Logistic Regression

- Learn $P(Y|X)$ directly
  - Assume a particular functional form for link function
  - Sigmoid applied to a linear function of the input features:
  $$P(Y = 0|X, W) = \frac{1}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

- $P(Y = 1|X, W) = 1 - P(Y = 0|X, W) = \frac{\exp(w_0 + \sum_i w_i X_i)}{1 + \exp(w_0 + \sum_i w_i X_i) \quad \in \mathbb{R}}$

- Features can be discrete or continuous!
Optimizing concave function – Gradient ascent

- Conditional likelihood for Logistic Regression is concave. Find optimum with gradient ascent

Gradient:

\[
\nabla_w l(w) = \left[ \frac{\partial l(w)}{\partial w_0}, \ldots, \frac{\partial l(w)}{\partial w_n} \right]'
\]

Update rule:

\[
\Delta w = \eta \nabla_w l(w)
\]

\[
w_i^{(t+1)} = w_i^{(t)} + \eta \frac{\partial l(w)}{\partial w_i}
\]

- Gradient ascent is simplest of optimization approaches
  - e.g., Conjugate gradient ascent can be much better

Gradient Ascent for LR

Gradient ascent algorithm: iterate until change < \varepsilon

\[
w_0^{(t+1)} = w_0^{(t)} + \eta \sum_{j=1}^{N} [y_j - \hat{P}(Y \hat{\Phi} = 1 | x_j, w_0^{(t)})] x_j
\]

For i=1,…,k,

\[
w_i^{(t+1)} = w_i^{(t)} + \eta \sum_{j=1}^{N} x_i^j [y_j - \hat{P}(Y \hat{\Phi} = 1 | x_j, w_i^{(t)})]
\]

repeat
The Cost, The Cost!!! Think about the cost…

What’s the cost of a gradient update step for LR???

\[ w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \left( -\lambda w_i^{(t)} + \sum_{j=1}^{N} x_i^j [y_j - \hat{p}(Y_i^j = 1 | x_i^j, w)] \right) \]

Learning Problems as Expectations

- Minimizing loss in training data:
  - Given dataset: \( x^1, x^2, \ldots, x^N \)
  - Sampled iid from some distribution \( p(x) \) on features:
  - Loss function, e.g., hinge loss, logistic loss,…
  - We often minimize loss in training data:
    \[ \ell_D(w) = \frac{1}{N} \sum_{j=1}^{N} \ell(w, x^j) \]

However, we should really minimize expected loss on all data:

\[ \ell(w) = E_x [\ell(w, x)] = \int p(x) \ell(w, x) dx \]

So, we are approximating the integral by the average on the training data.
Gradient descent in Terms of Expectations

- “True” objective function:
  \[
  \ell(w) = E_x [\ell(w, x)] = \int p(x) \ell(w, x) dx
  \]

- Taking the gradient:
  \[
  \nabla_w \ell(w) = \nabla_w \left( E_x [\ell(w, x)] \right) = E_x [\nabla_w \ell(w, x)]
  \]

- “True” gradient descent rule:
  \[
  w^{(t+1)} = w^{(t)} - \eta E_x [\nabla_w \ell(w, x)]
  \]

- How do we estimate expected gradient?

SGD: Stochastic Gradient Ascent (or Descent)

- “True” gradient:
  \[
  \nabla \ell(w) = E_x [\nabla \ell(w, x)]
  \]

- Sample based approximation:
  \[
  \nabla \ell(w) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla \ell(w, x_i)
  \]

- What if we estimate gradient with just one sample???
  - Unbiased estimate of gradient
    \[
    E_x [\nabla \ell(w, x_i)] = \nabla \ell(w)
    \]
  - Very noisy!
  - Called stochastic gradient ascent (or descent)
    - Among many other names
  - VERY useful in practice!!!
Stochastic Gradient Ascent for Logistic Regression

- Logistic loss as a stochastic function:
  \[ E_\mathbf{x} [\ell(\mathbf{w}, \mathbf{x})] = E_\mathbf{x} \left[ \ln P(y | \mathbf{x}, \mathbf{w}) - \lambda \| \mathbf{w} \|_2^2 \right] \]

- Batch gradient ascent updates:
  \[ w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \left\{ -\lambda w_i^{(t)} + \frac{1}{N} \sum_{j=1}^{N} x_i^{(j)} y^{(j)} - P(Y = 1 | \mathbf{x}^{(j)}, \mathbf{w}^{(t)}) \right\} \]

- Stochastic gradient ascent updates:
  - Online setting:
    \[ w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta_t \left\{ -\lambda w_i^{(t)} + x_i^{(t)} y^{(t)} - P(Y = 1 | \mathbf{x}^{(t)}, \mathbf{w}^{(t)}) \right\} \]

Stochastic Gradient Descent: general case

- Given a stochastic function of parameters:
  \[ f(\mathbf{w}) = E_\mathbf{x} [f(\mathbf{w}, \mathbf{x})] \]
  \[ \mathbf{w}^{\ast} = \arg \min_{\mathbf{w}} f(\mathbf{w}) = \arg \min_{\mathbf{w}} E_\mathbf{x} [f(\mathbf{w}, \mathbf{x})] \]

- Start from \( \mathbf{w}^{(0)} \)
  \( \mathbf{w}^{(0)} = 0 \)

- Repeat until convergence:
  - Get a sample data point \( \mathbf{x}^t \)
  - Update parameters:
    \[ \mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta_t \nabla f(\mathbf{w}^{(t)}, \mathbf{x}^t) \]

- Works on the online learning setting!
- Complexity of each gradient step is constant in number of examples!
- In general, step size changes with iterations
  \( \eta_t \) decreases with iterations, from theory typically:
  \[ \eta_t = \frac{c}{t} \]
What you should know…

- Classification: predict discrete classes rather than real values
- Logistic regression model: Linear model
  - Logistic function maps real values to $[0,1]$
- Optimize conditional likelihood
- Gradient computation
- Overfitting
- Regularization
- Regularized optimization
- Cost of gradient step is high, use stochastic gradient descent

Boosting
Fighting the bias-variance tradeoff

- **Simple (a.k.a. weak) learners are good**
  - e.g., naïve Bayes, logistic regression, decision stumps (or shallow decision trees)
  - Low variance, don’t usually overfit too badly

- **Simple (a.k.a. weak) learners are bad**
  - High bias, can’t solve hard learning problems

Can we make weak learners always good???
- No!!
- But often yes...

Voting (Ensemble Methods)

Instead of learning a single (weak) classifier, learn **many weak classifiers** that are good at different parts of the input space

- **Output class:** (Weighted) vote of each classifier
  - Classifiers that are most “sure” will vote with more conviction
  - Classifiers will be most “sure” about a particular part of the space
  - On average, do better than single classifier:

\[
H(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right)
\]

**E.g.,**
\[
h_t(x) = \begin{cases} 
+1 & \text{if } x_i = 1 \\
0 & \text{if } x_i = 0
\end{cases}
\]

**But how do you ???**
- force classifiers to learn about different parts of the input space?
- weigh the votes of different classifiers?
Boosting [Schapire, 1989]

- Idea: given a weak learning alg, run it multiple times on (reweighted) training data, then let learned classifiers vote

- On each iteration $t$:
  - weight each training example by how incorrectly it was classified
  - Learn a hypothesis $h_t$
  - A strength for this hypothesis $\alpha_t$

- Final classifier:
  $H(x) = \text{Sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right)$

- Practically useful
- Theoretically interesting

Learning from weighted data

- Sometimes not all data points are equal
  - Some data points are more equal than others

- Consider a weighted dataset
  - $D(j)$ – weight of $j$th training example $(x_j, y_j)$
  - Interpretations:
    - $j$th training example counts as $D(j)$ examples
    - If I were to “resample” data, I would get more samples of “heavier” data points

- Now, in all calculations, whenever used, $j$th training example counts as $D(j)$ “examples”

\[
\text{for gradient descent}
\] \[
\omega(t+1) \leftarrow \omega(t) - \eta \sum_{j=1}^{N} D(j) \ell \left( w_j, x_i \right)
\]
AdaBoost

- Initialize weights to uniform dist: \( D_1(j) = \frac{1}{N} \)
- For \( t = 1 \ldots T \)
  - Train weak learner \( h_t \) on distribution \( D_t \) over the data
  - Choose weight \( \alpha_t \)
  - Update weights:
    \[
    D_{t+1}(j) = \frac{D_t(j) \exp(-\alpha_t y^j h_t(x^j))}{Z_t}
    \]
    Where \( Z_t \) is normalizer:
    \[
    Z_t = \sum_{j=1}^{N} D_t(j) \exp(-\alpha_t y^j h_t(x^j))
    \]
- Output final classifier:
  \[
  H(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right)
  \]

Picking Weight of Weak Learner

- Weigh \( h_t \) higher if it did well on training data (weighted by \( D_t \)):
  \[
  \alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)
  \]
  Magic: \( \alpha_t \) works like a perfect learner on weighted data => \( \epsilon_t \) ~= 0 => ignore \( h_t \)
  - If \( \epsilon_t = 0 \) => \( h_t \) perfect on weighted data
  - If \( \epsilon_t < \frac{1}{2} \) => \( h_t \) works better than random guess
  - If \( \epsilon_t > \frac{1}{2} \) => \( h_t \) is no better than random guess
  - If \( \epsilon_t = \frac{1}{2} \) => \( h_t \) is same as random guess
- Where \( \epsilon_t \) is the weighted training error:
  \[
  \epsilon_t = \frac{1}{N} \sum_{j=1}^{N} D_t(j) \mathbb{1}[h_t(x^j) \neq y^j]
  \]
Training error of final classifier is bounded by:

$$\frac{1}{N} \sum_{j=1}^{N} \mathbb{I}[H(x^j) \neq y^j] \leq \frac{1}{N} \sum_{j=1}^{N} \exp(-y^j f(x^j))$$

Where

\[ f(x) = \sum_{t} \alpha_t h_t(x); \quad H(x) = \text{sign}(f(x)) \]

Why choose $\alpha_t$ for hypothesis $h_t$ this way?

[Schapire, 1989]
Training error of final classifier is bounded by:

\[
\frac{1}{N} \sum_{j=1}^{N} \mathbb{I}[H(x_j) \neq y_j] \leq \frac{1}{N} \sum_{j=1}^{N} \exp(-y_j f(x_j)) = \prod_{t=1}^{T} Z_t
\]

Where \( f(x) = \sum_t \alpha_t h_t(x); H(x) = \text{sign}(f(x)) \)

If we minimize \( \prod_t Z_t \), we minimize our training error.

AdaBoost tightens this bound greedily, by choosing \( \alpha_t \) and \( h_t \) on each iteration to minimize \( Z_t \):

\[
Z_t = \sum_{j=1}^{N} D_t(j) \exp(-\alpha_t y^j h_t(x^j))
\]

[Schapire, 1989]