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
  - Subgradient descent
  - Support vector machines

Learning as optimization

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

- 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
  - Recall the bias and variance tradeoff for learners
- 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
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?

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

- **Zero/one loss** is hard to optimize
  - Small changes in $w$ 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:**

- **Zero/one** $\ell_{\text{zero}}(y, \hat{y}) = 1[y \neq \hat{y}]$
- **Hinge** $\ell_{\text{hinge}}(y, \hat{y}) = \max(0, 1 - y\hat{y})$
- **Logistic** $\ell_{\text{logistic}}(y, \hat{y}) = \frac{1}{\log 2} \log(1 + \exp[-y\hat{y}])$
- **Exponential** $\ell_{\text{exp}}(y, \hat{y}) = \exp[-y\hat{y}]$
- **Squared** $\ell_{\text{squared}}(y, \hat{y}) = (y - \hat{y})^2$

**Weight regularization**

- 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

**Convex surrogate loss functions**

- **Zero/one loss** is hard to optimize
  - Small changes in $w$ 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**:

- **Zero/one** $\ell_{\text{zero}}(y, \hat{y}) = 1[y \neq \hat{y}]$
- **Hinge** $\ell_{\text{hinge}}(y, \hat{y}) = \max(0, 1 - y\hat{y})$
- **Logistic** $\ell_{\text{logistic}}(y, \hat{y}) = \frac{1}{\log 2} \log(1 + \exp[-y\hat{y}])$
- **Exponential** $\ell_{\text{exp}}(y, \hat{y}) = \exp[-y\hat{y}]$
- **Squared** $\ell_{\text{squared}}(y, \hat{y}) = (y - \hat{y})^2$
### 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:

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

http://en.wikipedia.org/wiki/Lp_space

### General optimization framework

\[
\min_{\mathbf{w}} \sum_{n} \ell \left( y_n, \mathbf{w}^T \mathbf{x}_n \right) + \lambda R(\mathbf{w})
\]

- 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**

- Compute gradient at the current location \( g^{(k)} \)
- Take a step down the gradient \( p_{k+1} = p_k - \eta_k g^{(k)} \)

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

**Non-convex function**

Local optima = Global optima
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

Example: Exponential loss

\[ L(w) = \sum_n \exp(-y_n w^T x_n) + \frac{\lambda}{2} ||w||^2 \]  
objective

\[ \frac{dL}{dw} = \sum_n -y_n x_n \exp(-y_n w^T x_n) + \lambda w \]  
gradient

\[ w \leftarrow w - \eta \left( \sum_n -y_n x_n \exp(-y_n w^T x_n) + \lambda w \right) \]  
update

Batch and online gradients

\[ L(w) = \sum_n L_n(w) \]  
objective

\[ w \leftarrow w - \frac{dL}{dw} \]  
gradient descent

- batch gradient
  \[ w \leftarrow w - \eta \left( \sum_n \frac{dL_n}{dw} \right) \]  
  sum of n gradients
  update weight after you see all points

- online gradient
  \[ w \leftarrow w - \eta \left( \frac{dL_n}{dw} \right) \]  
  gradient at nth point
  update weights after you see each point

Online gradients are the default method for multi-layer perceptrons

Subgradient

\[ f^{(\text{hinge})}(y, w^T x) = \max(0, 1 - yw^T x) \]  
subgradient

- 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:

\[ \frac{df^{(\text{hinge})}}{dw} = \begin{cases} 
0 & \text{if } yw^T x > 1 \\
-yx & \text{otherwise} 
\end{cases} \]
Example: Hinge loss

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

\[ \frac{d\mathcal{L}}{dw} = \sum_n -1[y_n w^T x_n \leq 1] y_n x_n + \lambda w \] subgradient

\[ w \leftarrow w - \eta \left( \sum_n -1[y_n w^T x_n \leq 1] y_n x_n + \lambda w \right) \] update

- loss term
  \[ w \leftarrow w + \eta y_n x_n \]
  - only for points \( y_n w^T x_n \leq 1 \)
- regularization term
  \[ w \leftarrow (1 - \eta \lambda) w \]
  - shrinks weights towards zero
- perceptron update \( y_n w^T x_n \leq 0 \)

Example: Squared loss

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

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

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

Matrix inversion vs. gradient descent

- Assume, we have D features and N points
- Overall time via matrix inversion
  - The closed form solution involves computing:
    \[ w = \left( X^T X + \lambda I_D \right)^{-1} X^T Y \]
  - 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:
    \[ \frac{d\mathcal{L}}{dw} = \sum_n -2(y_n - w^T x_n)x_n + \lambda w \]
  - 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
Picking a good hyperplane

- 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

\[
\min_w \frac{1}{\delta(w)} \quad \text{maximize margin}
\]

subject to: \( y_n w^T x_n \geq 1, \forall n \) separate by a non-trivial margin

Non-separable case: soft margin SVM

\[
\min_w \frac{1}{\delta(w)} + C \sum_n \xi_n \quad \text{maximize margin minimize slack}
\]

subject to: \( y_n w^T x_n \geq 1 - \xi_n, \forall n \) allow some slack
\[
\xi_n \geq 0
\]

Margin of a classifier

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

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

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

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

separate by a non-trivial margin

**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: \( y_n w^T x_n \geq 1 - \xi_n, \forall n \) \( \xi_n \geq 0 \)

allow some slack

Slack variables

$$\min_w \frac{1}{2}||w||^2 + C \sum_n \xi_n$$

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

\( \xi_n \geq 0 \)

- 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?

\[
\begin{array}{ll}
\text{if } y_n w^T x_n = 0.8 & \text{then } \xi_n = 0.2 \\
\text{if } y_n w^T x_n = -1 & \text{then } \xi_n = 2.0 \\
\text{if } y_n w^T x_n = 2.5 & \text{then } \xi_n = 0 \\
\end{array}
\]

Same as hinge loss with squared norm regularization!

Optimization for linear models

- 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 0.0001 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:
  - SVMperf, LIBLINEAR, PEGASOS
  - SVMperf, LIBLINEAR use a different optimization method

Slides credit

- Figures of various “p-norms” are from Wikipedia
  - http://en.wikipedia.org/wiki/Lp_space
- Some of the slides are based on CIML book by Hal Daume III

* the function is strongly convex: \( f(y) \geq f(x) + \nabla f(x)^T (y-x) + \frac{m}{2}||y-x||_2^2 \)
% Code to plot various loss functions
y1=1;
y2=linspace(-2,3,500);
zeroOneLoss = y1*y2 <=0;
hingeLoss = max(0, 1-y1*y2);
logisticLoss = log(1+exp(-y1*y2))/log(2);
expLoss = exp(-y1*y2);
squaredLoss = (y1-y2).^2;

% Plot them
figure(1); clf; hold on;
plot(y2, zeroOneLoss, 'k-', 'LineWidth',1);
plot(y2, hingeLoss, 'b-', 'LineWidth',1);
plot(y2, logisticLoss, 'r-', 'LineWidth',1);
plot(y2, expLoss, 'g-', 'LineWidth',1);
plot(y2, squaredLoss, 'm-', 'LineWidth',1);
ylabel('Prediction', 'FontSize',16);
xlabel('Loss', 'FontSize',16);
legend({'Zero/one', 'Hinge', 'Logistic', 'Exponential', 'Squared'}, 'Location','NorthEast', 'FontSize',16);
box on;