Linear Models: (Sub)gradient Descent

CMSC 422
SOHEIL FEIZI
sfeizi@cs.umd.edu

Slides adapted from MARINE CARPUAT
Recap: Linear Models

• General framework for binary classification
• Cast learning as optimization problem
• Optimization objective combines 2 terms
  – Loss function
  – Regularizer
• Does not assume data is linearly separable
• Lets us separate model definition from training algorithm (**Gradient Descent**)
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 \(x\)
Casting Linear Classification as an Optimization Problem

Objective function

Loss function measures how well classifier fits training data

Regularizer prefers solutions that generalize well

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

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

The loss function above is called 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 \[ \exp(-y_n w^T x_n) \]

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

Figure credit: Piyush Rai
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 $p = 2$, $p = 1$, $p < 1$

Figure credit: Piyush Rai
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** \texttt{GradientDescent}(\mathcal{F}, K, \eta_1, \ldots)

1: \( z^{(0)} \leftarrow \langle 0, 0, \ldots, 0 \rangle \)  
   // initialize variable we are optimizing
2: \textbf{for} \( k = 1 \ldots K \) \textbf{do}  
3: \hspace{1em} \( \mathbf{g}^{(k)} \leftarrow \nabla_{\mathbf{z}} \mathcal{F}|_{z^{(k-1)}} \)  
   // compute gradient at current location
4: \hspace{1em} \( z^{(k)} \leftarrow z^{(k-1)} - \eta^{(k)} \mathbf{g}^{(k)} \)  
   // take a step down the gradient
5: \textbf{end for}
6: \textbf{return} \( z^{(K)} \)
Illustrating gradient descent in 1-dimensional case

Figure credit: Piyush Rai
Impact of step size

Image source: https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0
Illustrating gradient descent in 2-dimensional case

Image source: [https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0](https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0)
Illustrating gradient descent in 2-dimensional case

Image source: https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0
Gradient descent algorithm

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

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

• 2 questions
  – When to stop?
    • When the gradient gets close to zero
    • When the objective stops changing much
    • When the parameters stop changing much
    • Early
    • When performance on held-out dev set plateaus
  – How to choose the step size?
    • Start with large steps, then take smaller steps
Now let’s calculate gradients for multivariate objectives

• Consider the following learning objective

\[ \mathcal{L}(w, b) = \sum_n \exp \left[ -y_n (w \cdot x_n + b) \right] + \frac{\lambda}{2} \|w\|^2 \]

• What do we need to do to run gradient descent?
(1) Derivative with respect to $b$

\[
\frac{\partial L}{\partial b} = \frac{\partial}{\partial b} \sum_n \exp \left[ -y_n (w \cdot x_n + b) \right] + \frac{\partial}{\partial b} \frac{\lambda}{2} \|w\|^2 \tag{6.12}
\]

\[
= \sum_n \frac{\partial}{\partial b} \exp \left[ -y_n (w \cdot x_n + b) \right] + 0 \tag{6.13}
\]

\[
= \sum_n \left( \frac{\partial}{\partial b} - y_n (w \cdot x_n + b) \right) \exp \left[ -y_n (w \cdot x_n + b) \right] \tag{6.14}
\]

\[
= - \sum_n y_n \exp \left[ -y_n (w \cdot x_n + b) \right] \tag{6.15}
\]
(2) Gradient with respect to $w$

$$
\nabla_w \mathcal{L} = \nabla_w \sum_n \exp \left[ - y_n (w \cdot x_n + b) \right] + \nabla_w \frac{\lambda}{2} \|w\|^2 
$$

$$
= \sum_n (\nabla_w - y_n (w \cdot x_n + b)) \exp \left[ - y_n (w \cdot x_n + b) \right] + \lambda w
$$

$$
= - \sum_n y_n x_n \exp \left[ - y_n (w \cdot x_n + b) \right] + \lambda w
$$
Subgradients

• Problem: some objective functions are not differentiable everywhere
  – Hinge loss, l1 norm

• Solution: subgradient optimization
  – Let’s ignore the problem, and just try to apply gradient descent anyway!!
  – we will just differentiate by parts...
Example: subgradient of hinge loss

For a given example \( n \)

\[
\partial_w \max \{0, 1 - y_n(w \cdot x_n + b)\} \\
= \partial_w \begin{cases} 
0 & \text{if } y_n(w \cdot x_n + b) > 1 \\
y_n(w \cdot x_n + b) & \text{otherwise}
\end{cases}
\]

\[
= \begin{cases} 
0 & \text{if } y_n(w \cdot x_n + b) > 1 \\
-y_n x_n & \text{otherwise}
\end{cases}
\]
Subgradient Descent for Hinge Loss

Algorithm 23 HingeRegularizedGD(D, λ, MaxIter)

1: \( w \leftarrow \langle 0, 0, \ldots \rangle \), \( b \leftarrow 0 \)  \hspace{1cm} // initialize weights and bias
2: for iter = 1 \ldots MaxIter do  
3: \hspace{0.5cm} g \leftarrow \langle 0, 0, \ldots \rangle \), \( g \leftarrow 0 \) \hspace{1cm} // initialize gradient of weights and bias
4: \hspace{0.5cm} for all \((x,y) \in D\) do
5: \hspace{1.5cm} if \( y(w \cdot x + b) \leq 1 \) then
6: \hspace{2cm} g \leftarrow g + yx \hspace{1cm} // update weight gradient
7: \hspace{2cm} g \leftarrow g + y \hspace{1cm} // update bias derivative
8: \hspace{1.5cm} end if
9: \hspace{0.5cm} end for
10: g \leftarrow g - \lambda w \hspace{1cm} // add in regularization term
11: w \leftarrow w + \eta g \hspace{1cm} // update weights
12: b \leftarrow b + \eta g \hspace{1cm} // update bias
13: end for
14: return \( w, b \)
What is the perceptron optimizing?

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

• Loss function is a variant of the hinge loss

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

• Gradient descent
  – A generic algorithm to minimize objective functions
  – Works well as long as functions are well behaved (i.e. convex)
  – Subgradient descent can be used at points where derivative is not defined
  – Choice of step size is important

• Can be used to find parameters of linear models

• Optional: alternatives to gradient descent