Linear models

The slides are closely adapted from Subhransu Maji’s slides
Overview

- **Linear models**
  - Perceptron: model and learning algorithm combined as one
  - Is there a better way to learn linear models?
- **We will separate models and learning algorithms**
  - Learning as optimization
  - Surrogate loss function
  - Regularization
  - Gradient descent
  - Batch and online gradients
  - Support vector machines

\[
\begin{align*}
\{ \text{model design} \} &= \{ \text{optimization} \} \\
\{ \text{optimization} \} &= \{ \text{optimization} \}
\end{align*}
\]
Learning as optimization

\[
\min_w \sum_n 1[y_n w^T x_n < 0]
\]

fewest mistakes

- The **perceptron algorithm** will find an optimal \( w \) if the data is **separable**
  - efficiency depends on the **margin** and **norm** of the data
- However, if the data is not separable, optimizing this is **NP-hard**
  - i.e., there is no efficient way to minimize this unless \( P=NP \)
In addition to minimizing training error, we want a simpler model.

- Remember our goal is to minimize generalization error.

We can add a regularization term $R(w)$ that prefers simpler models.

- For example, we may prefer decision trees of shallow depth.

Here $\lambda$ is a hyperparameter of optimization problem.

$$\min_w \sum_n 1[y_n w^T x_n < 0] + \lambda R(w)$$

fewest mistakes  
simpler model
Learning as optimization

\[ \min_w \sum_n 1[y_n w^T x_n < 0] + \lambda R(w) \]

- The questions that remain are:
  - What are good ways to adjust the optimization problem so that there are efficient algorithms for solving it?
  - What are good regularizations \( R(w) \) for hyperplanes?
  - Assuming that the optimization problem can be adjusted appropriately, what algorithms exist for solving the regularized optimization problem?
**Convex surrogate loss functions**

- **Zero/one loss** is hard to optimize
  - Small changes in \( \mathbf{w} \) can cause large changes in the loss
- **Surrogate loss**: replace **Zero/one loss** by a smooth function
  - Easier to optimize if the surrogate loss is convex
- **Examples:**

\[
y = +1 \quad \hat{y} \leftarrow \mathbf{w}^T \mathbf{x}
\]

- Zero/one: \( \ell^{(0/1)}(y, \hat{y}) = 1[y\hat{y} \leq 0] \)
- Hinge: \( \ell^{(hin)}(y, \hat{y}) = \max \{0, 1 - y\hat{y}\} \)
- Logistic: \( \ell^{(log)}(y, \hat{y}) = \frac{1}{\log 2} \log (1 + \exp[-y\hat{y}]) \)
- Exponential: \( \ell^{(exp)}(y, \hat{y}) = \exp[-y\hat{y}] \)
- Squared: \( \ell^{(sqr)}(y, \hat{y}) = (y - \hat{y})^2 \)
What are good regularization functions $R(w)$ for hyperplanes?

We would like the weights —

- To be small —
  - Change in the features cause small change to the score
  - Robustness to noise

- To be sparse —
  - Use as few features as possible
  - Similar to controlling the depth of a decision tree

This is a form of inductive bias
Weight regularization

- Just like the surrogate loss function, we would like $R(w)$ to be convex
- Small weights regularization

$$R^{(\text{norm})}(w) = \sqrt{\sum_{d} w_d^2}$$

$$R^{(\text{sqr})}(w) = \sum_{d} w_d^2$$

- Sparsity regularization

$$R^{(\text{count})}(w) = \sum_{d} 1[|w_d| > 0]$$

not convex

- Family of “p-norm” regularization

$$R^{(\text{p-norm})}(w) = \left( \sum_{d} |w_d|^p \right)^{1/p}$$
Contours of p-norms

$$\|x\|_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{\frac{1}{p}}$$

convex for $p \geq 1$

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}$$

$$\|x\|_{\infty} = \max_{i=1,\ldots,n} |x_i|$$

http://en.wikipedia.org/wiki/Lp_space
Contours of p-norms

\[ \|x\|_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{\frac{1}{p}} \]

not convex for \(0 \leq p < 1\)

\[ p = \frac{2}{3} \]

Counting non-zeros:

\[ p = 0 \]

\[ R^{(\text{count})}(w) = \sum_d 1[|w_d| > 0] \]

http://en.wikipedia.org/wiki/Lp_space
General optimization framework

Select a suitable:
- convex surrogate loss
- convex regularization

Select the hyperparameter $\lambda$

Minimize the regularized objective with respect to $\mathbf{w}$

This framework for optimization is called Tikhonov regularization or generally Structural Risk Minimization (SRM)

http://en.wikipedia.org/wiki/Tikhonov_regularization
Optimization by gradient descent

Convex function

- Step size
- Local optima = global optima

Non-convex function

- Local optima
- Global optima

\[ g^{(k)} \leftarrow \nabla_p F(p)|_{p_k} \]

compute gradient at the current location

\[ p_{k+1} \leftarrow p_k - \eta_k g^{(k)} \]

take a step down the gradient
Choice of step size

- The step size is important —
  - too small: slow convergence
  - too large: no convergence

- A strategy is to use large step sizes initially and small step sizes later:
  \[
  \eta_t \leftarrow \eta_0 / (t_0 + t)
  \]

- There are methods that converge faster by adapting step size to the curvature of the function
  - Field of convex optimization

http://stanford.edu/~boyd/cvxbook/
Convex functions

$D$ – a domain in $\mathbb{R}^n$.

A **convex function** $f : D \to \mathbb{R}$ is one that satisfies, for any $x_0$ and $x_1$ in $D$:

$$f((1 - \alpha)x_0 + \alpha x_1) \leq (1 - \alpha)f(x_0) + \alpha f(x_1).$$

Line joining $(x_0, f(x_0))$ and $(x_1, f(x_1))$ lies above the function graph.
Example: Exponential loss

\[ \mathcal{L}(\mathbf{w}) = \sum_n \exp(-y_n \mathbf{w}^T \mathbf{x}_n) + \frac{\lambda}{2} \|\mathbf{w}\|^2 \quad \text{objective} \]

\[ \frac{d\mathcal{L}}{d\mathbf{w}} = \sum_n -y_n \mathbf{x}_n \exp(-y_n \mathbf{w}^T \mathbf{x}_n) + \lambda \mathbf{w} \quad \text{gradient} \]

\[ \mathbf{w} \leftarrow \mathbf{w} - \eta \left( \sum_n -y_n \mathbf{x}_n \exp(-y_n \mathbf{w}^T \mathbf{x}_n) + \lambda \mathbf{w} \right) \quad \text{update} \]

**loss term**

\[ \mathbf{w} \leftarrow \mathbf{w} + cy_n \mathbf{x}_n \]

high for misclassified points

similar to the perceptron update rule!

**regularization term**

\[ \mathbf{w} \leftarrow (1 - \eta \lambda) \mathbf{w} \]

shrinks weights towards zero
Batch and online gradients

\[ \mathcal{L}(w) = \sum_n \mathcal{L}_n(w) \quad \text{objective} \]

\[ w \leftarrow w - \eta \frac{d\mathcal{L}}{dw} \quad \text{gradient descent} \]

**batch gradient**

\[ w \leftarrow w - \eta \left( \sum_n \frac{d\mathcal{L}_n}{dw} \right) \]

- sum of n gradients
- update weight after you see all points

**online gradient**

\[ w \leftarrow w - \eta \left( \frac{d\mathcal{L}_n}{dw} \right) \]

- gradient at n\(^{th}\) point
- update weights after you see each point

Online gradients are the default method for multi-layer perceptrons
The hinge loss is not differentiable at $z=1$

- Subgradient is any direction that is below the function
- For the hinge loss a possible subgradient is:

$$
\ell^{(hinge)}(y, w^T x) = \max(0, 1 - yw^T x)
$$

$$
\frac{d\ell^{hinge}}{dw} = \begin{cases} 
0 & \text{if } yw^T x > 1 \\
-yx & \text{otherwise}
\end{cases}
$$
Example: Hinge loss

\[ \mathcal{L}(\mathbf{w}) = \sum_n \max(0, 1 - y_n \mathbf{w}^T \mathbf{x}_n) + \frac{\lambda}{2} ||\mathbf{w}||^2 \]  

**objective**

\[ \frac{d\mathcal{L}}{d\mathbf{w}} = \sum_n -1[y_n \mathbf{w}^T \mathbf{x}_n \leq 1]y_n \mathbf{x}_n + \lambda \mathbf{w} \]  

**subgradient**

\[ \mathbf{w} \leftarrow \mathbf{w} - \eta \left( \sum_n -1[y_n \mathbf{w}^T \mathbf{x}_n \leq 1]y_n \mathbf{x}_n + \lambda \mathbf{w} \right) \]  

**update**

- **loss term**
  \[ \mathbf{w} \leftarrow \mathbf{w} + \eta y_n \mathbf{x}_n \]
  only for points \( y_n \mathbf{w}^T \mathbf{x}_n \leq 1 \)

- **perceptron update** \( y_n \mathbf{w}^T \mathbf{x}_n \leq 0 \)

- **regularization term**
  \[ \mathbf{w} \leftarrow (1 - \eta \lambda) \mathbf{w} \]
  shrinks weights towards zero
**Example: Squared loss**

\[
\mathcal{L}(w) = \sum_{n} (y_n - w^T x_n)^2 + \frac{\lambda}{2} ||w||^2
\]

**objective**

**matrix notation**

\[
\begin{bmatrix}
  x_{1,1} & x_{1,2} & \cdots & x_{1,D} \\
  x_{2,1} & x_{2,2} & \cdots & x_{2,D} \\
  \vdots & \vdots & \ddots & \vdots \\
  x_{N,1} & x_{N,2} & \cdots & x_{N,D}
\end{bmatrix}
\begin{bmatrix}
  w_1 \\
  w_2 \\
  \vdots \\
  w_D
\end{bmatrix}
= 
\begin{bmatrix}
  \sum_d x_{1,d} \omega_d \\
  \sum_d x_{2,d} \omega_d \\
  \vdots \\
  \sum_d x_{N,d} \omega_d
\end{bmatrix}
\begin{bmatrix}
  y_1 \\
  y_2 \\
  \vdots \\
  y_N
\end{bmatrix}
\]

**equivalent loss**

\[
\min_w \mathcal{L}(w) = \frac{1}{2} ||Xw - Y||^2 + \frac{\lambda}{2} ||w||^2
\]
Example: Squared loss

\[
\min_w \mathcal{L}(w) = \frac{1}{2} \|Xw - Y\|^2 + \frac{\lambda}{2} \|w\|^2 \quad \text{objective}
\]

\[
\nabla_w \mathcal{L}(w) = X^T (Xw - Y) + \lambda w
\]

\[
= X^T Xw - X^T Y + \lambda w
\]

\[
= \left( X^T X + \lambda I \right) w - X^T Y
\]

At optima the gradient=0

\[
\left( X^T X + \lambda I \right) w - X^T Y = 0
\]

\[\iff\]

\[
\left( X^T X + \lambda I_D \right) w = X^T Y
\]

\[\iff\]

\[
w = \left( X^T X + \lambda I_D \right)^{-1} X^T Y
\]

exact closed-form solution
Matrix inversion vs. gradient descent

- Assume, we have D features and N points
- Overall time via matrix inversion
  - The closed form solution involves computing:
    \[ \mathbf{w} = \left( \mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_D \right)^{-1} \mathbf{X}^\top \mathbf{Y} \]
  - Total time is \( O(D^2N + D^3 + DN) \), assuming \( O(D^3) \) matrix inversion
  - If \( N > D \), then total time is \( O(D^2N) \)
- Overall time via gradient descent
  - Gradient:
    \[ \frac{d\mathcal{L}}{d\mathbf{w}} = \sum_n \left[ -2(y_n - \mathbf{w}^\top \mathbf{x}_n) \mathbf{x}_n + \lambda \mathbf{w} \right] \]
  - Each iteration: \( O(ND) \); \( T \) iterations: \( O(TND) \)
- Which one is faster?
  - Small problems \( D < 100 \): probably faster to run matrix inversion
  - Large problems \( D > 10,000 \): probably faster to run gradient descent
Which **hyperplane** is the best?
Support Vector Machines (SVMs)

- Maximize the distance to the nearest point (margin), while correctly classifying all the points
Optimization for SVMs

Separable case: hard margin SVM

\[
\min_{\mathbf{w}} \frac{1}{\delta(\mathbf{w})}
\]

maximize margin

subject to: \( y_n \mathbf{w}^T \mathbf{x}_n \geq 1, \forall n \) separate by a non-trivial margin
Support Vector Machines (SVMs)

- Maximize the distance to the nearest point (margin), while correctly classifying all the points.
Optimization for SVMs

Separable case: hard margin SVM

\[
\min_{\mathbf{w}} \frac{1}{\delta(\mathbf{w})}
\]

maximize margin

subject to: \( y_n \mathbf{w}^T \mathbf{x}_n \geq 1, \forall n \)

separate by a non-trivial margin

Non-separable case: soft margin SVM

\[
\min_{\mathbf{w}} \frac{1}{\delta(\mathbf{w})} + C \sum_n \xi_n
\]

maximize margin minimize slack

subject to: \( y_n \mathbf{w}^T \mathbf{x}_n \geq 1 - \xi_n, \forall n \)

allow some slack

\( \xi_n \geq 0 \)
Margin of a classifier

\[ w^T x - 1 = 0 \]

\[ w^T x + 1 = 0 \]

\[ \delta(w) = \frac{1}{||w||} \]

\[ \min_w \frac{1}{\delta(w)} \equiv \min_w ||w|| \]

 maximizing margin = minimizing norm
Equivalent optimization for SVMs

Separable case: hard margin SVM

\[
\min_w \frac{1}{2} \|w\|^2 \quad \text{maximize margin}
\]

subject to: \( y_n w^T x_n \geq 1, \forall n \)

Non-separable case: soft margin SVM

\[
\min_w \frac{1}{2} \|w\|^2 + C \sum_n \xi_n \quad \text{maximize margin minimize slack}
\]

subject to: \( y_n w^T x_n \geq 1 - \xi_n, \forall n \) \quad \text{allow some slack}

\( \xi_n \geq 0 \)
Suppose I tell you what $w$ is, but forgot to give you the slack variables.

Can you derive the optimal slack for the $n^{th}$ example?

- $y_n w^T x_n = 0.8$, $\xi_n = ?$ 0.2
- $y_n w^T x_n = -1$, $\xi_n = ?$ 2.0
- $y_n w^T x_n = 2.5$, $\xi_n = ?$ 0

$$\xi_n = \begin{cases} 
0 & y_n w^T x_n \geq 1 \\
1 - y_n w^T x_n & \text{otherwise}
\end{cases}$$

Same as hinge loss with squared norm regularization!
Support vectors

Which points contribute to the loss?

$\mathbf{w}^T \mathbf{x} - 1 = 0$

$\mathbf{w}^T \mathbf{x} + 1 = 0$

margin $\delta(\mathbf{w})$
Support vectors

Which points contribute to the loss?

\[ \mathbf{w}^T \mathbf{x} - 1 = 0 \]

\[ \mathbf{w}^T \mathbf{x} + 1 = 0 \]
Under suitable conditions*, provided you pick the step sizes appropriately, the convergence rate of gradient descent is $O(1/N)$

- i.e., if you want a solution within 0.001 of the optimal you have to run the gradient descent for $N=1000$ iterations.

For linear models (hinge/logistic/exponential loss) and squared-norm regularization there are off-the-shelf solvers that are fast in practice: SVM$^\text{perf}$, LIBLINEAR, PEGASOS

- SVM$^\text{perf}$, LIBLINEAR use a different optimization method

* the function is strongly convex: $f(y) \geq f(x) + \nabla f(x)^T (y-x) + \frac{m}{2} \|y-x\|^2_2$
Objective: detect (localize) standing humans in an image
- cf face detection with a sliding window classifier

- reduces object detection to binary classification
  - does an image window contain a person or not?

Method: the HOG detector
Training data and features

• Positive data – 1208 positive window examples

• Negative data – 1218 negative window examples (initially)
Feature: histogram of oriented gradients (HOG)

- tile window into 8 x 8 pixel cells
- each cell represented by HOG

Feature vector dimension = 16 x 8 (for tiling) x 8 (orientations) = 1024

Slide from Andrew Zisserman
Examples

Slide from Andrew Zisserman
Averaged features

Averaged positive examples

Slide from Andrew Zisserman
Algorithm

Training (Learning)

- Represent each example window by a HOG feature vector

\[ x_i \in \mathbb{R}^d, \text{ with } d = 1024 \]

- Train a SVM classifier

Testing (Detection)

- Sliding window classifier

\[ f(x) = \mathbf{w}^\top \mathbf{x} + b \]
Example output

Dalal and Triggs, CVPR 2005

Slide from Andrew Zisserman
Model

\[ f(x) = w^\top x + b \]
What do negative weights mean?

\[ w_x > 0 \]
\[ (w_+ - w_-)x > 0 \]
\[ w_+ > w_- \]

Pedestrian model > Background model

Complete system should compete pedestrian/pillar/doorway models

Discriminative models come equipped with own bg

(avoid firing on doorways by penalizing vertical edges)
Linear regression

- We want
  \[ y = w^T x \]
  \[ \hat{w} = \arg\min_w \sum (w^T x_i - y_i)^2 \]

- There are two solutions:
  - 1. Gradient decent
  - 2. Luckily, setting gradient to zero leads to a closed form solution
    - Write it in matrix form:
      \[ \hat{w} = \arg\min_w (Xw - Y)^2 \]
    - Set derivative to zero:
      \[ d \frac{d}{dw} (Xw - Y)^2 = 0 \]
    - Some linear algebra:
      \[ X^T Xw - X^T Y = 0 \]
    - Close form solution
      \[ \hat{w} = (X^T X)^{-1} X^T Y \]
Figures of various “p-norms” are from Wikipedia
  ‣ http://en.wikipedia.org/wiki/Lp_space

Slides are closely following and adapted from Hal Daume’s book and Subranshu Maji’s course.

Some slides are adopted from Andrew Zisserman’s course:
  ‣ http://www.robots.ox.ac.uk/~az/lectures/ml/lect2.pdf