Binary Classification with Linear Models

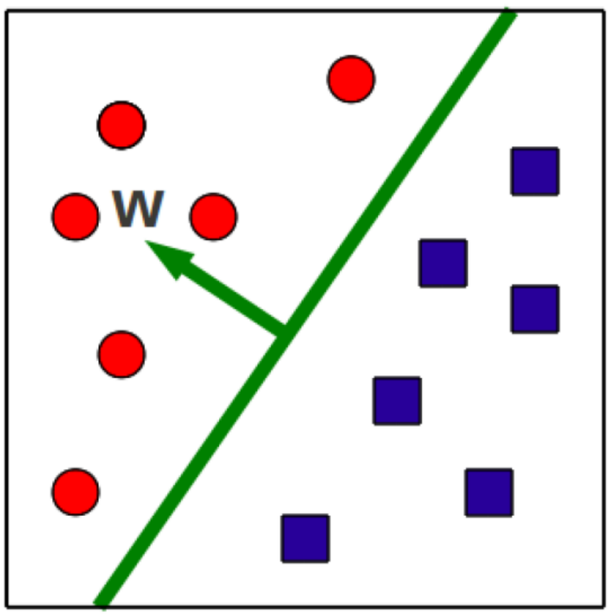
CMSC 422

SOHEIL FEIZI

sfeizi@cs.umd.edu

Slides adapted from MARINE CARPUAT

Binary classification via hyperplanes



- A classifier is a hyperplane (w,b)
- At test time, we check on what side of the hyperplane examples fall

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

- This is a **linear classifier**
  - Because the prediction is a linear combination of feature values x

Task: Binary Classification

Given:

1. An input space $\mathcal{X}$

2. An unknown distribution $\mathcal{D}$ over $\mathcal{X} \times \{-1, +1\}$

Compute: A function f minimizing: $\mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\left[f(\boldsymbol{x}) \neq y\right]$

Learning a Linear Classifier as an Optimization Problem

Objective function

Loss function measures how well classifier fits training data

Regularizer prefers solutions that generalize well

$$\min_{\mathbf{w}, b} L(\mathbf{w}, b)$$

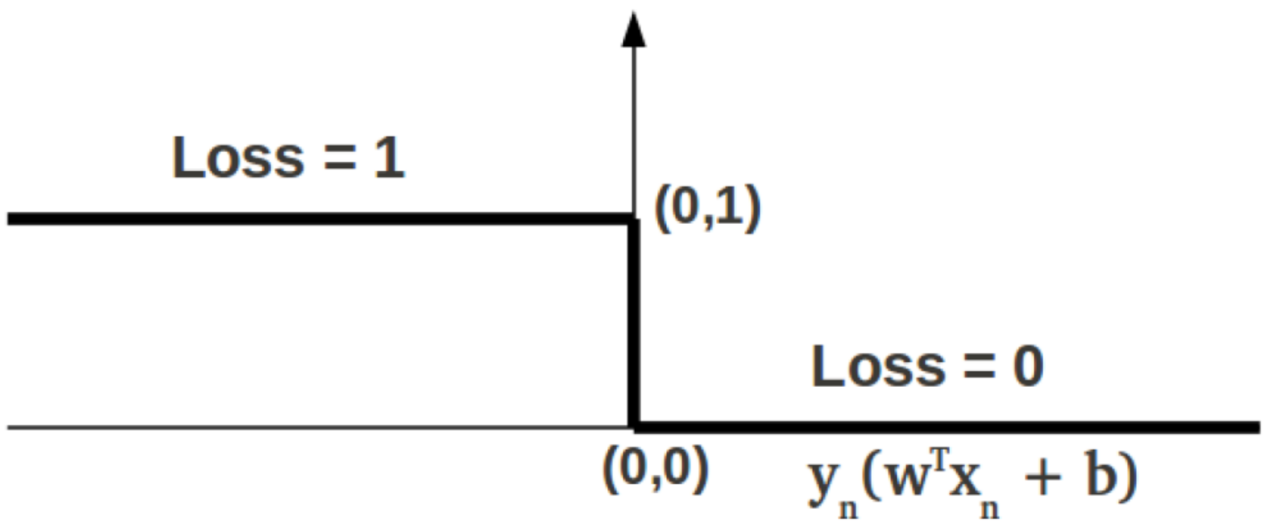
Learning a Linear Classifier as an Optimization Problem

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

- **Problem:** The 0-1 loss above is NP-hard to optimize exactly/approximately in general

- **Solution:** Different loss function approximations and regularizers lead to specific algorithms
  (e.g., perceptron, support vector machines, logistic regression, etc.)

The 0-1 Loss

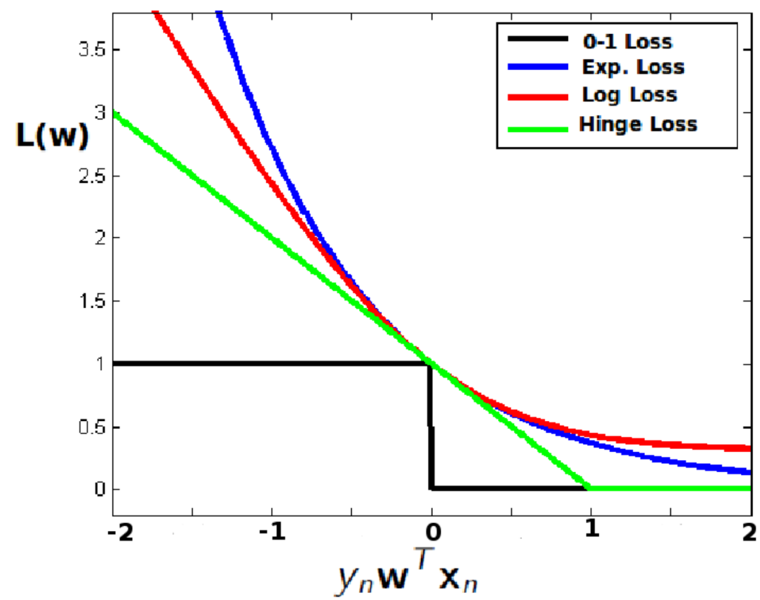


- Small changes in w,b can lead to big changes in the loss value
- 0-1 loss is non-smooth, non-convex

Approximating the 0-1 loss with surrogate loss functions

- Examples (with 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 $\exp(-y_n \mathbf{w}^T \mathbf{x}_n)$

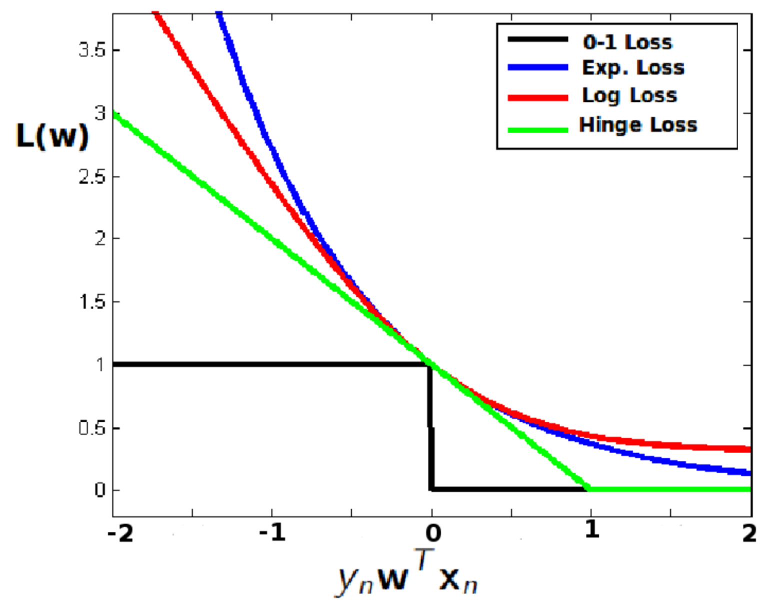
- All are convex upper-bounds on the 0-1 loss



Approximating the 0-1 loss with surrogate loss functions

- Examples (with 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 $\exp(-y_n \mathbf{w}^T \mathbf{x}_n)$

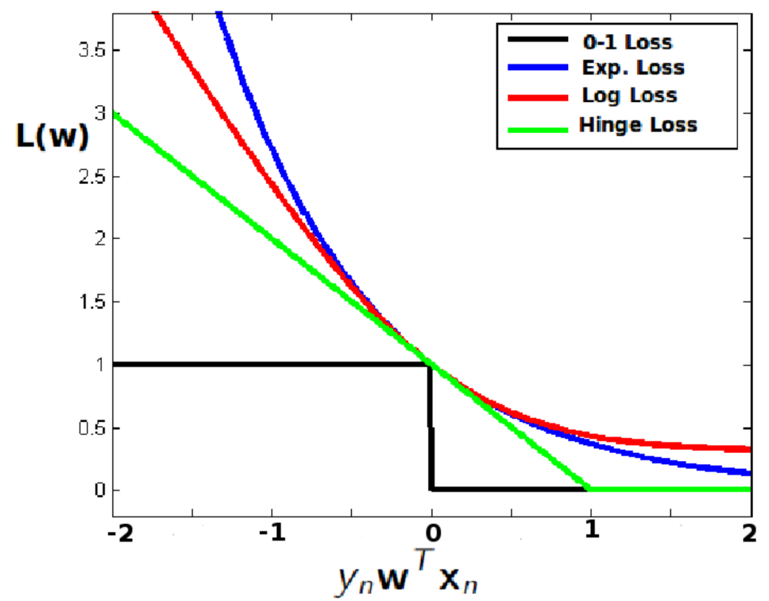
- **Q: Which of these loss functions is not smooth?**



Approximating the 0-1 loss with surrogate loss functions

- Examples (with 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 $\exp(-y_n \mathbf{w}^T \mathbf{x}_n)$

- **Q: Which of these loss functions is most sensitive to outliers?**



Casting Linear Classification as an Optimization Problem

Objective function

Loss function measures how well classifier fits training data

Regularizer prefers solutions that generalize well

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

$\mathbb{I}(.)$ Indicator function: 1 if (.) is true, 0 otherwise

The loss function above is called the 0-1 loss

The regularizer term

- Goal: find simple solutions (inductive bias)

- Ideally, we want most entries of w to be zero, so prediction depends only on a small number of features.

- Formally, we want to minimize:

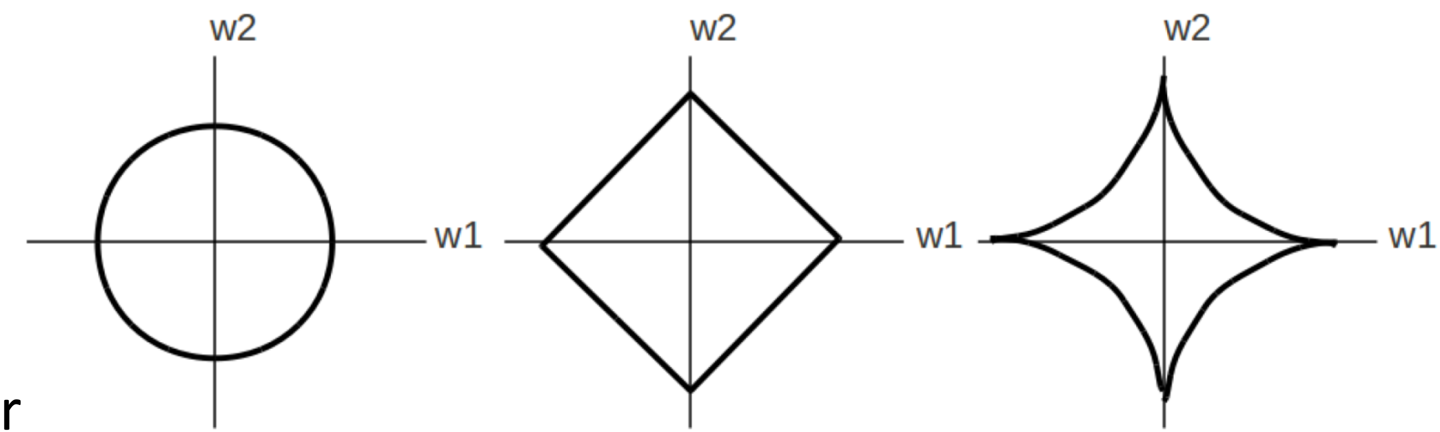
$$R^{cnt}(\mathbf{w}, b) = \sum_{d=1}^{D} \mathbb{I}(w_d \neq 0)$$

- That's NP-hard, so we use approximations instead.
  - E.g., we encourage w_d's to be small

Norm-based Regularizers

- l_p norms can be used as regularizers

$$||\mathbf{w}||_2^2 = \sum_{d=1}^{D} w_d^2$$
$$||\mathbf{w}||_1 = \sum_{d=1}^{D} |w_d|$$
$$||\mathbf{w}||_p = (\sum_{d=1}^{D} w_d^p)^{1/p}$$



Contour plots for p = 2 p = 1 p < 1

Norm-based Regularizers

- l_p norms can be used as regularizers
- Smaller p favors sparse vectors w
  - i.e. most entries of w are close or equal to 0

- l_2 norm: convex, smooth, easy to optimize
- l_1 norm: encourages sparse w, convex, but not smooth at axis points
- $p < 1$: norm becomes non convex and hard to optimize

Casting Linear Classification as an Optimization Problem

Objective function

Loss function measures how well classifier fits training data

Regularizer prefers solutions that generalize well

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

$\mathbb{I}(.)$ Indicator function: 1 if (.) is true, 0 otherwise

The loss function above is called the 0-1 loss

What is the perceptron optimizing?

Algorithm 5 PerceptronTrain($\mathbf{D}$, $MaxIter$)

```
w_d ← 0, for all d = 1 ... D                          // initialize weights
b ← 0                                                  // initialize bias
for iter = 1 ... MaxIter do
    for all (x,y) ∈ D do
        a ← Σ_{d=1}^{D} w_d x_d + b                    // compute activation for this example
        if ya ≤ 0 then
            w_d ← w_d + y x_d, for all d = 1 ... D     // update weights
            b ← b + y                                  // update bias
        end if
    end for
end for
return w_0, w_1, ..., w_D, b
```

- Loss function is a variant of the hinge loss

$$\max\{0, -y_n(\mathbf{w}^T\mathbf{x}_n + b)\}$$

Gradient descent

- A general solution for our optimization problem

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

Idea: take iterative steps to update parameters in the direction of the gradient

Gradient descent algorithm

Objective function to minimize

Number of steps

Step size

Algorithm 22 $\textsc{GradientDescent}(\mathcal{F}, K, \eta_1, \ldots)$

```
z^(0) ← ⟨0, 0, ..., 0⟩                      // initialize variable we are optimizing
for k = 1 ... K do
    g^(k) ← ∇_z F |_{z^(k-1)}                // compute gradient at current location
    z^(k) ← z^(k-1) − η^(k) g^(k)            // take a step down the gradient
end for
return z^(K)
```

$z^{(0)} \leftarrow \langle 0, 0, \ldots, 0 \rangle$ // initialize variable we are optimizing

for $k = 1 \ldots K$ **do**

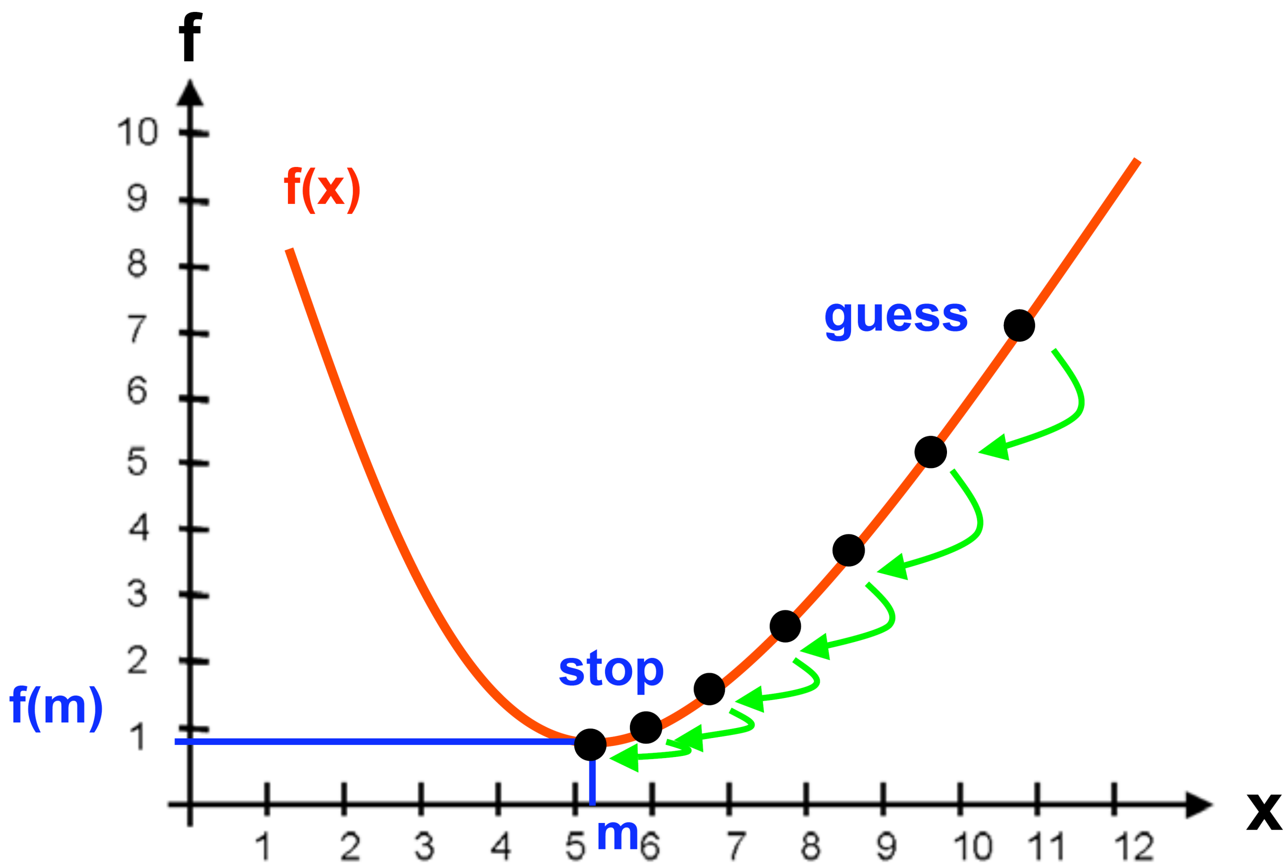
$\quad g^{(k)} \leftarrow \nabla_z \mathcal{F}|_{z^{(k-1)}}$ // compute gradient at current location

$\quad z^{(k)} \leftarrow z^{(k-1)} - \eta^{(k)} g^{(k)}$ // take a step down the gradient

end for

return $z^{(K)}$

Illustrating gradient descent in 1-dimensional case



Recap: Linear Models

- General framework for binary classification
- Cast learning as optimization problem
- Optimization objective combines 2 terms
  - loss function: measures how well classifier fits training data
  - Regularizer: measures how simple classifier is
- Does not assume data is linearly separable
- Lets us separate model definition from training algorithm **(Gradient Descent)**