i=1}^{N} \epsilon_i \text{ , such that (s.t.) } y_i(w^T x_i + w_0) \geq 1 - \epsilon_i \text{ and } \epsilon_i > 0 \text{ for } i = 1,2,\cdots,N$$

1. Formulate Lagrangian function (primal problem)

$$L = \frac{1}{2} \|\boldsymbol{w}\|_2^2 - C \sum_{i=1}^{N} \epsilon_i - \sum_{i=1}^{N} \alpha_i \left(y_i (\boldsymbol{w}^T \boldsymbol{x}_i + w_0) - 1 + \epsilon_i \right) - \sum_{i=1}^{N} \mu_i \epsilon_i$$

2. Minimize Lagrangian to solve for primal variables w and w_0

$$\frac{\partial L}{\partial \boldsymbol{w}} = 0 \rightarrow \boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i$$

$$\frac{\partial L}{\partial w_0} = 0 \rightarrow 0 = \sum_{i=1}^{N} \alpha_i y_i$$

$$\frac{\partial L}{\partial \epsilon_i} = 0 \rightarrow 0 = C - \alpha_i - \mu_i$$

Support Vector Machines: Linearly separable case

$$\min_{w} \frac{1}{2} w^T w + C \sum_{i=1}^{N} \epsilon_i \text{, such that (s.t.) } y_i(w^T x_i + w_0) \geq 1 - \epsilon_i \text{ and } \epsilon_i > 0 \text{ for } i = 1,2,\cdots,N$$

3. Substitute the primal variables w and w_0 into the Lagrangian and express in terms of dual variables α_i

$$L = \frac{1}{2} \|\boldsymbol{w}\|_2^2 - C \sum_{i=1}^{N} \epsilon_i - \boldsymbol{w}^{\boldsymbol{T}} \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i - w_0 \sum_{i=1}^{N} \alpha_i y_i + \sum_{i=1}^{N} \alpha_i - \sum_{i=1}^{N} \alpha_i \epsilon_i - \sum_{i=1}^{N} \mu_i \epsilon_i$$

$$= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} \boldsymbol{x}_i^T \boldsymbol{x}_{i'}$$

Scribe Notes: Solve from the beginning of (3) to the end

Support Vector Machines: Linearly separable case

$$\min_{w} \frac{1}{2} w^T w, \text{ such that (s.t.) } y_i(w^T x_i + w_0) \geq 1 \text{ for } i = 1,2,\cdots,N$$

4. Maximize the Lagrangian with respect to dual variables (dual problem)

$$\max_{\alpha_i} L = \max_{\alpha_i} \left\{ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} \boldsymbol{x}_i^T \boldsymbol{x}_{i'} \right\}$$

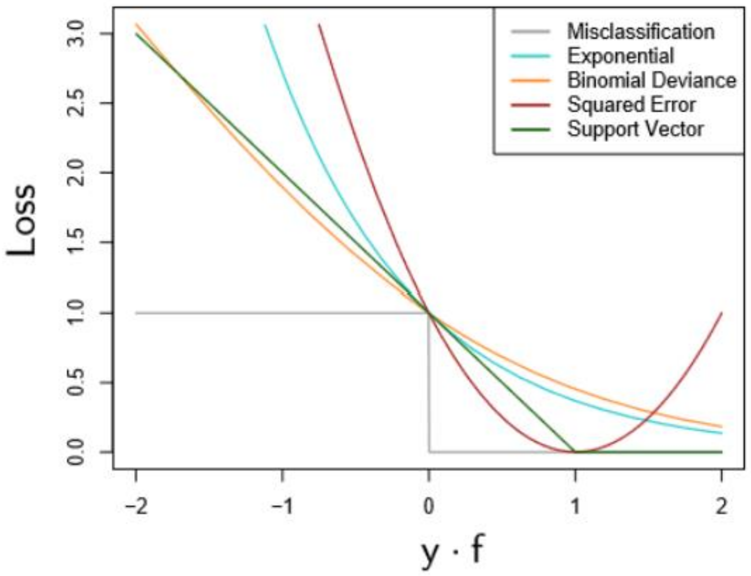
$$\text{s.t.} \sum_{i=1}^{N} \alpha_i y_i = 0 \text{ and } 0 \leq \alpha_i \leq C, for\ i = 1,\cdots,N$$

- Solved numerically suing quadratic optimization methods
- The dual depends on data size N and no on the data dimensionality D
- Most of the α_i will vanish with $\alpha_i = 0$ only a small percentage
- The set of x_i whose $\alpha_i > 0$ are the support vectors
  - $0 < \alpha_i < C$: instances lying on the margin
  - $\alpha_i = C$: instances in the margin or misclassified

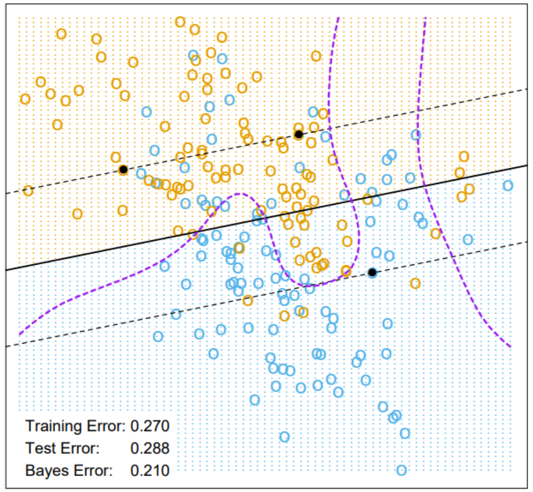
Support Vector Machines: Hinge Loss

- Decision rule: $f(x) = sign(\boldsymbol{w}^T\boldsymbol{x} + w_0)$
  - $f(x) = 1$, if $\boldsymbol{w}^T\boldsymbol{x} + w_0 > 0$
  - $f(x) = -1$, if $\boldsymbol{w}^T\boldsymbol{x} + w_0 < 0$
- If $f(x)$ is the output and y_i the actual label

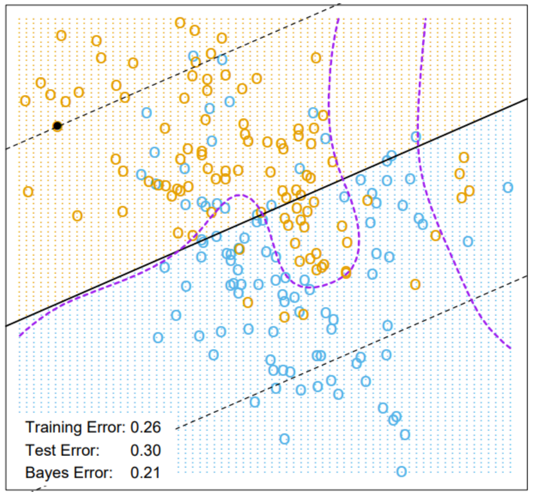
$$L_{Hinge}(f(\boldsymbol{x}), y) = \begin{cases} 0 & if\ y(\boldsymbol{w}^T\boldsymbol{x} + w_0) \geq 1 \\ 1 - y(\boldsymbol{w}^T\boldsymbol{x} + w_0) & otherwise \end{cases}$$



Support Vector Machines: Tuning C



$C = 10000$



$C = 0.01$

So Far

- SVM aims at finding the hyperplane from which instances have a margin of distance
- Prime and dual problem formulation (Lagrange multiplies)
- Support vectors: instances closest to separating hyperplane
- Linearly separable case: maximize margin of separation between two classes
- Non-separable case: look for the hyperplane that yield the least error (soft erro)
  - Prime: minimizes Lagrangian wrt the primal variables of the problem
  - Dual: maximizes Lagrangian wrt multipliers

Coordinate Ascent

$$w(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} \boldsymbol{x}_i^T \boldsymbol{x}_{i'}$$

- Unconstrained optimization problem
  - We have already seen gradient descent (or ascent, if we negate the optimization function), now we consider another optimization method called coordinate ascent

```
Loop until convergence
    1  For i = 1, ..., m
        1a  α_i = arg max_{α̂_i} ω(α_1, ..., α̂_i, ..., α_m)
```

  - In the innermost loop, hold all variables constant except for some fixed α_i
  - Re-optimize w with respect to just the parameter α_i
  - When argmax of the inner loop can be performed efficiently, coordinate ascent can be a fairly efficient algorithm

Sequential Minimal Optimization (SMO)

$$w(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} \boldsymbol{x}_i^T \boldsymbol{x}_{i'}$$

$$\text{s.t.} \sum_{i=1}^{N} \alpha_i y_i = 0 \text{ and } 0 \le \alpha_i \le C, for\ i = 1, \cdots, N$$

Loop until convergence

1 For $i = 1, \ldots, m$

1a Select some α_i and α_j to update

1b Re-optimize $\boldsymbol{\omega}$ wrt α_i and α_j while holding all other α_k's fixed

- Let's assume that we optimize wrt α_1, α_2, while $\alpha_3, \cdots, \alpha_N$ are constant
  - From the constraint:

$$\alpha_1 y_1 + \alpha_2 y_2 = \sum_{i=3}^{N} \alpha_i y_i = \zeta \rightarrow \alpha_1 = (\zeta - \alpha_2 y_2)/y_1$$

  - The objective is $w(\alpha) = w((\zeta - \alpha_2 y_2)/y_1, \alpha_2, \cdots, \alpha_N)$ (some quadrative function of $\alpha_2 \rightarrow$ easily solved by setting its derivative to zero)

Support Vector Machines: Non-separable case

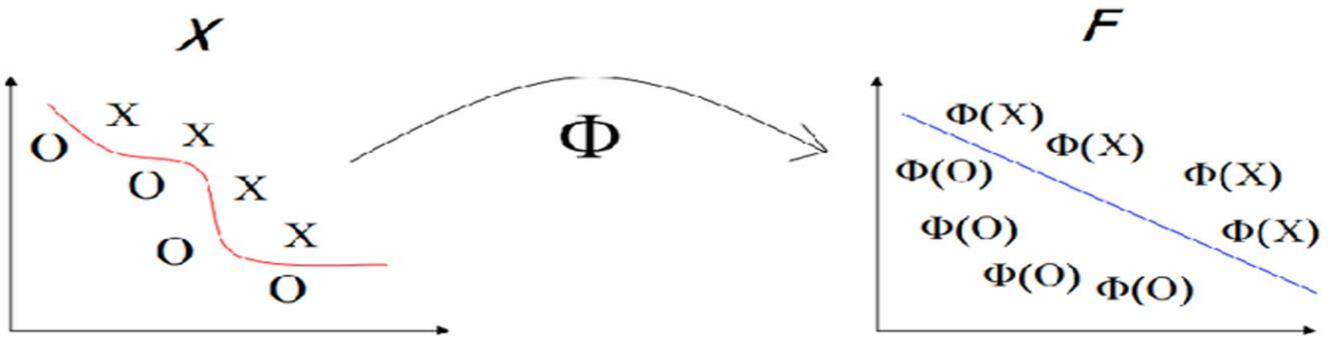
4. Maximize the Lagrangian with respect to dual variables (dual problem)

$$\max_{\alpha_i} L = \max_{\alpha_i} \left\{ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} \boldsymbol{x}_i^T \boldsymbol{x}_{i'} \right\}$$

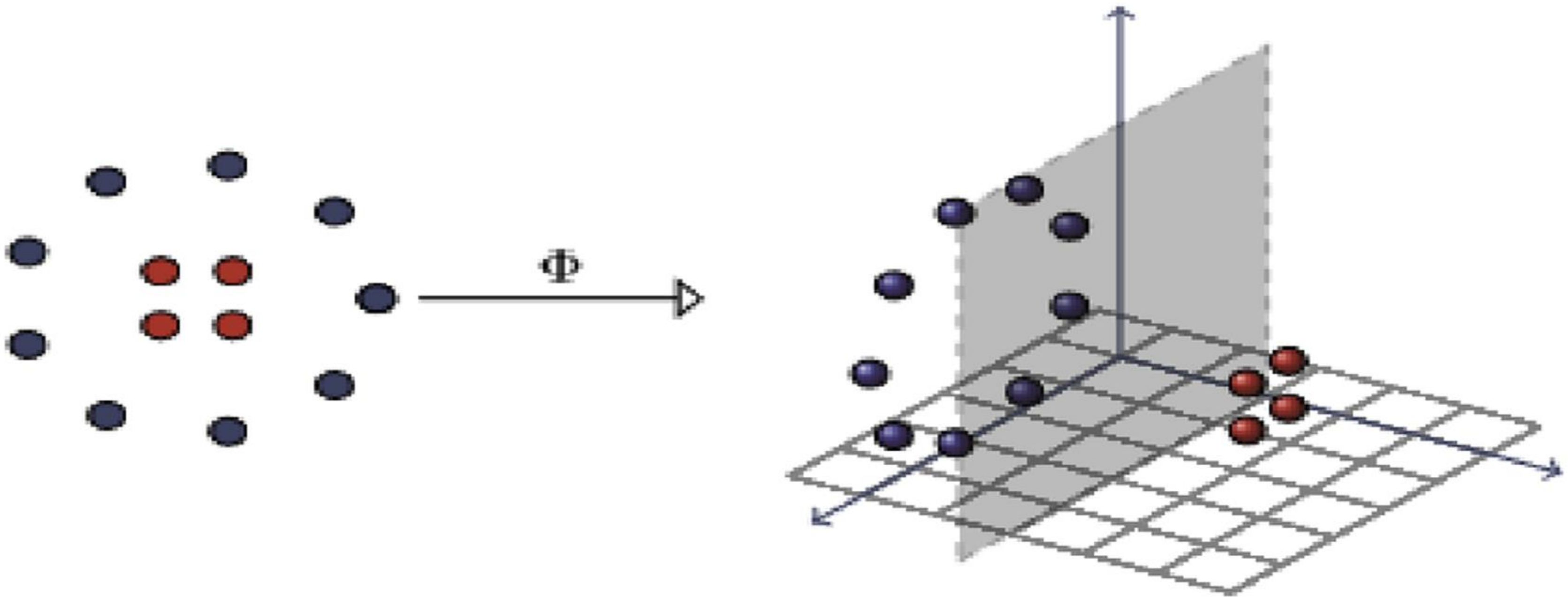
$$\text{s.t.} \sum_{i=1}^{N} \alpha_i y_i = 0 \text{ and } 0 \le \alpha_i \le C, for\ i = 1, \cdots, N$$

Kernel Functions

- Motivation
  - Given a set of vectors, there are many tools available for one to use to detect linear relations among the data
  - But what if the relations are non-linear in the original space?
  - Solution: Map the data into a (possibly high dimensional) vector space where linear relations exist among the data, then apply a linear algorithm in this space



Higher Dimensions



Kernel Functions

- A function that takes as its input vectors in the original space and returns the dot product of the vectors in the feature space is called a kernel
- Definition
  - A (positive semi-definite) kernel function $L(\cdot,\cdot)$ is a bivariate function for which any $\boldsymbol{x}_m$ and $\boldsymbol{x}_n$

$$K(\boldsymbol{x}_m, \boldsymbol{x}_n) = K(\boldsymbol{x}_n, \boldsymbol{x}_m) \; and \; K(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)\,\phi(\boldsymbol{x}_m)$$

- Examples

$$K(\boldsymbol{x}_n, \boldsymbol{x}_m) = \left(\boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{x}_m\right)^2, K(\boldsymbol{x}_n, \boldsymbol{x}_m) = \exp\left(-\frac{\|\boldsymbol{x}_n - \boldsymbol{x}_m\|_2^2}{2\sigma^2}\right)$$

  - Using kernels, we do not need to embed the data into the space explicitly, because a number of algorithms only require the inner products between input vector. We never need the coordinates of the data in the feature space
  - Kernels can be perceived as similarity measures

Kernel Functions

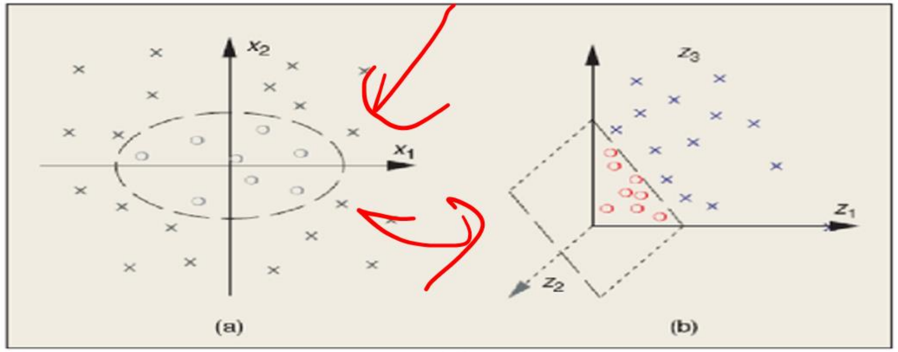
- Example
  - Consider a two-dimensional input space $\boldsymbol{X} \in \mathbb{R}^2$ with the feature map:

$$\phi(\boldsymbol{x}): \boldsymbol{x} = [x_1, x_2]^T \to \left[x_1^2, x_2^2, \sqrt{2}x_1x_2\right]^T \in \mathbb{R}^3$$

  - The inner product in the feature space is

$$\phi(\boldsymbol{x})\phi(\boldsymbol{z}) = \left\langle \left(x_1^2, x_2^2, \sqrt{2}x_1x_2\right), \left(x_1^2, x_2^2, \sqrt{2}x_1x_2\right) \right\rangle$$
$$= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 = (x_1z_1 + x_2z_2)^2 = \langle \boldsymbol{x}, \boldsymbol{y} \rangle^2$$

  - Then $K(\boldsymbol{x}, \boldsymbol{z}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle^2$



1. Effect of the map $\phi(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$ (a) Input space $\mathcal{X}$ and (b) feature space $\mathcal{H}$.

Kernel Functions

- Mercer's condition
  – A bivariate function $K(\cdot,\cdot)$ is a positive semidefinite kernel function, if and only if, for any N and any $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N$ the following matrix, called the Gram matrix, is positive semi-definite.

$$\boldsymbol{K} = \begin{pmatrix} \phi(\boldsymbol{x}_1)\phi(\boldsymbol{x}_1) & \cdots & \phi(\boldsymbol{x}_1)\phi(\boldsymbol{x}_N) \\ \vdots & \ddots & \vdots \\ \phi(\boldsymbol{x}_N)\phi(\boldsymbol{x}_1) & \cdots & \phi(\boldsymbol{x}_N)\phi(\boldsymbol{x}_N) \end{pmatrix} = \boldsymbol{\Phi}^T\boldsymbol{\Phi}$$

  – Mercer's condition tells us whether or not a prospective kernel is actually a dot product in some space

Examples of Kernel Functions

- Polynomial kernel function with degree of d

$$K(\boldsymbol{x}_n, \boldsymbol{x}_m) = (\boldsymbol{x}_n^T \boldsymbol{x}_m + c)^d$$

  – For $c \geq 0$ and d a positive integer

- Gaussian kernel, RBF (radial basis function) kernel, or Gaussian RBF kernel

$$K(\boldsymbol{x}_n, \boldsymbol{x}_m) = \exp\left(-\frac{\|\boldsymbol{x}_n - \boldsymbol{x}_m\|_2^2}{2\sigma^2}\right)$$

- Sigmoid Kernel

$$K(\boldsymbol{x}_n, \boldsymbol{x}_m) = \tanh(\mathrm{a}(\boldsymbol{x}_n^T \boldsymbol{x}_m) + b)$$

- Most of those kernels have parameters to be tuned: d, c, σ^2, etc. They are hyperparameters and are often tuned on holdout data or with cross-validation

Examples of Kernel Functions

- Document Similarity
  - Let x_{ij} be he # times a word j occurs in a document I (bag-of-words)
  - Cosine similarity between documents i and i' counts the number of shared words

$$K(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) = \tanh(\mathrm{a}(\boldsymbol{x}_n^T \boldsymbol{x}_m) + b)$$

- Edit distance (e.g. gene alignment)
  - # insertions, deletions, substitutions it takes to convert one gene to another

Rules for composing kernels

- There are infinite number of kernels to use
  - If $K(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$ is a kernel, then $\mathrm{c}K(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$, $\mathrm{c} > 0$, is a kernel
  - If $K(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$ is a kernel, then $e^{K(\boldsymbol{x}_i, \boldsymbol{x}_{i'})}$ is a kernel
  - If $K_1(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$ and $K_2(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$ are kernels, then $\alpha K_1(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) + \omega K_2(\boldsymbol{x}_i, \boldsymbol{x}_{i'}), \alpha, \omega > 0$ is a kernel
  - If $K_1(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$ and $K_2(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$ are kernels, then $K_1(\boldsymbol{x}_i, \boldsymbol{x}_{i'})K_2(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$ is a kernel

- In practice, using which kernel, or which kernels to compose a new kernel, remains somewhat of a “black art”, though most people will start with polynomial and Gaussian RBF kernels.

- Example: Audio-visual speech recognition, where notion of similarity is differently defined for speech and image

Kernel Trick

- Many learning methods depend on computing inner products between features, e.g. linear regression

- For those methods, we can use a kernel function in the place of the inner products, i.e. “kernelizing” the methods, thus, introducing nonlinear features/basis

- Instead of first transforming the original features into the new feature space and then computing the inner product, we can compute the inner product in the new feature space directly through the kernel function.

Support Vector Machines: Kernel Trick

- Set of basis functions $\boldsymbol{z} = \phi(\boldsymbol{x})$
- Map the problem into the new space and solve as before
- For SVM, this leads to certain simlifications
- Setup for two classes
  - Input: $\boldsymbol{x} \in \mathbb{R}^N$
  - Output: $y \in \{-1,1\}$
  - Training data: $\mathcal{D}^{train} = \{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_N, y_N)\}$
  - Non-separable model:

$$f(\boldsymbol{x}_i) = \begin{cases} 1 & if\ w^T\phi(\boldsymbol{x}_i) + w_0 > -(1-\epsilon_i) \\ -1 & if\ w^T\phi(\boldsymbol{x}_i) + w_0 \le -(1-\epsilon_i) \end{cases}$$

Support Vector Machines: Linearly separable case

$$\min_{w} \frac{1}{2} w^T w + C \sum_{i=1}^{N} \epsilon_i \text{, such that (s.t.) } y_i(w^T \phi(\boldsymbol{x}_i) + w_0) \geq 1 - \epsilon_i \text{ and } \epsilon_i > 0 \text{ for } i = 1,2,\cdots,N$$

1. Formulate Lagrangian function (primal problem)

$$L = \frac{1}{2} \|\boldsymbol{w}\|_2^2 - C \sum_{i=1}^{N} \epsilon_i - \sum_{i=1}^{N} \alpha_i \big(y_i(\boldsymbol{w}^{\boldsymbol{T}} \phi(\boldsymbol{x}_i) + w_0\big) - 1 + \epsilon_i) - \sum_{i=1}^{N} \mu_i \epsilon_i$$

2. Minimize Lagrangian to solve for primal variables w and w_0

$$\frac{\partial L}{\partial \boldsymbol{w}} = 0 \rightarrow \boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \phi(\boldsymbol{x}_i)$$

$$\frac{\partial L}{\partial w_0} = 0 \rightarrow 0 = \sum_{i=1}^{N} \alpha_i y_i$$

$$\frac{\partial L}{\partial \epsilon_i} = 0 \rightarrow 0 = C - \alpha_i - \mu_i$$

Support Vector Machines: Linearly separable case

$$\min_{w} \frac{1}{2} w^T w + C \sum_{i=1}^{N} \epsilon_i \text{, such that (s.t.) } y_i(w^T x_i + w_0) \geq 1 - \epsilon_i \text{ and } \epsilon_i > 0 \text{ for } i = 1,2,\cdots,N$$

3. Substitute the primal variables w and w_0 into the Lagrangian and express in terms of dual variables α_i

$$L = \frac{1}{2} \|\boldsymbol{w}\|_2^2 - C \sum_{i=1}^{N} \epsilon_i - \boldsymbol{w}^{\boldsymbol{T}} \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i - w_0 \sum_{i=1}^{N} \alpha_i y_i + \sum_{i=1}^{N} \alpha_i - \sum_{i=1}^{N} \alpha_i \epsilon_i - \sum_{i=1}^{N} \mu_i \epsilon_i$$

$$= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_{i'})$$

Support Vector Machines: Kernel Trick

- Instead of transforming $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i'}$ through ϕ and computing their inner product, we directly apply the kernel function to the original space
- Lagrangian for the dual problem

$$L = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} K(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$$

$$K(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_{i'})$$

- For taking a decision in the test set

$$\sum_{i'=1}^{N} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x})$$

- The cost of computing the primal variables is $\mathrm{O}(\mathrm{N}^3)$, while for the dual variable is $\mathrm{O}(D^3)$. A kernel method can be useful in high dimensional settings

Support Vector Machines: Multiple Kernel Learning

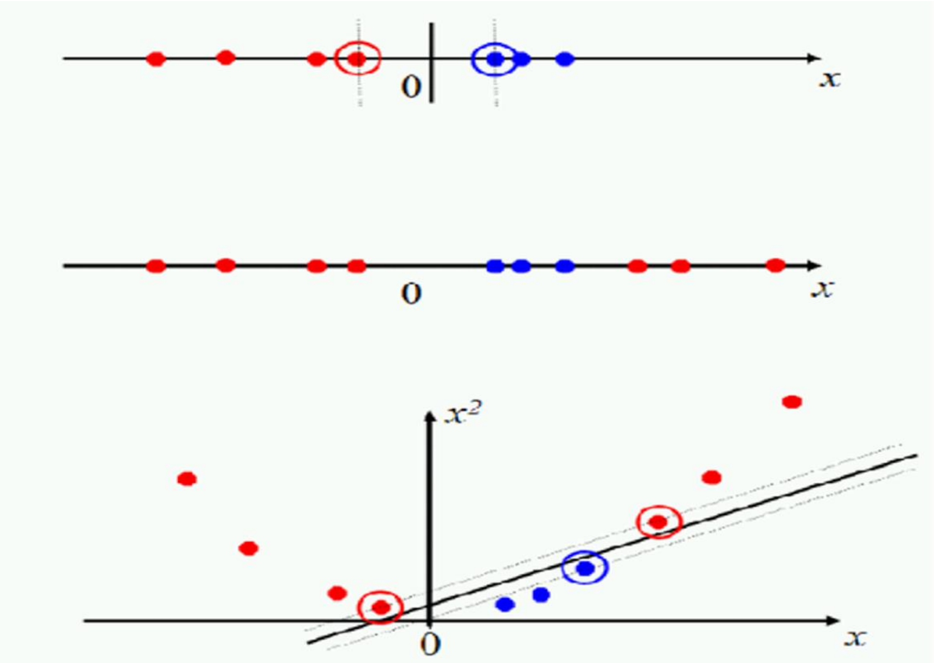
- Weighted sum of the kernels: $K(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) = \sum_{l=1}^{L} \nu_l \, K_l(\boldsymbol{x}_i, \boldsymbol{x}_{i'}), \nu_l \geq 0$
- Objective function

$$L = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} \sum_{l=1}^{L} \nu_l \, K_l(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$$

- Solved for both the support vectors, α_i and the weights
- For taking a decision in the test set

$$\sum_{i'=1}^{N} \alpha_i y_i \sum_{l=1}^{L} \nu_l \, K(\boldsymbol{x}_i, \boldsymbol{x})$$

Support Vector Machines: Multiple Kernel Learning



- Projecting data that is not linearly separable into a higher dimensional space can make it linearly separable

Multi-class kernel machines

- One-vs-all approach
  - Define K two-class problems, each separating one class from all others
  - Learn K binary support vector machines $f_i(x), i = 1, \ldots, K$
    - Class 1: Examples from class i
    - Class -1: Examples from all classes besides class p
  - Decision during testing

$$f_i(x) = argmax_i f_i(x)$$

Multi-class kernel machines

- One-vs-one approach
  - Define K(K-1) two-class problems, each separating class I from class j
  - Learn K(K-1) binary SVM $f_{ij}(x)$
    - Class 1: Examples from class i
    - Class -1: Examples from class j
    - Note that $f_{ij} = -f_{ji}$
  - Decision during testing

$$f_i(x) = argmax_i \left(\sum_j f_j(x) \right)$$

Multi-class kernel machines

- Comparison of one-vs-all and one-vs-one approach
  - One-vs-one
    - Requires $O(K^2)$ classifiers instead of $O(K)$
    - But each classifier is on average smaller $O\left(\frac{2N}{K}\right)$
  - One-vs-all approach solve $O(K)$ separate problems, each of size $O(N)$

Multi-class kernel machines

- Multi-class formulation
  - Define K weights for each class $w_1, \dots, w_k$ and K bias terms $w_{01}, \dots, w_{0K}$
  - Training data $\{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_N, y_N)\}, y_i \in \{1, \dots, K\}$
  - Optimization criterion

$$\min_{w_1,\dots,w_k} \frac{1}{2}\sum_{k=1}^{K} \|w_k\|_2^2 + C\sum_{k=1}^{K}\sum_{i=1}^{N} \epsilon_{nk}$$

$$s.t.\, \boldsymbol{w}_{y_n}^t \boldsymbol{x}_n + \mathrm{w}_0 \geq \boldsymbol{w}_k^T \boldsymbol{x}_i + w_{0k} + 2 - \epsilon_{nk}, \forall k \neq y_n$$

(ie. So that the weight for each class yields a sufficient margin form the other classes)

What have we learnt so far

- Kernel Functions $K(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$
  - $K(\boldsymbol{x}_m, \boldsymbol{x}_n) = K(\boldsymbol{x}_n, \boldsymbol{x}_m)$ and $K(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)\, \phi(\boldsymbol{x}_m)$
  - Positive definite kernel -> positive definite Gram matrix
- Kernel trick
  - Instead of first transforming the original features into the new features space and then computing the inener product, we can compute the inner product in the new features space directly thought he kernel function
  - Kernel SVM: The cost of computing the primal variables is $\mathrm{O}(\mathrm{N}^3)$, while for the dual variable is $\mathrm{O}(D^3)$. A kernel method can be useful in high dimensional settings
- Multi-class SVM
  - One-vs-one, one-vs-all, multiclass formulation

Takeaways and Next Time

- Primal-Dual formulation of SVM
- Kernelization
- Next time: Introduction to dimension reduction and unsupervised learning