Slides adapted from Prof Carpuat and Duraiswami

CMSC 422 Introduction to Machine Learning

Lecture 13 Binary Classification with Linear Models

Furong Huang / furongh@cs.umd.edu

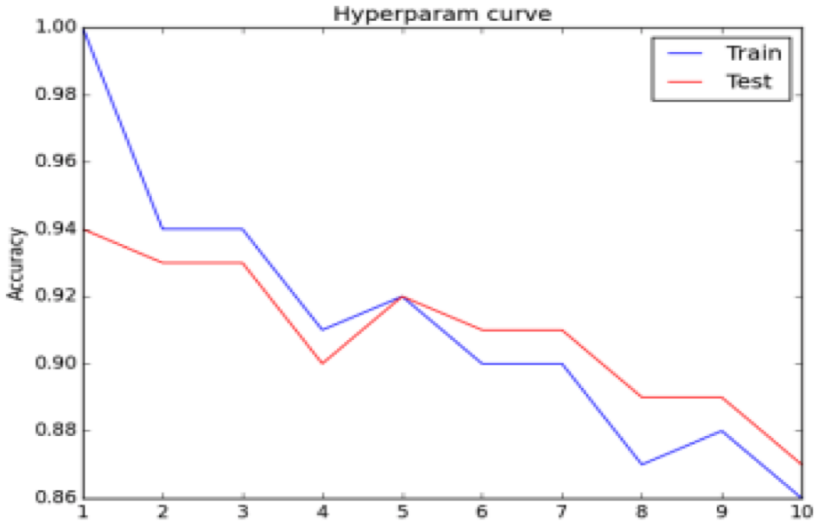
UNIVERSITY OF MARYLAND

Project 1 – regrading requests on piazza

- all grade requests submitted by Friday will be handled by Friday evening
- requests submitted during spring break or the week after will be handled daily starting the Monday after spring break
- All grade requests must be submitted by the Thursday after Spring Break.

Common misunderstandings - I

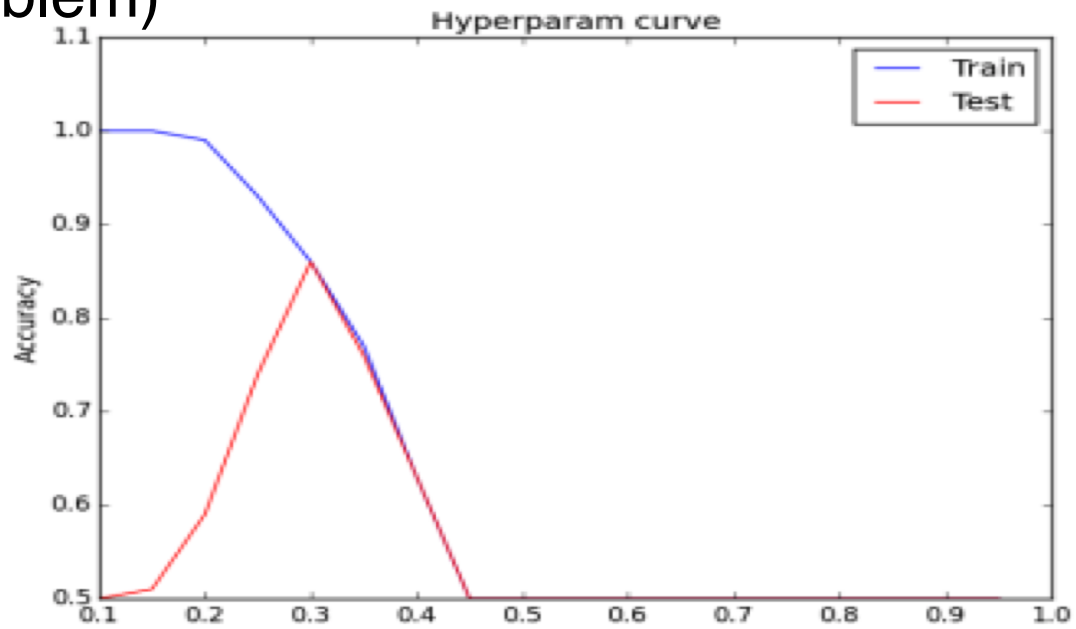
Misunderstanding I: The classifier is best when performance on test and training are equal. (KNN problem)



The best classifier here is for K = 5? (Reasoning, "Since the training and test are equal we are generalizing perfectly." ?) **NO**
In this case, it may not matter much as the difference in accuracy is only .02, but if you had training 95 & test 90, you wouldn't want to choose training 60 test 60 just because they're equal.

Common misunderstandings - II

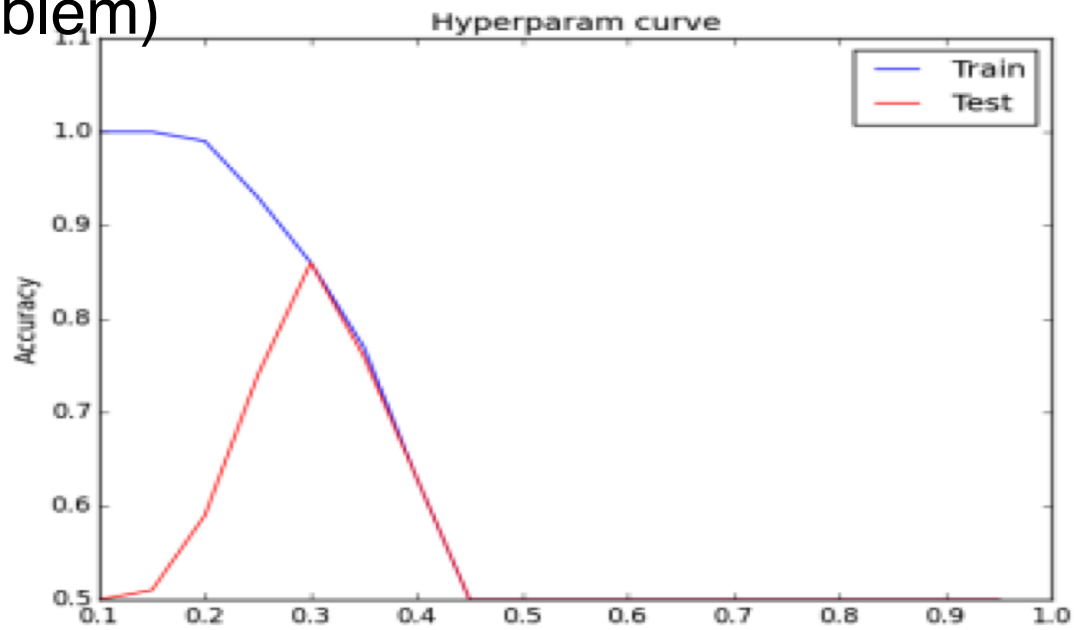
Misunderstanding II: Generally misunderstanding underfitting (NN eps problem)



Misunderstanding II(a): "We are underfitting for eps < .35. Because the test accuracy increases with eps, it is clear we have not learned everything we can learn from our dataset, therefore we are underfitting. " NO

Common misunderstandings - II

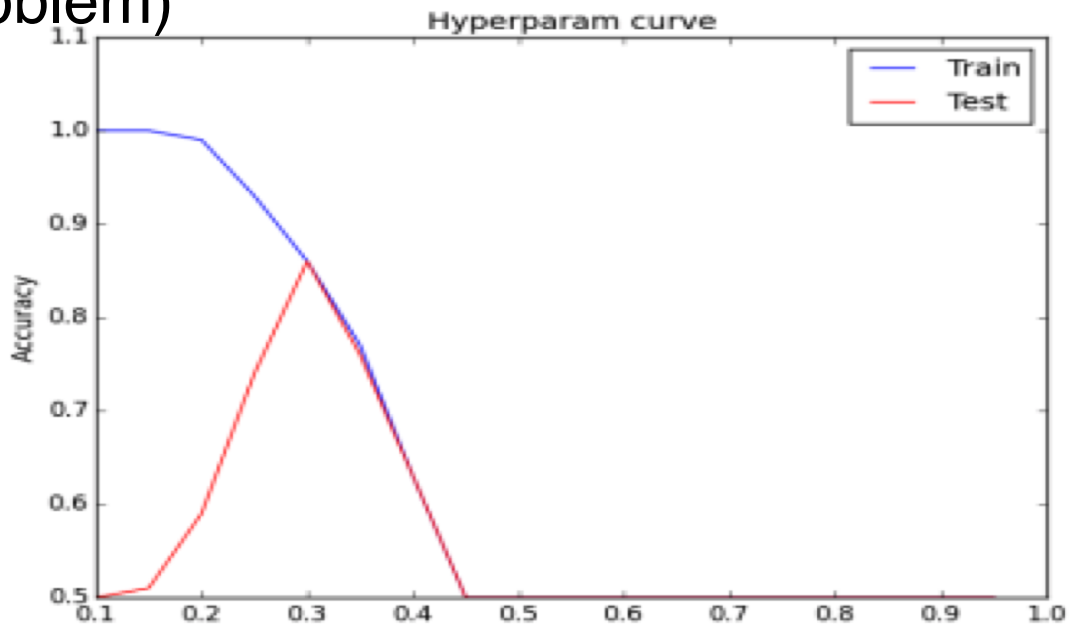
Misunderstanding II: Generally misunderstanding underfitting (NN eps problem)



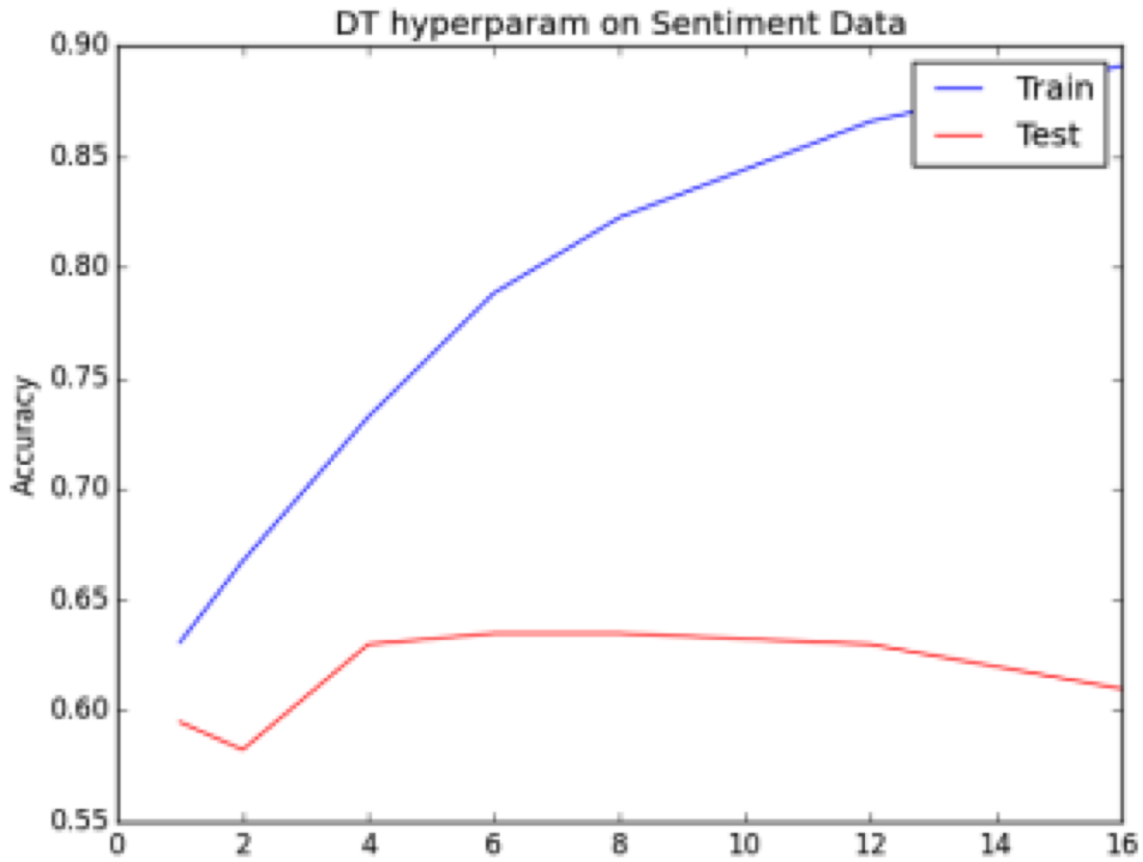
"Test accuracy shows how well you generalize, training accuracy shows how much you've learned." With 100% training and 50% test, we've learned too much, cannot generalize. Therefore, we're overfitting not underfitting.

Common misunderstandings - II

Misunderstanding II: Generally misunderstanding underfitting (NN eps problem)



Misunderstanding II(b): "Although training accuracy is decreasing, test accuracy is increasing, therefore we can not be overfitting".


DT hyperparam on Sentiment Data
Train
Test
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0
2
4
6
8
10
12
14
16

Common misunderstandings - performance on low examples

Case 1 Why does the training accuracy decrease? "there is too much noise in the data" or "it's too hard to learn 1200 examples" ? NO

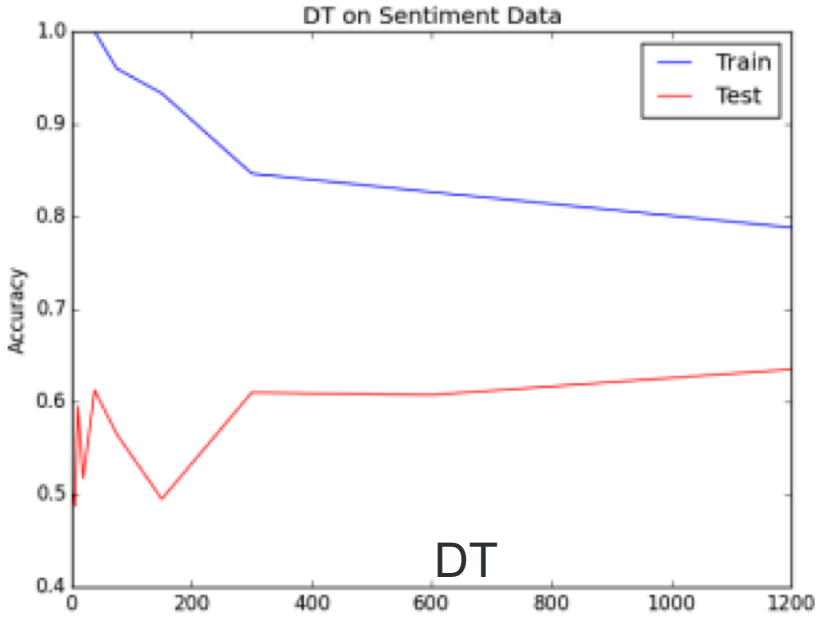
It was because the DT had a fixed max-depth of 9. Also, this example has 100% accuracy early on.

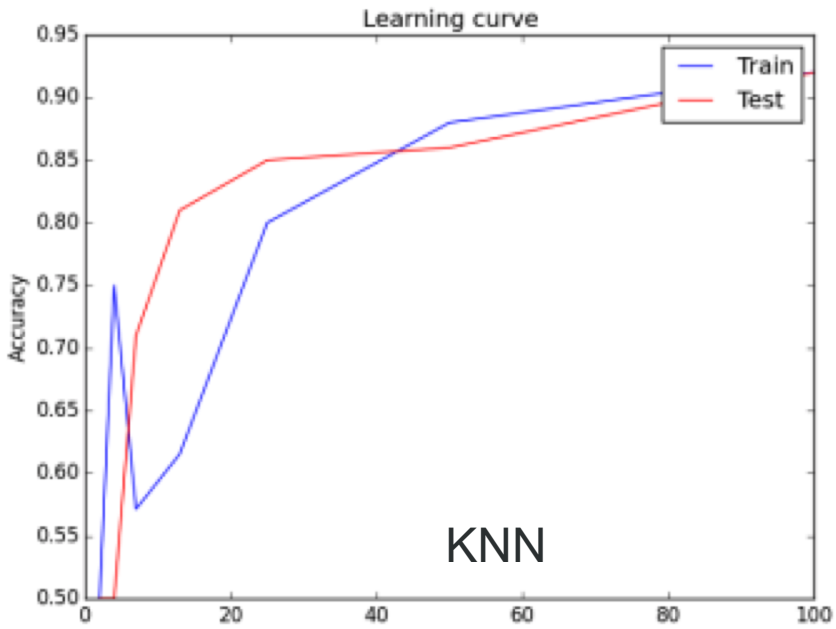
XOR example : DT with a depth of 2, we can get 100% accuracy, but with a depth of 1 we cannot.

By limiting the max-depth we limit what we can learn.

Common misunderstandings - performance on low examples

Case 2 Why is there jaggedness on the left?





The behavior can be very random when you've only seen a few examples.

What we learned last time

- Ranking
- Bias and Fairness
  - Unsupervised adaptation

Supervised adaptation

Goal: learn a classifier f that achieves low expected loss under new distribution $\mathcal{D}^{new}$

Given labeled training data from old distribution

$$\mathcal{D}^{\text{old}} \left\langle \boldsymbol{x}_n^{(\text{old})}, y_n^{(\text{old})} \right\rangle_{n=1}^{N}$$

And labeled examples from new distribution

$$\mathcal{D}^{\text{new}}: \left\langle \boldsymbol{x}_m^{(\text{new})}, y_m^{(\text{new})} \right\rangle_{m=1}^{M}$$

One solution: feature augmentation

Map inputs to a new augmented representation

$$\begin{array}{lcccc} & \text{shared} & \text{old-only} & & \text{new-only} \\ \boldsymbol{x}_n^{(\text{old})} \mapsto \langle & \boldsymbol{x}_n^{(\text{old})} & , \boldsymbol{x}_n^{(\text{old})} & , & \underbrace{0, 0, \ldots, 0}_{D\text{-many}} \rangle \\ \boldsymbol{x}_m^{(\text{new})} \mapsto \langle & \boldsymbol{x}_m^{(\text{new})} & , \underbrace{0, 0, \ldots, 0}_{D\text{-many}} & , & \boldsymbol{x}_m^{(\text{new})} \rangle \end{array}$$

One solution: feature augmentation

- Transform D_{old} and D_{new} training examples
- Train a classifier on new representations
- Done!

One solution: feature augmentation

- Adding instance weighting might be useful if N >> M
- Most effective when distributions are "not too close but not too far"
  - In practice, always try "old only", "new only", "union of old and new" as well!

Bias and how to deal with it

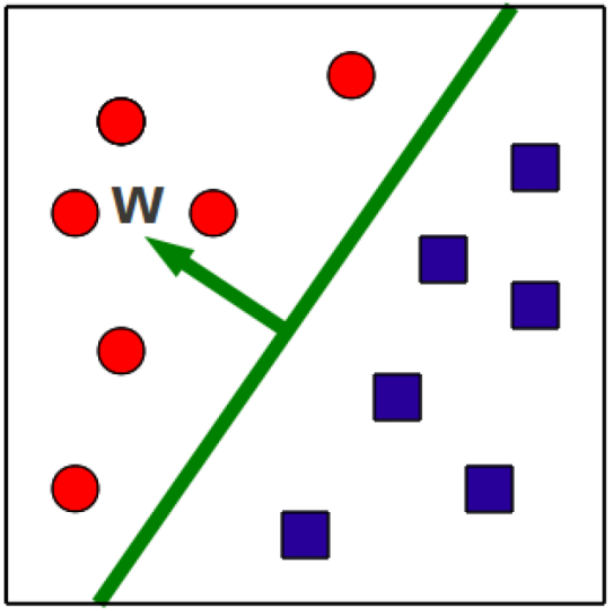
- Train/test mismatch
- Unsupervised adaptation
- Supervised adaptation

Topics

- Linear Models
  - Loss functions
  - Regularization
- Gradient Descent
- Calculus refresher
  - Convexity
  - Gradients

[CIML Chapter 6]

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

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

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

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

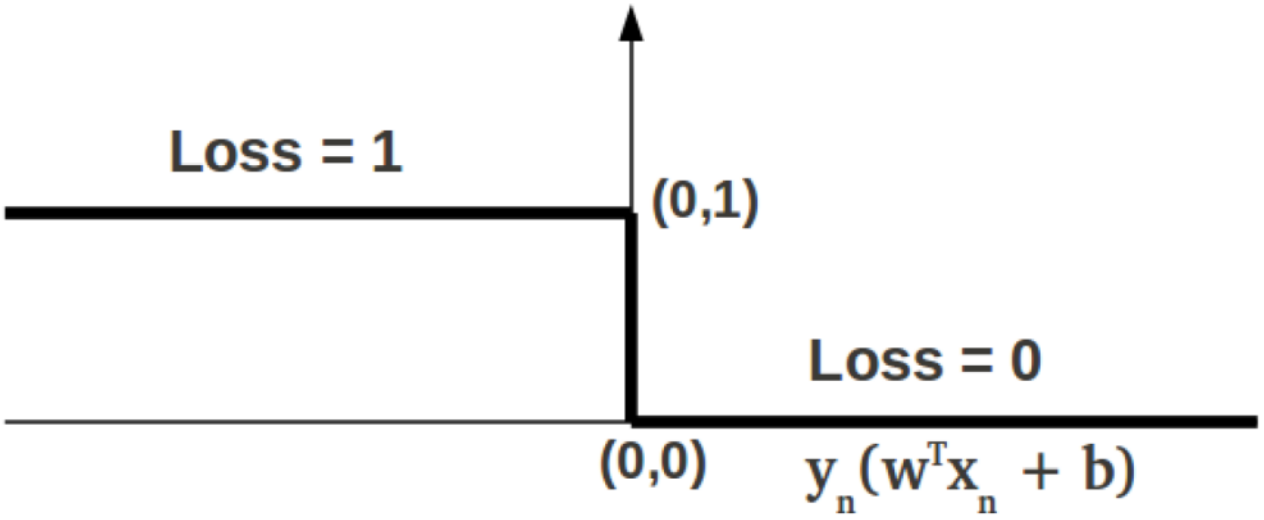
The loss function above is called the 0-1 loss

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

Calculus refresher:

Smooth functions, convex functions

Approximating the 0-1 loss with surrogate loss functions

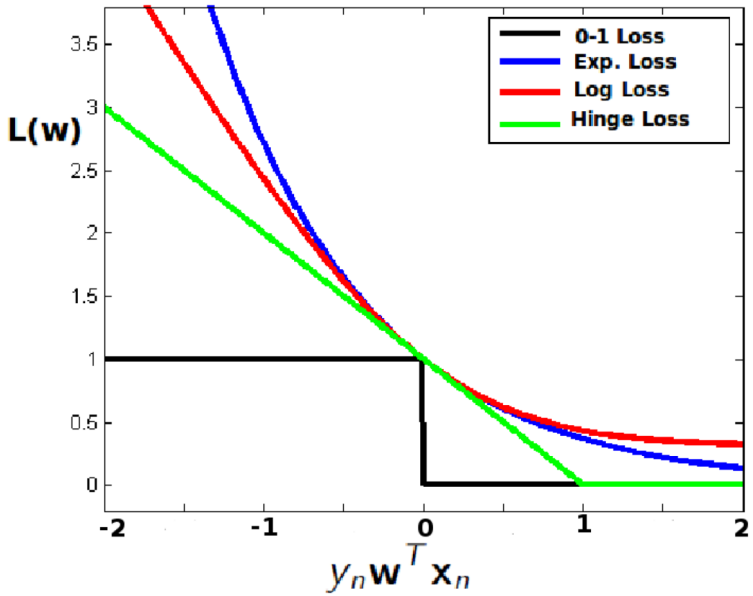
Examples (with b = 0)

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

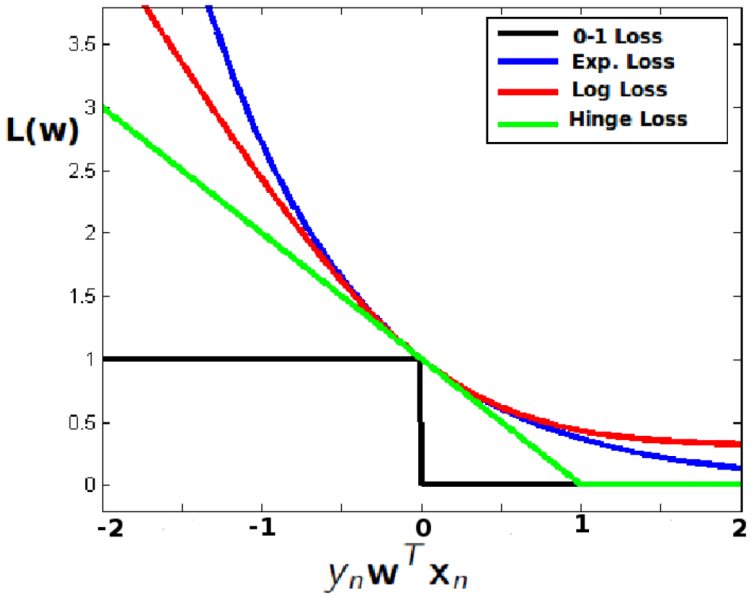
Examples (with b = 0)

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

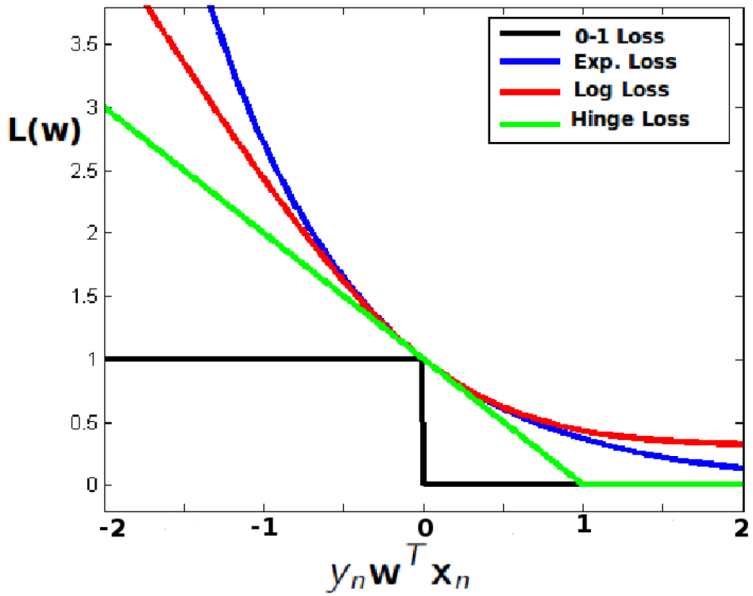
Examples (with b = 0)

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

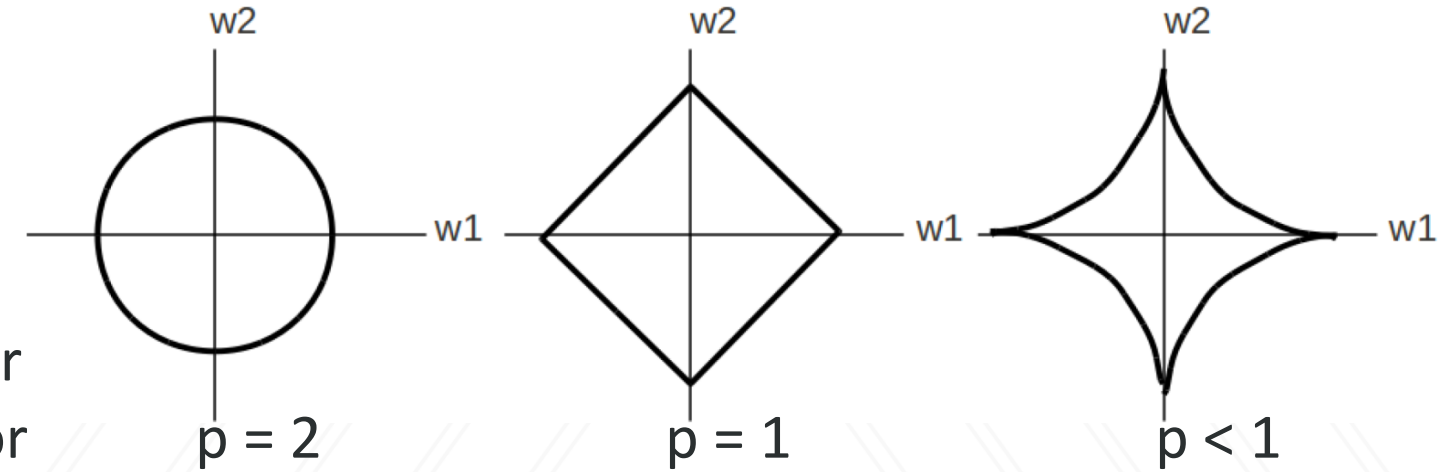
l_p norms can be used as regularizers

$$||\mathbf{w}||_2^2 = \sum_{d=1}^{D} w_d^2$$

$$||\mathbf{w}||_1 = \sum_{d=1}^{D} |w_d|$$

$$||\mathbf{w}||_p = \left(\sum_{d=1}^{D} w_d^p\right)^{1/p}$$

Contour plots for



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(**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

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

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

Calculus refresher:
Gradients

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

Algorithm 22 GradientDescent($\mathcal{F}$, K, η_1, ...)

1: $z^{(0)} \leftarrow \langle 0, 0, \ldots, 0 \rangle$ // initialize variable we are optimizing
2: **for** $k = 1 \ldots K$ **do**
3: $\quad g^{(k)} \leftarrow \nabla_z \mathcal{F}\big|_{z^{(k-1)}}$ // compute gradient at current location
4: $\quad z^{(k)} \leftarrow z^{(k-1)} - \eta^{(k)} g^{(k)}$ // take a step down the gradient
5: **end for**
6: **return** $z^{(K)}$

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)**

UNIVERSITY OF
MARYLAND

Furong Huang
3251 A.V. Williams, College Park, MD 20740
301.405.8010 / furongh@cs.umd.edu