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".
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 $D_{new}$

Given labeled training data from old distribution

$$D_{old} \left\{ (x_n^{(\text{old})}, y_n^{(\text{old})}) \right\}_{n=1}^N$$

And labeled examples from new distribution

$$D_{new} : \left\{ (x_m^{(\text{new})}, y_m^{(\text{new})}) \right\}_{m=1}^M$$
One solution: feature augmentation

Map inputs to a new augmented representation

\[
\begin{align*}
\mathbf{x}_n^{(\text{old})} & \mapsto \left\langle \mathbf{x}_n^{(\text{old})}, \mathbf{x}_n^{(\text{old})}, 0,0,\ldots,0 \right\rangle \\
\mathbf{x}_m^{(\text{new})} & \mapsto \left\langle \mathbf{x}_m^{(\text{new})}, 0,0,\ldots,0, \mathbf{x}_m^{(\text{new})} \right\rangle
\end{align*}
\]
One solution: feature augmentation

- Transform $D_{\text{old}}$ and $D_{\text{new}}$ 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

\[
\hat{y} = \text{sign}(w^T x + b)
\]

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: $\mathbb{E}_{(x,y) \sim \mathcal{D}}[f(x) \neq y]$
Learning a Linear Classifier as an Optimization Problem

Objective function

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

Loss function measures how well classifier fits training data

Indicator function: 1 if (. ) is true, 0 otherwise

The loss function above is called the 0-1 loss

Regularizer prefers solutions that generalize well
Learning a Linear Classifier as an Optimization Problem

\[
\min_{w, b} L(w, b) = \min_{w, b} \sum_{n=1}^{N} \mathbb{I}(y_n(w^T x_n + b) < 0) + \lambda R(w, b)
\]

• **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 \[ [1 - y_n w^T x_n]_+ = \max\{0, 1 - y_n w^T x_n\} \]

Log loss \[ \log[1 + \exp(-y_n w^T x_n)] \]

Exponential loss \[ \exp(-y_n w^T x_n) \]

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: $[1 - y_n w^T x_n]_+ = \max\{0, 1 - y_n w^T x_n\}$

Log loss: $\log[1 + \exp(-y_n w^T x_n)]$

Exponential loss: $\exp(-y_n w^T x_n)$

Q: Which of these loss functions is not smooth?
Approximating the 0-1 loss with surrogate loss functions

Examples (with b = 0)

Hinge loss \[ [1 - y_n w^T x_n]_+ = \max\{0, 1 - y_n w^T x_n\} \]

Log loss \[ \log[1 + \exp(-y_n w^T x_n)] \]

Exponential loss \[ \exp(-y_n w^T x_n) \]

Q: Which of these loss functions is most sensitive to outliers?
Casting Linear Classification as an Optimization Problem

**Objective function**

\[ \min_{w,b} L(w, b) = \min_{w,b} \sum_{n=1}^{N} \mathbb{I}(y_n(w^T x_n + b) < 0) + \lambda R(w, b) \]

**Loss function**

measures how well classifier fits training data

**Regularizer**

prefers solutions that generalize well

\( \mathbb{I}(\cdot) \) Indicator function: 1 if \( \cdot \) 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:

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

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

\[ \|w\|_2^2 = \sum_{d=1}^{D} w_d^2 \]
\[ \|w\|_1 = \sum_{d=1}^{D} |w_d| \]
\[ \|w\|_p = \left( \sum_{d=1}^{D} w_d^p \right)^{1/p} \]

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

\[ \min_{w,b} L(w, b) = \min_{w,b} \sum_{n=1}^{N} \mathbb{I}(y_n(w^T x_n + b) < 0) + \lambda R(w, b) \]

Loss function measures how well classifier fits training data

Regularizer prefers solutions that generalize well

\[ \mathbb{I}(\cdot) \] 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 \textbf{PerceptronTrain}(D, MaxIter)

1. $w_d \leftarrow 0$, for all $d = 1 \ldots D$ \hspace{1cm} // initialize weights
2. $b \leftarrow 0$ \hspace{1cm} // initialize bias
3. \textbf{for iter} = $1 \ldots$ MaxIter \textbf{do}
4. \hspace{0.5cm} \textbf{for all} $(x,y) \in D$ \textbf{do}
5. \hspace{1.5cm} $a \leftarrow \sum_{d=1}^{D} w_d x_d + b$ \hspace{1cm} // compute activation for this example
6. \hspace{1.5cm} \textbf{if} $ya \leq 0$ \textbf{then}
7. \hspace{2cm} $w_d \leftarrow w_d + yx_d$, for all $d = 1 \ldots D$ \hspace{1cm} // update weights
8. \hspace{2cm} $b \leftarrow b + y$ \hspace{1cm} // update bias
9. \hspace{1.5cm} \textbf{end if}
10. \hspace{0.5cm} \textbf{end for}
11. \textbf{end for}
12. \textbf{return} $w_0, w_1, \ldots, w_D, b$

Loss function is a variant of the hinge loss

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

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

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

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

1: $z^{(0)} \leftarrow \langle 0, 0, \ldots, 0 \rangle$  // initialize variable we are optimizing
2: for $k = 1 \ldots K$ do
3:    $g^{(k)} \leftarrow \nabla_{z} \mathcal{F}|_{z^{(k-1)}}$  // compute gradient at current location
4:    $z^{(k)} \leftarrow z^{(k-1)} - \eta^{(k)} g^{(k)}$  // 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)
Furong Huang
3251 A.V. Williams, College Park, MD 20740
301.405.8010 / furongh@cs.umd.edu