Support Vector Machines (Contd.), Classification Loss Functions and Regularizers

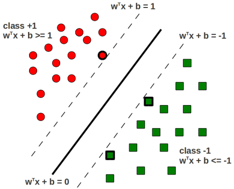
Piyush Rai

CS5350/6350: Machine Learning

September 13, 2011

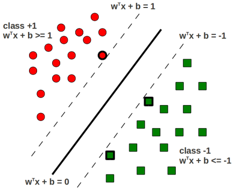
SVM (Recap)

- SVM finds the maximum margin hyperplane that separates the classes



SVM (Recap)

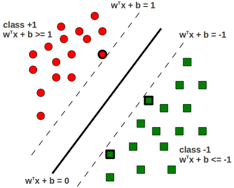
- SVM finds the maximum margin hyperplane that separates the classes



- Margin $\gamma = \frac{1}{||\mathbf{w}||}$

SVM (Recap)

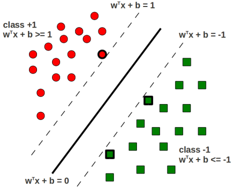
- SVM finds the maximum margin hyperplane that separates the classes



- Margin $\gamma = \frac{1}{||\mathbf{w}||} \Rightarrow$ maximizing the margin $\gamma \equiv$ minimizing $||\mathbf{w}||$ (the norm)

SVM (Recap)

- SVM finds the maximum margin hyperplane that separates the classes



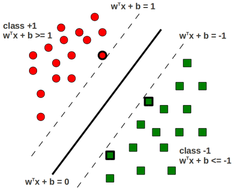
- Margin $\gamma = \frac{1}{||\mathbf{w}||} \Rightarrow$ maximizing the margin $\gamma \equiv$ minimizing $||\mathbf{w}||$ (the norm)
- The optimization problem for the separable case (no misclassified training example)

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

$$\text{subject to} \quad y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1, \qquad n = 1, \ldots, N$$

SVM (Recap)

- SVM finds the maximum margin hyperplane that separates the classes



- Margin $\gamma = \frac{1}{||\mathbf{w}||} \Rightarrow$ maximizing the margin $\gamma \equiv$ minimizing $||\mathbf{w}||$ (the norm)
- The optimization problem for the separable case
  (no misclassified training example)

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

$$\text{subject to} \quad y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1, \qquad n = 1, \ldots, N$$

- This is a Quadratic Program (QP) with N linear inequality constraints

SVM: The Optimization Problem

- Our optimization problem is:

$$\boxed{\begin{array}{ll} \text{Minimize} \quad f(\mathbf{w}, b) = \dfrac{||\mathbf{w}||^2}{2} & \\ \text{subject to} \quad 1 \leq y_n(\mathbf{w}^T\mathbf{x}_n + b), & \qquad n = 1, \ldots, N \end{array}}$$

SVM: The Optimization Problem

- Our optimization problem is:

$$\text{Minimize} \quad f(\mathbf{w}, b) = \frac{||\mathbf{w}||^2}{2}$$
$$\text{subject to} \quad 1 \leq y_n(\mathbf{w}^T\mathbf{x}_n + b), \qquad n = 1, \ldots, N$$

- Introducing Lagrange Multipliers α_n ($n = \{1, \ldots, N\}$), one for each constraint, leads to the Primal Lagrangian:

$$\text{Minimize} \quad L_P(\mathbf{w}, b, \alpha) = \frac{||\mathbf{w}||^2}{2} + \sum_{n=1}^{N} \alpha_n \{1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)\}$$
$$\text{subject to} \quad \alpha_n \geq 0; \quad n = 1, \ldots, N$$

SVM: The Optimization Problem

- Our optimization problem is:

$$\text{Minimize} \quad f(\mathbf{w}, b) = \frac{||\mathbf{w}||^2}{2}$$
$$\text{subject to} \quad 1 \leq y_n(\mathbf{w}^T\mathbf{x}_n + b), \qquad n = 1, \dots, N$$

- Introducing Lagrange Multipliers α_n ($n = \{1, \dots, N\}$), one for each constraint, leads to the Primal Lagrangian:

$$\text{Minimize} \quad L_P(\mathbf{w}, b, \alpha) = \frac{||\mathbf{w}||^2}{2} + \sum_{n=1}^{N} \alpha_n \{1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)\}$$
$$\text{subject to} \quad \alpha_n \geq 0; \quad n = 1, \dots, N$$

- We can now solve this Lagrangian
  - i.e., optimize $L(\mathbf{w}, b, \alpha)$ w.r.t. $\mathbf{w}$, b, and α
  - .. making use of the Lagrangian Duality theory..

SVM: The Optimization Problem

- Take (partial) derivatives of L_P w.r.t. $\mathbf{w}$, b and set them to zero

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n,$$

SVM: The Optimization Problem

- Take (partial) derivatives of L_P w.r.t. $\mathbf{w}$, b and set them to zero

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n, \quad \frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0$$

SVM: The Optimization Problem

- Take (partial) derivatives of L_P w.r.t. $\mathbf{w}$, b and set them to zero

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n, \quad \frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0$$

- Substituting these in the Primal Lagrangian L_P gives the Dual Lagrangian

$$\text{Maximize} \quad L_D(\mathbf{w}, b, \alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m,n=1}^{N} \alpha_m \alpha_n y_m y_n (\mathbf{x}_m^T \mathbf{x}_n)$$

$$\text{subject to} \quad \sum_{n=1}^{N} \alpha_n y_n = 0, \quad \alpha_n \geq 0; \quad n = 1, \ldots, N$$

SVM: The Optimization Problem

- Take (partial) derivatives of L_P w.r.t. $\mathbf{w}$, b and set them to zero

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n, \quad \frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0$$

- Substituting these in the Primal Lagrangian L_P gives the Dual Lagrangian

$$\text{Maximize} \quad L_D(\mathbf{w}, b, \alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m,n=1}^{N} \alpha_m \alpha_n y_m y_n (\mathbf{x}_m^T \mathbf{x}_n)$$

$$\text{subject to} \quad \sum_{n=1}^{N} \alpha_n y_n = 0, \quad \alpha_n \geq 0; \quad n = 1, \dots, N$$

- It's a Quadratic Programming problem in α

SVM: The Optimization Problem

- Take (partial) derivatives of L_P w.r.t. $\mathbf{w}$, b and set them to zero

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n, \quad \frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0$$

- Substituting these in the Primal Lagrangian L_P gives the Dual Lagrangian

$$\text{Maximize} \quad L_D(\mathbf{w}, b, \alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m,n=1}^{N} \alpha_m \alpha_n y_m y_n (\mathbf{x}_m^T \mathbf{x}_n)$$

$$\text{subject to} \quad \sum_{n=1}^{N} \alpha_n y_n = 0, \quad \alpha_n \geq 0; \quad n = 1, \ldots, N$$

- It's a Quadratic Programming problem in α
  - Several off-the-shelf solvers exist to solve such QPs

SVM: The Optimization Problem

- Take (partial) derivatives of L_P w.r.t. $\mathbf{w}$, b and set them to zero

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n, \quad \frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0$$

- Substituting these in the Primal Lagrangian L_P gives the Dual Lagrangian

$$\text{Maximize} \quad L_D(\mathbf{w}, b, \alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m,n=1}^{N} \alpha_m \alpha_n y_m y_n (\mathbf{x}_m^T \mathbf{x}_n)$$

$$\text{subject to} \quad \sum_{n=1}^{N} \alpha_n y_n = 0, \quad \alpha_n \geq 0; \quad n = 1, \ldots, N$$

- It's a Quadratic Programming problem in α
  - Several off-the-shelf solvers exist to solve such QPs
  - Some examples: `quadprog` (MATLAB), CVXOPT, CPLEX, IPOPT, etc.

SVM: The Solution

- Once we have the α_n's, **w** and b can be computed as:

$$\mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n$$

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

SVM: The Solution

- Once we have the α_n's, $\mathbf{w}$ and b can be computed as:

$$\mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n$$

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

- **Note:** Most α_n's in the solution are zero (sparse solution)

SVM: The Solution

- Once we have the α_n's, $\mathbf{w}$ and b can be computed as:

$$\mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n$$

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

- **Note:** Most α_n's in the solution are zero (sparse solution)
  - Reason: Karush-Kuhn-Tucker (KKT) conditions

SVM: The Solution

- Once we have the α_n's, $\mathbf{w}$ and b can be computed as:

$$\mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n$$

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

- **Note:** Most α_n's in the solution are zero (sparse solution)
  - Reason: Karush-Kuhn-Tucker (KKT) conditions
  - For the optimal α_n's

$$\alpha_n\{1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)\} = 0$$

SVM: The Solution

- Once we have the α_n's, $\mathbf{w}$ and b can be computed as:

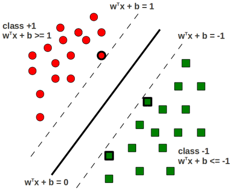
$$\mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n$$

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

- **Note:** Most α_n's in the solution are zero (sparse solution)
  - Reason: Karush-Kuhn-Tucker (KKT) conditions
  - For the optimal α_n's

$$\alpha_n\{1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)\} = 0$$

  - α_n is non-zero only if $\mathbf{x}_n$ lies on one of the two margin boundaries, i.e., for which $y_n(\mathbf{w}^T\mathbf{x}_n + b) = 1$
  - These examples are called support vectors
  - Support vectors "support" the margin boundaries



SVM - Non-separable case

- Non-separable case: No hyperplane can separate the classes perfectly

SVM - Non-separable case

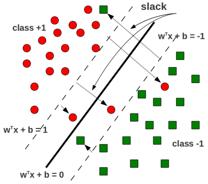
- Non-separable case: No hyperplane can separate the classes perfectly
- Still want to find the maximum margin hyperplane but this time:

SVM - Non-separable case

- Non-separable case: No hyperplane can separate the classes perfectly
- Still want to find the maximum margin hyperplane but this time:
  - We will allow some training examples to be misclassified

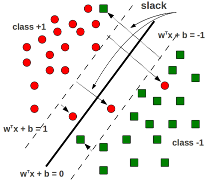
SVM - Non-separable case

- Non-separable case: No hyperplane can separate the classes perfectly
- Still want to find the maximum margin hyperplane but this time:
  - We will allow some training examples to be misclassified
  - We will allow some training examples to fall **within** the margin region



SVM - Non-separable case

- Non-separable case: No hyperplane can separate the classes perfectly
- Still want to find the maximum margin hyperplane but this time:
  - We will allow some training examples to be misclassified
  - We will allow some training examples to fall **within** the margin region

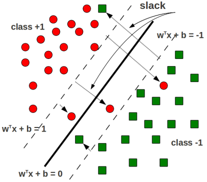


- Recall: For the separable case (training loss = 0), the constraints were:

$$y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 \quad \forall n$$

SVM - Non-separable case

- Non-separable case: No hyperplane can separate the classes perfectly
- Still want to find the maximum margin hyperplane but this time:
  - We will allow some training examples to be misclassified
  - We will allow some training examples to fall **within** the margin region



- Recall: For the separable case (training loss = 0), the constraints were:

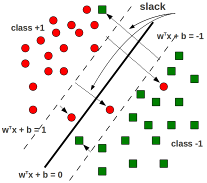
$$y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 \quad \forall n$$

- For the non-separable case, we relax the above constraints as:

$$\boxed{y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \xi_n \quad \forall n}$$

SVM - Non-separable case

- Non-separable case: No hyperplane can separate the classes perfectly
- Still want to find the maximum margin hyperplane but this time:
  - We will allow some training examples to be misclassified
  - We will allow some training examples to fall **within** the margin region



- Recall: For the separable case (training loss = 0), the constraints were:

$$y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 \quad \forall n$$

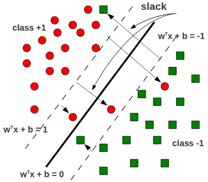
- For the non-separable case, we relax the above constraints as:

$$\boxed{y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \xi_n \quad \forall n}$$

- ξ_n is called slack variable (distance $\mathbf{x}_n$ goes past the margin boundary)

SVM - Non-separable case

- Non-separable case: No hyperplane can separate the classes perfectly
- Still want to find the maximum margin hyperplane but this time:
  - We will allow some training examples to be misclassified
  - We will allow some training examples to fall **within** the margin region



- Recall: For the separable case (training loss = 0), the constraints were:

$$y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 \quad \forall n$$

- For the non-separable case, we relax the above constraints as:

$$\boxed{y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \xi_n \quad \forall n}$$

- ξ_n is called slack variable (distance $\mathbf{x}_n$ goes past the margin boundary)
- $\xi_n \geq 0, \forall n$, misclassification when $\xi_n > 1$

SVM - Non-separable case

- Non-separable case: We will allow misclassified training examples

SVM - Non-separable case

- Non-separable case: We will allow misclassified training examples
  - .. but we want their number to be minimized

SVM - Non-separable case

- Non-separable case: We will allow misclassified training examples
  - .. but we want their number to be minimized
    $\Rightarrow$ by *minimizing* the sum of slack variables ($\sum_{n=1}^{N} \xi_n$)

SVM - Non-separable case

- Non-separable case: We will allow misclassified training examples
  - .. but we want their number to be minimized
    $\Rightarrow$ by *minimizing* the sum of slack variables ($\sum_{n=1}^{N} \xi_n$)
- The optimization problem for the non-separable case

$$\text{Minimize} \quad f(\mathbf{w}, b) = \frac{||\mathbf{w}||^2}{2} + C \sum_{n=1}^{N} \xi_n$$

$$\text{subject to} \quad y_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1 - \xi_n, \quad \xi_n \geq 0 \qquad n = 1, \ldots, N$$

SVM - Non-separable case

- Non-separable case: We will allow misclassified training examples
  - .. but we want their number to be minimized
    $\Rightarrow$ by *minimizing* the sum of slack variables ($\sum_{n=1}^{N} \xi_n$)
- The optimization problem for the non-separable case

$$\text{Minimize} \quad f(\mathbf{w}, b) = \frac{||\mathbf{w}||^2}{2} + C \sum_{n=1}^{N} \xi_n$$

$$\text{subject to} \quad y_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1 - \xi_n, \quad \xi_n \geq 0 \qquad n = 1, \ldots, N$$

- C dictates which term ($\frac{||\mathbf{w}||^2}{2}$ or $C \sum_{n=1}^{N} \xi_n$) will dominate the minimization

SVM - Non-separable case

- Non-separable case: We will allow misclassified training examples
  - .. but we want their number to be minimized
    $\Rightarrow$ by *minimizing* the sum of slack variables ($\sum_{n=1}^{N} \xi_n$)
- The optimization problem for the non-separable case

$$\text{Minimize} \quad f(\mathbf{w}, b) = \frac{||\mathbf{w}||^2}{2} + C\sum_{n=1}^{N} \xi_n$$

$$\text{subject to} \quad y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \xi_n, \quad \xi_n \geq 0 \qquad n = 1, \ldots, N$$

- C dictates which term ($\frac{||\mathbf{w}||^2}{2}$ or $C\sum_{n=1}^{N} \xi_n$) will dominate the minimization
  - Small $C \Rightarrow \frac{||\mathbf{w}||^2}{2}$ dominates $\Rightarrow$ prefer large margins

SVM - Non-separable case

- Non-separable case: We will allow misclassified training examples
  - .. but we want their number to be minimized
    $\Rightarrow$ by *minimizing* the sum of slack variables ($\sum_{n=1}^{N} \xi_n$)
- The optimization problem for the non-separable case

$$\text{Minimize} \quad f(\mathbf{w}, b) = \frac{||\mathbf{w}||^2}{2} + C\sum_{n=1}^{N} \xi_n$$

$$\text{subject to} \quad y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \xi_n, \quad \xi_n \geq 0 \qquad n = 1, \ldots, N$$

- C dictates which term ($\frac{||\mathbf{w}||^2}{2}$ or $C\sum_{n=1}^{N} \xi_n$) will dominate the minimization
  - Small $C \Rightarrow \frac{||\mathbf{w}||^2}{2}$ dominates $\Rightarrow$ prefer large margins
    - .. but allow potentially large # of misclassified training examples

SVM - Non-separable case

- Non-separable case: We will allow misclassified training examples
  - .. but we want their number to be minimized
    $\Rightarrow$ by *minimizing* the sum of slack variables ($\sum_{n=1}^{N} \xi_n$)
- The optimization problem for the non-separable case

$$\text{Minimize} \quad f(\mathbf{w}, b) = \frac{||\mathbf{w}||^2}{2} + C\sum_{n=1}^{N} \xi_n$$

$$\text{subject to} \quad y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \xi_n, \quad \xi_n \geq 0 \qquad n = 1, \ldots, N$$

- C dictates which term ($\frac{||\mathbf{w}||^2}{2}$ or $C\sum_{n=1}^{N} \xi_n$) will dominate the minimization
  - Small $C \Rightarrow \frac{||\mathbf{w}||^2}{2}$ dominates $\Rightarrow$ prefer large margins
    - .. but allow potentially large # of misclassified training examples
  - Large $C \Rightarrow C\sum_{n=1}^{N} \xi_n$ dominates $\Rightarrow$ prefer small # of misclassified examples

SVM - Non-separable case

- Non-separable case: We will allow misclassified training examples
  - .. but we want their number to be minimized
    $\Rightarrow$ by *minimizing* the sum of slack variables $(\sum_{n=1}^{N} \xi_n)$
- The optimization problem for the non-separable case

$$\text{Minimize} \quad f(\mathbf{w}, b) = \frac{||\mathbf{w}||^2}{2} + C\sum_{n=1}^{N} \xi_n$$

$$\text{subject to} \quad y_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1 - \xi_n, \quad \xi_n \geq 0 \qquad n = 1, \ldots, N$$

- C dictates which term $(\frac{||\mathbf{w}||^2}{2}$ or $C\sum_{n=1}^{N} \xi_n)$ will dominate the minimization
  - Small $C \Rightarrow \frac{||\mathbf{w}||^2}{2}$ dominates $\Rightarrow$ prefer large margins
    - .. but allow potentially large # of misclassified training examples
  - Large $C \Rightarrow C\sum_{n=1}^{N} \xi_n$ dominates $\Rightarrow$ prefer small # of misclassified examples
    - .. at the expense of having a small margin

SVM - Non-separable case: The Optimization Problem

- Our optimization problem is:

$$\text{Minimize} \quad f(\mathbf{w}, b, \xi) = \frac{||\mathbf{w}||^2}{2} + C\sum_{n=1}^{N} \xi_n$$

$$\text{subject to} \quad 1 \leq y_n(\mathbf{w}^T\mathbf{x}_n + b) + \xi_n, \quad 0 \leq \xi_n \qquad n = 1, \dots, N$$

SVM - Non-separable case: The Optimization Problem

- Our optimization problem is:

$$\text{Minimize} \quad f(\mathbf{w}, b, \xi) = \frac{||\mathbf{w}||^2}{2} + C \sum_{n=1}^{N} \xi_n$$

$$\text{subject to} \quad 1 \leq y_n(\mathbf{w}^T \mathbf{x}_n + b) + \xi_n, \quad 0 \leq \xi_n \qquad n = 1, \ldots, N$$

- Introducing Lagrange Multipliers α_n, β_n $(n = \{1, \ldots, N\})$, for the constraints, leads to the Primal Lagrangian:

$$\text{Minimize} \quad L_P(\mathbf{w}, b, \xi, \alpha, \beta) = \frac{||\mathbf{w}||^2}{2} + + C \sum_{n=1}^{N} \xi_n + \sum_{n=1}^{N} \alpha_n \{1 - y_n(\mathbf{w}^T \mathbf{x}_n + b) - \xi_n\} - \sum_{n=1}^{N} \beta_n \xi_n$$

$$\text{subject to} \quad \alpha_n, \beta_n \geq 0; \quad n = 1, \ldots, N$$

SVM - Non-separable case: The Optimization Problem

- Our optimization problem is:

$$\text{Minimize} \quad f(\mathbf{w}, b, \xi) = \frac{||\mathbf{w}||^2}{2} + C\sum_{n=1}^{N} \xi_n$$

$$\text{subject to} \quad 1 \leq y_n(\mathbf{w}^T\mathbf{x}_n + b) + \xi_n, \quad 0 \leq \xi_n \qquad n = 1, \dots, N$$

- Introducing Lagrange Multipliers α_n, β_n ($n = \{1, \dots, N\}$), for the constraints, leads to the Primal Lagrangian:

$$\text{Minimize} \quad L_P(\mathbf{w}, b, \xi, \alpha, \beta) = \frac{||\mathbf{w}||^2}{2} + +C\sum_{n=1}^{N} \xi_n + \sum_{n=1}^{N} \alpha_n\{1 - y_n(\mathbf{w}^T\mathbf{x}_n + b) - \xi_n\} - \sum_{n=1}^{N} \beta_n\xi_n$$

$$\text{subject to} \quad \alpha_n, \beta_n \geq 0; \quad n = 1, \dots, N$$

- Comparison note: Terms in red font were not there in the separable case

SVM - Non-separable case: The Optimization Problem

- Take (partial) derivatives of L_P w.r.t. $\mathbf{w}$, b, ξ_n and set them to zero

SVM - Non-separable case: The Optimization Problem

- Take (partial) derivatives of L_P w.r.t. $\mathbf{w}$, b, ξ_n and set them to zero

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n,$$

SVM - Non-separable case: The Optimization Problem

- Take (partial) derivatives of L_P w.r.t. $\mathbf{w}$, b, ξ_n and set them to zero

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n, \quad \frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0,$$

SVM - Non-separable case: The Optimization Problem

- Take (partial) derivatives of L_P w.r.t. $\mathbf{w}$, b, ξ_n and set them to zero

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n, \quad \frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0, \quad \frac{\partial L_P}{\partial \xi_n} = 0 \Rightarrow C - \alpha_n - \beta_n = 0$$

SVM - Non-separable case: The Optimization Problem

- Take (partial) derivatives of L_P w.r.t. $\mathbf{w}$, b, ξ_n and set them to zero

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n, \quad \frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0, \quad \frac{\partial L_P}{\partial \xi_n} = 0 \Rightarrow C - \alpha_n - \beta_n = 0$$

- Using $C - \alpha_n - \beta_n = 0$ and $\beta_n \geq 0$

SVM - Non-separable case: The Optimization Problem

- Take (partial) derivatives of L_P w.r.t. $\mathbf{w}$, b, ξ_n and set them to zero

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n, \quad \frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0, \quad \frac{\partial L_P}{\partial \xi_n} = 0 \Rightarrow C - \alpha_n - \beta_n = 0$$

- Using $C - \alpha_n - \beta_n = 0$ and $\beta_n \geq 0 \Rightarrow \alpha_n \leq C$

SVM - Non-separable case: The Optimization Problem

- Take (partial) derivatives of L_P w.r.t. $\mathbf{w}$, b, ξ_n and set them to zero

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n, \quad \frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0, \quad \frac{\partial L_P}{\partial \xi_n} = 0 \Rightarrow C - \alpha_n - \beta_n = 0$$

- Using $C - \alpha_n - \beta_n = 0$ and $\beta_n \geq 0 \Rightarrow \alpha_n \leq C$
- Substituting these in the Primal Lagrangian L_P gives the Dual Lagrangian

$$\text{Maximize} \quad L_D(\mathbf{w}, b, \xi, \alpha, \beta) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m,n=1}^{N} \alpha_m \alpha_n y_m y_n (\mathbf{x}_m^T \mathbf{x}_n)$$

$$\text{subject to} \quad \sum_{n=1}^{N} \alpha_n y_n = 0, \quad 0 \leq \alpha_n \leq C; \quad n = 1, \ldots, N$$

SVM - Non-separable case: The Optimization Problem

- Take (partial) derivatives of L_P w.r.t. $\mathbf{w}$, b, ξ_n and set them to zero

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n, \quad \frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0, \quad \frac{\partial L_P}{\partial \xi_n} = 0 \Rightarrow C - \alpha_n - \beta_n = 0$$

- Using $C - \alpha_n - \beta_n = 0$ and $\beta_n \geq 0 \Rightarrow \alpha_n \leq C$
- Substituting these in the Primal Lagrangian L_P gives the Dual Lagrangian

$$\text{Maximize} \quad L_D(\mathbf{w}, b, \xi, \alpha, \beta) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m,n=1}^{N} \alpha_m \alpha_n y_m y_n (\mathbf{x}_m^T \mathbf{x}_n)$$

$$\text{subject to} \quad \sum_{n=1}^{N} \alpha_n y_n = 0, \quad 0 \leq \alpha_n \leq C; \quad n = 1, \ldots, N$$

- Again a Quadratic Programming problem in α

SVM - Non-separable case: The Optimization Problem

- Take (partial) derivatives of L_P w.r.t. $\mathbf{w}$, b, ξ_n and set them to zero

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n, \quad \frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0, \quad \frac{\partial L_P}{\partial \xi_n} = 0 \Rightarrow C - \alpha_n - \beta_n = 0$$

- Using $C - \alpha_n - \beta_n = 0$ and $\beta_n \geq 0 \Rightarrow \alpha_n \leq C$
- Substituting these in the Primal Lagrangian L_P gives the Dual Lagrangian

$$\text{Maximize} \quad L_D(\mathbf{w}, b, \xi, \alpha, \beta) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m,n=1}^{N} \alpha_m \alpha_n y_m y_n (\mathbf{x}_m^T \mathbf{x}_n)$$

$$\text{subject to} \quad \sum_{n=1}^{N} \alpha_n y_n = 0, \quad 0 \leq \alpha_n \leq C; \quad n = 1, \ldots, N$$

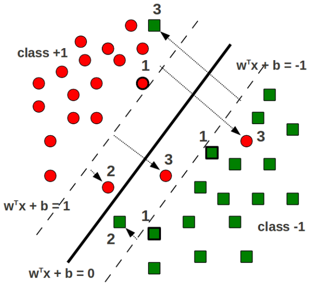
- Again a Quadratic Programming problem in α
- Given α, the solution for $\mathbf{w}, b$ has the same form as the separable case
- **Note:** α is again sparse. Nonzero α_n's correspond to the support vectors

Support Vectors in the non-separable case

- The separable case has only one type of support vectors
  - .. ones that lie on the margin boundaries $\mathbf{w}^T\mathbf{x} + b = -1$ and $\mathbf{w}^T\mathbf{x} + b = +1$

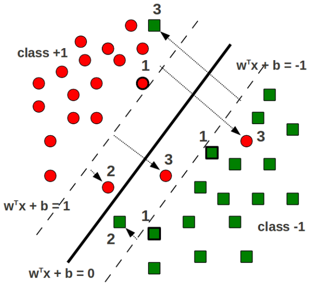
Support Vectors in the non-separable case

- The separable case has only one type of support vectors
  - .. ones that lie on the margin boundaries $\mathbf{w}^T\mathbf{x} + b = -1$ and $\mathbf{w}^T\mathbf{x} + b = +1$
- The non-separable case has three types of support vectors



Support Vectors in the non-separable case

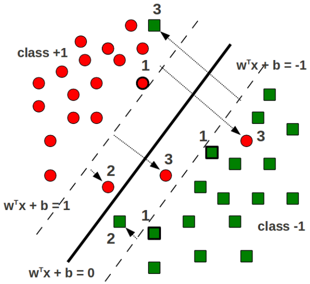
- The separable case has only one type of support vectors
  - .. ones that lie on the margin boundaries $\mathbf{w}^T\mathbf{x} + b = -1$ and $\mathbf{w}^T\mathbf{x} + b = +1$
- The non-separable case has three types of support vectors



1. Lying on the margin boundaries $\mathbf{w}^T\mathbf{x} + b = -1$ and $\mathbf{w}^T\mathbf{x} + b = +1$ ($\xi_n = 0$)

Support Vectors in the non-separable case

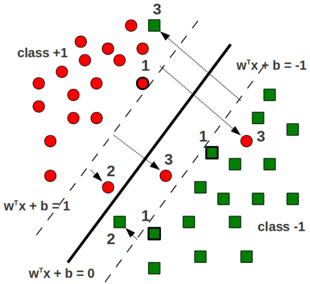
- The separable case has only one type of support vectors
  - .. ones that lie on the margin boundaries $\mathbf{w}^T\mathbf{x} + b = -1$ and $\mathbf{w}^T\mathbf{x} + b = +1$
- The non-separable case has three types of support vectors



1. Lying on the margin boundaries $\mathbf{w}^T\mathbf{x} + b = -1$ and $\mathbf{w}^T\mathbf{x} + b = +1$ ($\xi_n = 0$)
2. Lying within the margin region ($0 < \xi_n < 1$) but still on the correct side

Support Vectors in the non-separable case

- The separable case has only one type of support vectors
  - .. ones that lie on the margin boundaries $\mathbf{w}^T\mathbf{x} + b = -1$ and $\mathbf{w}^T\mathbf{x} + b = +1$
- The non-separable case has three types of support vectors



1. Lying on the margin boundaries $\mathbf{w}^T\mathbf{x} + b = -1$ and $\mathbf{w}^T\mathbf{x} + b = +1$ ($\xi_n = 0$)
2. Lying within the margin region ($0 < \xi_n < 1$) but still on the correct side
3. Lying on the wrong side of the hyperplane ($\xi_n \geq 1$)

Support Vector Machines: some notes

- Training time of the standard SVM is $O(N^3)$ (have to solve the QP)
  - Can be prohibitive for large datasets

Support Vector Machines: some notes

- Training time of the standard SVM is $O(N^3)$ (have to solve the QP)
  - Can be prohibitive for large datasets
- Lots of research has gone into speeding up the SVMs
  - Many approximate QP solvers are used to speed up SVMs
  - Online training (e.g., using stochastic gradient descent)

Support Vector Machines: some notes

- Training time of the standard SVM is $O(N^3)$ (have to solve the QP)
  - Can be prohibitive for large datasets
- Lots of research has gone into speeding up the SVMs
  - Many approximate QP solvers are used to speed up SVMs
  - Online training (e.g., using stochastic gradient descent)
- Several extensions exist

Support Vector Machines: some notes

- Training time of the standard SVM is $O(N^3)$ (have to solve the QP)
  - Can be prohibitive for large datasets
- Lots of research has gone into speeding up the SVMs
  - Many approximate QP solvers are used to speed up SVMs
  - Online training (e.g., using stochastic gradient descent)
- Several extensions exist
  - Nonlinear separation boundaries by applying the Kernel Trick (next class)

Support Vector Machines: some notes

- Training time of the standard SVM is $O(N^3)$ (have to solve the QP)
  - Can be prohibitive for large datasets
- Lots of research has gone into speeding up the SVMs
  - Many approximate QP solvers are used to speed up SVMs
  - Online training (e.g., using stochastic gradient descent)
- Several extensions exist
  - Nonlinear separation boundaries by applying the Kernel Trick (next class)
  - More than 2 classes (multiclass classification)

Support Vector Machines: some notes

- Training time of the standard SVM is $O(N^3)$ (have to solve the QP)
  - Can be prohibitive for large datasets
- Lots of research has gone into speeding up the SVMs
  - Many approximate QP solvers are used to speed up SVMs
  - Online training (e.g., using stochastic gradient descent)
- Several extensions exist
  - Nonlinear separation boundaries by applying the Kernel Trick (next class)
  - More than 2 classes (multiclass classification)
  - Structured outputs (structured prediction)

Support Vector Machines: some notes

- Training time of the standard SVM is $O(N^3)$ (have to solve the QP)
  - Can be prohibitive for large datasets
- Lots of research has gone into speeding up the SVMs
  - Many approximate QP solvers are used to speed up SVMs
  - Online training (e.g., using stochastic gradient descent)
- Several extensions exist
  - Nonlinear separation boundaries by applying the Kernel Trick (next class)
  - More than 2 classes (multiclass classification)
  - Structured outputs (structured prediction)
  - Real-valued outputs (support vector regression)

Support Vector Machines: some notes

- Training time of the standard SVM is $O(N^3)$ (have to solve the QP)
  - Can be prohibitive for large datasets
- Lots of research has gone into speeding up the SVMs
  - Many approximate QP solvers are used to speed up SVMs
  - Online training (e.g., using stochastic gradient descent)
- Several extensions exist
  - Nonlinear separation boundaries by applying the Kernel Trick (next class)
  - More than 2 classes (multiclass classification)
  - Structured outputs (structured prediction)
  - Real-valued outputs (support vector regression)
- Popular SVM implementations: libSVM, SVMLight, SVM-struct, etc.
  - Also `http://www.kernel-machines.org/software`

Loss Functions for Linear Classification

- We have seen two linear binary classification algorithms (Perceptron, SVM)

Loss Functions for Linear Classification

- We have seen two linear binary classification algorithms (Perceptron, SVM)
- Linear binary classification written as a general optimization problem:

$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T\mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

- $\mathbb{I}(.)$ is the indicator function (1 if (.) is true, 0 otherwise)

Loss Functions for Linear Classification

- We have seen two linear binary classification algorithms (Perceptron, SVM)
- Linear binary classification written as a general optimization problem:

$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T \mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

- $\mathbb{I}(.)$ is the indicator function (1 if (.) is true, 0 otherwise)
- The objective is sum of two parts: the loss function and the regularizer

Loss Functions for Linear Classification

- We have seen two linear binary classification algorithms (Perceptron, SVM)
- Linear binary classification written as a general optimization problem:

$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T\mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

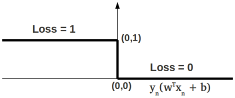
- $\mathbb{I}(.)$ is the indicator function (1 if (.) is true, 0 otherwise)
- The objective is sum of two parts: the loss function and the regularizer
  - Want to fit training data well and also want to have simple solutions

Loss Functions for Linear Classification

- We have seen two linear binary classification algorithms (Perceptron, SVM)
- Linear binary classification written as a general optimization problem:

$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T\mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

- $\mathbb{I}(.)$ is the indicator function (1 if (.) is true, 0 otherwise)
- The objective is sum of two parts: the loss function and the regularizer
  - Want to fit training data well and also want to have simple solutions
- The above loss function called the 0-1 loss

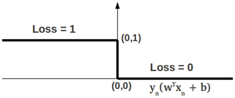


Loss Functions for Linear Classification

- We have seen two linear binary classification algorithms (Perceptron, SVM)
- Linear binary classification written as a general optimization problem:

$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T\mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

- $\mathbb{I}(.)$ is the indicator function (1 if (.) is true, 0 otherwise)
- The objective is sum of two parts: the loss function and the regularizer
  - Want to fit training data well and also want to have simple solutions
- The above loss function called the 0-1 loss



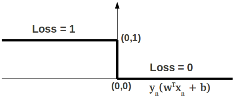
- The 0-1 loss is NP-hard to optimize (exactly/approximately) in general

Loss Functions for Linear Classification

- We have seen two linear binary classification algorithms (Perceptron, SVM)
- Linear binary classification written as a general optimization problem:

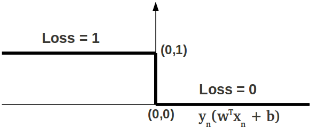
$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T\mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

- $\mathbb{I}(.)$ is the indicator function (1 if (.) is true, 0 otherwise)
- The objective is sum of two parts: the loss function and the regularizer
  - Want to fit training data well and also want to have simple solutions
- The above loss function called the 0-1 loss



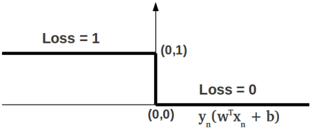
- The 0-1 loss is NP-hard to optimize (exactly/approximately) in general
- **Different loss function approximations and regularizers lead to specific algorithms** (e.g., Perceptron, SVM, Logistic Regression, etc.).

The 0-1 Loss



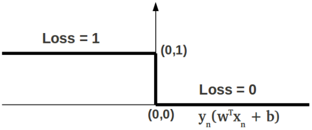
- It's a combinatorial optimization problem
- Can be shown to be NP-hard
  - .. using a reduction of a variant of the satisfiability problem

The 0-1 Loss



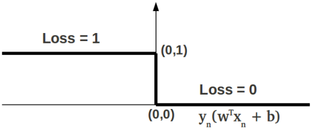
- It's a combinatorial optimization problem
- Can be shown to be NP-hard
  - .. using a reduction of a variant of the satisfiability problem
- No polynomial time algorithm

The 0-1 Loss



- It's a combinatorial optimization problem
- Can be shown to be NP-hard
  - .. using a reduction of a variant of the satisfiability problem
- No polynomial time algorithm
- Loss function is non-smooth, non-convex

The 0-1 Loss



- It's a combinatorial optimization problem
- Can be shown to be NP-hard
  - .. using a reduction of a variant of the satisfiability problem
- No polynomial time algorithm
- Loss function is non-smooth, non-convex
- Small changes in $\mathbf{w}$, b can change the loss by a lot

Approximations to the 0-1 loss

- We use loss functions that are convex approximations to the 0-1 loss
  - These are called surrogate loss functions

Approximations to the 0-1 loss

- We use loss functions that are convex approximations to the 0-1 loss
  - These are called surrogate loss functions
- Examples of surrogate loss functions (assuming $b = 0$):
  - Hinge loss: $[1 - y_n \mathbf{w}^T \mathbf{x}_n]_+ = \max\{0, 1 - y_n \mathbf{w}^T \mathbf{x}_n\}$

Approximations to the 0-1 loss

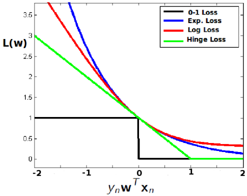
- We use loss functions that are convex approximations to the 0-1 loss
  - These are called surrogate loss functions
- Examples of surrogate loss functions (assuming $b = 0$):
  - Hinge loss: $[1 - y_n \mathbf{w}^T \mathbf{x}_n]_+ = \max\{0, 1 - y_n \mathbf{w}^T \mathbf{x}_n\}$
  - Log loss: $\log[1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n)]$

Approximations to the 0-1 loss

- We use loss functions that are convex approximations to the 0-1 loss
  - These are called surrogate loss functions
- Examples of surrogate loss functions (assuming $b = 0$):
  - Hinge loss: $[1 - y_n \mathbf{w}^T \mathbf{x}_n]_+ = \max\{0, 1 - y_n \mathbf{w}^T \mathbf{x}_n\}$
  - Log loss: $\log[1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n)]$
  - Exponential loss: $\exp(-y_n \mathbf{w}^T \mathbf{x}_n)$

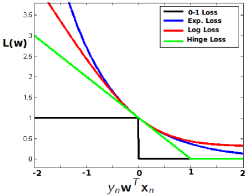
Approximations to the 0-1 loss

- We use loss functions that are convex approximations to the 0-1 loss
  - These are called surrogate loss functions
- Examples of surrogate loss functions (assuming $b = 0$):
  - Hinge loss: $[1 - y_n \mathbf{w}^T \mathbf{x}_n]_+ = \max\{0, 1 - y_n \mathbf{w}^T \mathbf{x}_n\}$
  - Log loss: $\log[1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n)]$
  - Exponential loss: $\exp(-y_n \mathbf{w}^T \mathbf{x}_n)$
  - All are convex upper bounds on the 0-1 loss
  - Minimizing a convex upper bound also pushes down the original function
  - Unlike 0-1 loss, these loss functions depend on how far the examples are from the hyperplane



Approximations to the 0-1 loss

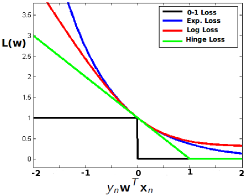
- We use loss functions that are convex approximations to the 0-1 loss
  - These are called surrogate loss functions
- Examples of surrogate loss functions (assuming $b = 0$):
  - Hinge loss: $[1 - y_n \mathbf{w}^T \mathbf{x}_n]_+ = \max\{0, 1 - y_n \mathbf{w}^T \mathbf{x}_n\}$
  - Log loss: $\log[1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n)]$
  - Exponential loss: $\exp(-y_n \mathbf{w}^T \mathbf{x}_n)$
  - All are convex upper bounds on the 0-1 loss
  - Minimizing a convex upper bound also pushes down the original function
  - Unlike 0-1 loss, these loss functions depend on how far the examples are from the hyperplane



- Apart from convexity, smoothness is the other desirable for loss functions

Approximations to the 0-1 loss

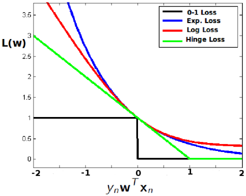
- We use loss functions that are convex approximations to the 0-1 loss
  - These are called surrogate loss functions
- Examples of surrogate loss functions (assuming $b = 0$):
  - Hinge loss: $[1 - y_n \mathbf{w}^T \mathbf{x}_n]_+ = \max\{0, 1 - y_n \mathbf{w}^T \mathbf{x}_n\}$
  - Log loss: $\log[1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n)]$
  - Exponential loss: $\exp(-y_n \mathbf{w}^T \mathbf{x}_n)$
  - All are convex upper bounds on the 0-1 loss
  - Minimizing a convex upper bound also pushes down the original function
  - Unlike 0-1 loss, these loss functions depend on how far the examples are from the hyperplane



- Apart from convexity, smoothness is the other desirable for loss functions
  - Smoothness allows using gradient (or stochastic gradient) descent

Approximations to the 0-1 loss

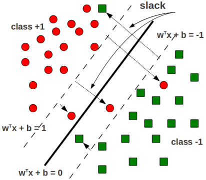
- We use loss functions that are convex approximations to the 0-1 loss
  - These are called surrogate loss functions
- Examples of surrogate loss functions (assuming $b = 0$):
  - Hinge loss: $[1 - y_n \mathbf{w}^T \mathbf{x}_n]_+ = \max\{0, 1 - y_n \mathbf{w}^T \mathbf{x}_n\}$
  - Log loss: $\log[1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n)]$
  - Exponential loss: $\exp(-y_n \mathbf{w}^T \mathbf{x}_n)$
  - All are convex upper bounds on the 0-1 loss
  - Minimizing a convex upper bound also pushes down the original function
  - Unlike 0-1 loss, these loss functions depend on how far the examples are from the hyperplane



- Apart from convexity, smoothness is the other desirable for loss functions
  - Smoothness allows using gradient (or stochastic gradient) descent
  - Note: hinge loss is not smooth at (1,0) but subgradient descent can be used

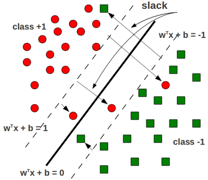
Loss functions for specific algorithms

- Recall **SVM** non-separable case: we minimized the sum of slacks $\sum_{n=1}^{N} \xi_n$



Loss functions for specific algorithms

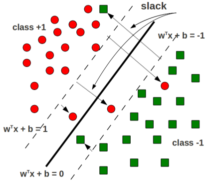
- Recall **SVM** non-separable case: we minimized the sum of slacks $\sum_{n=1}^{N} \xi_n$



  - No penalty ($\xi_n = 0$) if $y_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1$
  - Linear penalty ($\xi_n = 1 - y_n(\mathbf{w}^T \mathbf{x}_n + b)$) if $y_n(\mathbf{w}^T \mathbf{x}_n + b) < 1$

Loss functions for specific algorithms

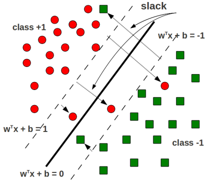
- Recall **SVM** non-separable case: we minimized the sum of slacks $\sum_{n=1}^{N} \xi_n$



  - No penalty ($\xi_n = 0$) if $y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1$
  - Linear penalty ($\xi_n = 1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)$) if $y_n(\mathbf{w}^T\mathbf{x}_n + b) < 1$
  - It's precisely the hinge loss $\max\{0, 1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)\}$

Loss functions for specific algorithms

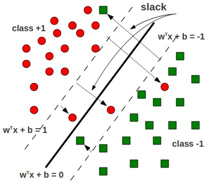
- Recall **SVM** non-separable case: we minimized the sum of slacks $\sum_{n=1}^{N} \xi_n$



  - No penalty ($\xi_n = 0$) if $y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1$
  - Linear penalty ($\xi_n = 1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)$) if $y_n(\mathbf{w}^T\mathbf{x}_n + b) < 1$
  - It's precisely the hinge loss $\max\{0, 1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)\}$
  - Note: Some SVMs minimize the sum of **squared** slacks $\sum_{n=1}^{N} \xi_n^2$

Loss functions for specific algorithms

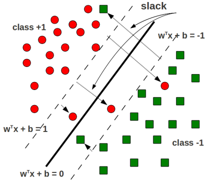
- Recall **SVM** non-separable case: we minimized the sum of slacks $\sum_{n=1}^{N} \xi_n$



  - No penalty ($\xi_n = 0$) if $y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1$
  - Linear penalty ($\xi_n = 1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)$) if $y_n(\mathbf{w}^T\mathbf{x}_n + b) < 1$
  - It's precisely the hinge loss $\max\{0, 1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)\}$
  - Note: Some SVMs minimize the sum of **squared** slacks $\sum_{n=1}^{N} \xi_n^2$
- **Perceptron** uses a variant of the hinge loss: $\max\{0, -y_n(\mathbf{w}^T\mathbf{x}_n + b)\}$

Loss functions for specific algorithms

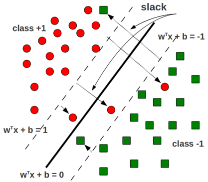
- Recall **SVM** non-separable case: we minimized the sum of slacks $\sum_{n=1}^{N} \xi_n$



  - No penalty ($\xi_n = 0$) if $y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1$
  - Linear penalty ($\xi_n = 1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)$) if $y_n(\mathbf{w}^T\mathbf{x}_n + b) < 1$
  - It's precisely the hinge loss $\max\{0, 1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)\}$
  - Note: Some SVMs minimize the sum of **squared** slacks $\sum_{n=1}^{N} \xi_n^2$
- **Perceptron** uses a variant of the hinge loss: $\max\{0, -y_n(\mathbf{w}^T\mathbf{x}_n + b)\}$
- **Logistic Regression** uses the log loss
  - Misnomer: Logistic Regression does classification, not regression!

Loss functions for specific algorithms

- Recall **SVM** non-separable case: we minimized the sum of slacks $\sum_{n=1}^{N} \xi_n$



  - No penalty ($\xi_n = 0$) if $y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1$
  - Linear penalty ($\xi_n = 1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)$) if $y_n(\mathbf{w}^T\mathbf{x}_n + b) < 1$
  - It's precisely the hinge loss $\max\{0, 1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)\}$
  - Note: Some SVMs minimize the sum of **squared** slacks $\sum_{n=1}^{N} \xi_n^2$
- **Perceptron** uses a variant of the hinge loss: $\max\{0, -y_n(\mathbf{w}^T\mathbf{x}_n + b)\}$
- **Logistic Regression** uses the log loss
  - Misnomer: Logistic Regression does classification, not regression!
- **Boosting** uses the exponential loss

Regularizers

- Recall: The optimization problem for regularized linear binary classification:

$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T \mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

Regularizers

- Recall: The optimization problem for regularized linear binary classification:

$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T \mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

- We have already seen the approximation choices for the 0-1 loss function

Regularizers

- Recall: The optimization problem for regularized linear binary classification:

$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T\mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

- We have already seen the approximation choices for the 0-1 loss function
- What about the regularizer term $R(\mathbf{w}, b)$ that *ensures simple solutions*?

Regularizers

- Recall: The optimization problem for regularized linear binary classification:

$$\min_{\mathbf{w},b} L(\mathbf{w},b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T\mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w},b)$$

- We have already seen the approximation choices for the 0-1 loss function
- What about the regularizer term $R(\mathbf{w},b)$ that *ensures simple solutions*?
- The regularizer $R(\mathbf{w},b)$ determines what each entry w_d of $\mathbf{w}$ looks like

Regularizers

- Recall: The optimization problem for regularized linear binary classification:

$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T\mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

- We have already seen the approximation choices for the 0-1 loss function
- What about the regularizer term $R(\mathbf{w}, b)$ that *ensures simple solutions*?
- The regularizer $R(\mathbf{w}, b)$ determines what each entry w_d of $\mathbf{w}$ looks like
- Ideally, we want most entries w_d of $\mathbf{w}$ be zero

Regularizers

- Recall: The optimization problem for regularized linear binary classification:

$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T \mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

- We have already seen the approximation choices for the 0-1 loss function
- What about the regularizer term $R(\mathbf{w}, b)$ that *ensures simple solutions*?
- The regularizer $R(\mathbf{w}, b)$ determines what each entry w_d of $\mathbf{w}$ looks like
- Ideally, we want most entries w_d of $\mathbf{w}$ be zero , so prediction depends only on a small number of features (for which $w_d \neq 0$).

Regularizers

- Recall: The optimization problem for regularized linear binary classification:

$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T \mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

- We have already seen the approximation choices for the 0-1 loss function
- What about the regularizer term $R(\mathbf{w}, b)$ that *ensures simple solutions*?
- The regularizer $R(\mathbf{w}, b)$ determines what each entry w_d of $\mathbf{w}$ looks like
- Ideally, we want most entries w_d of $\mathbf{w}$ be zero , so prediction depends only on a small number of features (for which $w_d \neq 0$). Desired minimization:

$$R^{cnt}(\mathbf{w}, b) = \sum_{d=1}^{D} \mathbb{I}(w_d \neq 0)$$

Regularizers

- Recall: The optimization problem for regularized linear binary classification:

$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T\mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

- We have already seen the approximation choices for the 0-1 loss function
- What about the regularizer term $R(\mathbf{w}, b)$ that *ensures simple solutions*?
- The regularizer $R(\mathbf{w}, b)$ determines what each entry w_d of $\mathbf{w}$ looks like
- Ideally, we want most entries w_d of $\mathbf{w}$ be zero , so prediction depends only on a small number of features (for which $w_d \neq 0$). Desired minimization:

$$R^{cnt}(\mathbf{w}, b) = \sum_{d=1}^{D} \mathbb{I}(w_d \neq 0)$$

- $R^{cnt}(\mathbf{w}, b)$ is NP-hard to minimize, so its approximations are used

Regularizers

- Recall: The optimization problem for regularized linear binary classification:

$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T\mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

- We have already seen the approximation choices for the 0-1 loss function
- What about the regularizer term $R(\mathbf{w}, b)$ that *ensures simple solutions*?
- The regularizer $R(\mathbf{w}, b)$ determines what each entry w_d of **w** looks like
- Ideally, we want most entries w_d of **w** be zero , so prediction depends only on a small number of features (for which $w_d \neq 0$). Desired minimization:

$$R^{cnt}(\mathbf{w}, b) = \sum_{d=1}^{D} \mathbb{I}(w_d \neq 0)$$

- $R^{cnt}(\mathbf{w}, b)$ is NP-hard to minimize, so its approximations are used
  - A good approximation is to make the individual w_d's small

Regularizers

- Recall: The optimization problem for regularized linear binary classification:

$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T\mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

- We have already seen the approximation choices for the 0-1 loss function
- What about the regularizer term $R(\mathbf{w}, b)$ that *ensures simple solutions*?
- The regularizer $R(\mathbf{w}, b)$ determines what each entry w_d of $\mathbf{w}$ looks like
- Ideally, we want most entries w_d of $\mathbf{w}$ be zero , so prediction depends only on a small number of features (for which $w_d \neq 0$). Desired minimization:

$$R^{cnt}(\mathbf{w}, b) = \sum_{d=1}^{D} \mathbb{I}(w_d \neq 0)$$

- $R^{cnt}(\mathbf{w}, b)$ is NP-hard to minimize, so its approximations are used
  - A good approximation is to make the individual w_d's small
  - Small $w_d \Rightarrow$ small changes in some feature x_d won't affect prediction by much

Regularizers

- Recall: The optimization problem for regularized linear binary classification:

$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T\mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

- We have already seen the approximation choices for the 0-1 loss function
- What about the regularizer term $R(\mathbf{w}, b)$ that *ensures simple solutions*?
- The regularizer $R(\mathbf{w}, b)$ determines what each entry w_d of $\mathbf{w}$ looks like
- Ideally, we want most entries w_d of $\mathbf{w}$ be zero , so prediction depends only on a small number of features (for which $w_d \neq 0$). Desired minimization:

$$R^{cnt}(\mathbf{w}, b) = \sum_{d=1}^{D} \mathbb{I}(w_d \neq 0)$$

- $R^{cnt}(\mathbf{w}, b)$ is NP-hard to minimize, so its approximations are used
  - A good approximation is to make the individual w_d's small
  - Small $w_d \Rightarrow$ small changes in some feature x_d won't affect prediction by much
  - Small individual weights w_d is a notion of function simplicity

Norm based Regularizers

- Norm based regularizers are used as approximations to $R^{cnt}(\mathbf{w}, b)$

Norm based Regularizers

- Norm based regularizers are used as approximations to $R^{cnt}(\mathbf{w}, b)$
  - ℓ_2 squared norm: $||\mathbf{w}||_2^2 = \sum_{d=1}^{D} w_d^2$

Norm based Regularizers

- Norm based regularizers are used as approximations to $R^{cnt}(\mathbf{w}, b)$
    - ℓ_2 squared norm: $||\mathbf{w}||_2^2 = \sum_{d=1}^{D} w_d^2$
    - ℓ_1 norm: $||\mathbf{w}||_1 = \sum_{d=1}^{D} |w_d|$

Norm based Regularizers

- Norm based regularizers are used as approximations to $R^{cnt}(\mathbf{w}, b)$
  - ℓ_2 squared norm: $||\mathbf{w}||_2^2 = \sum_{d=1}^{D} w_d^2$
  - ℓ_1 norm: $||\mathbf{w}||_1 = \sum_{d=1}^{D} |w_d|$
  - ℓ_p norm: $||\mathbf{w}||_p = (\sum_{d=1}^{D} w_d^p)^{1/p}$

Norm based Regularizers

- Norm based regularizers are used as approximations to $R^{cnt}(\mathbf{w}, b)$
  - ℓ_2 squared norm: $||\mathbf{w}||_2^2 = \sum_{d=1}^{D} w_d^2$
  - ℓ_1 norm: $||\mathbf{w}||_1 = \sum_{d=1}^{D} |w_d|$
  - ℓ_p norm: $||\mathbf{w}||_p = (\sum_{d=1}^{D} w_d^p)^{1/p}$

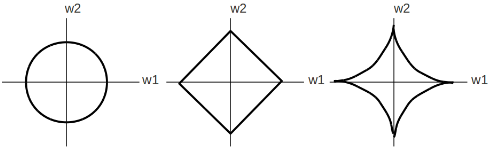


Figure: Contour plots. Left: ℓ_2 norm, Center: ℓ_1 norm, Right: ℓ_p norm (for $p < 1$)

Norm based Regularizers

- Norm based regularizers are used as approximations to $R^{cnt}(\mathbf{w}, b)$
  - ℓ_2 squared norm: $||\mathbf{w}||_2^2 = \sum_{d=1}^{D} w_d^2$
  - ℓ_1 norm: $||\mathbf{w}||_1 = \sum_{d=1}^{D} |w_d|$
  - ℓ_p norm: $||\mathbf{w}||_p = (\sum_{d=1}^{D} w_d^p)^{1/p}$

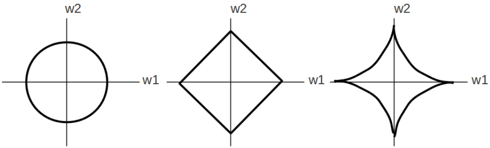


Figure: Contour plots. Left: ℓ_2 norm, Center: ℓ_1 norm, Right: ℓ_p norm (for $p < 1$)

- Smaller p favors sparser vector $\mathbf{w}$ (most entries of $\mathbf{w}$ close/equal to 0)

Norm based Regularizers

- Norm based regularizers are used as approximations to $R^{cnt}(\mathbf{w}, b)$
  - ℓ_2 squared norm: $||\mathbf{w}||_2^2 = \sum_{d=1}^{D} w_d^2$
  - ℓ_1 norm: $||\mathbf{w}||_1 = \sum_{d=1}^{D} |w_d|$
  - ℓ_p norm: $||\mathbf{w}||_p = (\sum_{d=1}^{D} w_d^p)^{1/p}$

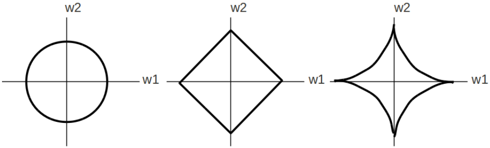


Figure: Contour plots. Left: ℓ_2 norm, Center: ℓ_1 norm, Right: ℓ_p norm (for $p < 1$)

- Smaller p favors sparser vector $\mathbf{w}$ (most entries of $\mathbf{w}$ close/equal to 0)
  - But the norm becomes non-convex for $p < 1$ and is **hard to optimize**

Norm based Regularizers

- Norm based regularizers are used as approximations to $R^{cnt}(\mathbf{w}, b)$
  - ℓ_2 squared norm: $||\mathbf{w}||_2^2 = \sum_{d=1}^{D} w_d^2$
  - ℓ_1 norm: $||\mathbf{w}||_1 = \sum_{d=1}^{D} |w_d|$
  - ℓ_p norm: $||\mathbf{w}||_p = (\sum_{d=1}^{D} w_d^p)^{1/p}$

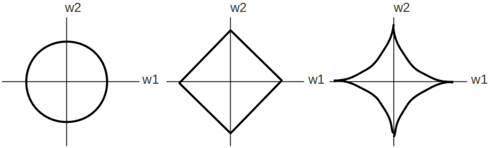


Figure: Contour plots. Left: ℓ_2 norm, Center: ℓ_1 norm, Right: ℓ_p norm (for $p < 1$)

- Smaller p favors sparser vector $\mathbf{w}$ (most entries of $\mathbf{w}$ close/equal to 0)
  - But the norm becomes non-convex for $p < 1$ and is **hard to optimize**
- The ℓ_1 norm is the most preferred regularizer for sparse $\mathbf{w}$ (many w_d's zero)

Norm based Regularizers

- Norm based regularizers are used as approximations to $R^{cnt}(\mathbf{w}, b)$
  - ℓ_2 squared norm: $||\mathbf{w}||_2^2 = \sum_{d=1}^{D} w_d^2$
  - ℓ_1 norm: $||\mathbf{w}||_1 = \sum_{d=1}^{D} |w_d|$
  - ℓ_p norm: $||\mathbf{w}||_p = (\sum_{d=1}^{D} w_d^p)^{1/p}$

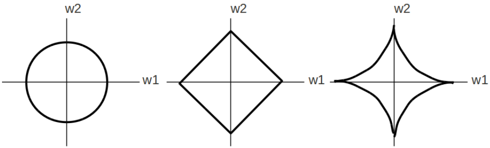


Figure: Contour plots. Left: ℓ_2 norm, Center: ℓ_1 norm, Right: ℓ_p norm (for $p < 1$)

- Smaller p favors sparser vector $\mathbf{w}$ (most entries of $\mathbf{w}$ close/equal to 0)
  - But the norm becomes non-convex for $p < 1$ and is **hard to optimize**
- The ℓ_1 norm is the most preferred regularizer for sparse $\mathbf{w}$ (many w_d's zero)
  - Convex, but it's not smooth at the axis points

Norm based Regularizers

- Norm based regularizers are used as approximations to $R^{cnt}(\mathbf{w}, b)$
  - ℓ_2 squared norm: $||\mathbf{w}||_2^2 = \sum_{d=1}^{D} w_d^2$
  - ℓ_1 norm: $||\mathbf{w}||_1 = \sum_{d=1}^{D} |w_d|$
  - ℓ_p norm: $||\mathbf{w}||_p = (\sum_{d=1}^{D} w_d^p)^{1/p}$

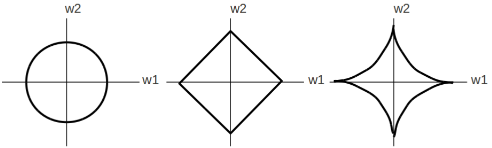


Figure: Contour plots. Left: ℓ_2 norm, Center: ℓ_1 norm, Right: ℓ_p norm (for $p < 1$)

- Smaller p favors sparser vector $\mathbf{w}$ (most entries of $\mathbf{w}$ close/equal to 0)
  - But the norm becomes non-convex for $p < 1$ and is **hard to optimize**
- The ℓ_1 norm is the most preferred regularizer for sparse $\mathbf{w}$ (many w_d's zero)
  - Convex, but it's not smooth at the axis points
  - .. but several methods exists to deal with it, e.g., subgradient descent

Norm based Regularizers

- Norm based regularizers are used as approximations to $R^{cnt}(\mathbf{w}, b)$
  - ℓ_2 squared norm: $||\mathbf{w}||_2^2 = \sum_{d=1}^{D} w_d^2$
  - ℓ_1 norm: $||\mathbf{w}||_1 = \sum_{d=1}^{D} |w_d|$
  - ℓ_p norm: $||\mathbf{w}||_p = (\sum_{d=1}^{D} w_d^p)^{1/p}$

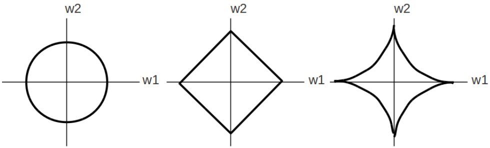


Figure: Contour plots. Left: ℓ_2 norm, Center: ℓ_1 norm, Right: ℓ_p norm (for $p < 1$)

- Smaller p favors sparser vector $\mathbf{w}$ (most entries of $\mathbf{w}$ close/equal to 0)
  - But the norm becomes non-convex for $p < 1$ and is **hard to optimize**
- The ℓ_1 norm is the most preferred regularizer for sparse $\mathbf{w}$ (many w_d's zero)
  - Convex, but it's not smooth at the axis points
  - .. but several methods exists to deal with it, e.g., subgradient descent
- The ℓ_2 squared norm tries to keep the individual w_d's small

Norm based Regularizers

- Norm based regularizers are used as approximations to $R^{cnt}(\mathbf{w}, b)$
  - ℓ_2 squared norm: $||\mathbf{w}||_2^2 = \sum_{d=1}^{D} w_d^2$
  - ℓ_1 norm: $||\mathbf{w}||_1 = \sum_{d=1}^{D} |w_d|$
  - ℓ_p norm: $||\mathbf{w}||_p = (\sum_{d=1}^{D} w_d^p)^{1/p}$

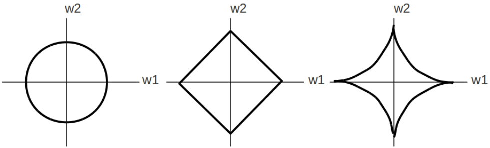


Figure: Contour plots. Left: ℓ_2 norm, Center: ℓ_1 norm, Right: ℓ_p norm (for $p < 1$)

- Smaller p favors sparser vector $\mathbf{w}$ (most entries of $\mathbf{w}$ close/equal to 0)
  - But the norm becomes non-convex for $p < 1$ and is **hard to optimize**
- The ℓ_1 norm is the most preferred regularizer for sparse $\mathbf{w}$ (many w_d's zero)
  - Convex, but it's not smooth at the axis points
  - .. but several methods exists to deal with it, e.g., subgradient descent
- The ℓ_2 squared norm tries to keep the individual w_d's small
  - Convex, smooth, and the easiest to deal with

Next class..

- Introduction to Kernels
- Nonlinear classification algorithms
  - Kernelized Perceptron
  - Kernelized Support Vector Machines