Logistic Regression: Online, Lazy, Kernelized, Sequential, etc.

Harsha Veeramachaneni

Thomson Reuter Research and Development

April 1, 2010

What is Logistic Regression?

- It's called regression but it's actually for classification.
- Assume we have a classification problem[1] from $\mathbf{x} \in R^d$ to $\mathbf{y} \in \{0, 1\}$.
- Assume further that we want the classifier to output posterior class probabilities.

Example: Say we have $\mathbf{x} = [\mathbf{x_1}, \mathbf{x_2}]$ and a binary classification problem. We would like a classifier that can compute $p(y = 0|x)$ and $p(y = 1|x)$

[1] Everything we say applies to multi-class problems but we'll restrict the talk to binary for simplicity

In the Beginning There Was a Model [2]

- Say we have two scoring functions $s_0(x)$ and $s_1(x)$ and we write $p(y = i|x) = \frac{s_i(x)}{s_0(x)+s_1(x)}$.
- As long as $s_i(x) > 0$ we obtain a probability like number.
- Say we project x along some vector and map the projection to a positive quantity.
- For example, $s_i(x) = e^{w_i{}^T x}$

[2] All models are wrong. Some are useful.

Modeling Continued

According to our definition above

$$
\begin{aligned}
p(y = 1|x) &= \frac{s_1(x)}{s_0(x) + s_1(x)} \\
&= \frac{exp(w_1^T x)}{exp(w_0^T x) + exp(w_1^T x)} \\
&= \frac{exp((w_1 - w_0)^T x)}{1 + exp((w_1 - w_0)^T x)} \\
&= \frac{exp(\beta^T x)}{1 + exp(\beta^T x)}
\end{aligned}
$$

where $\beta = w_1 - w_0$

$f(z) = \frac{exp(z)}{1+exp(z)}$ is called the logistic function.

Model

Therefore

$$p(y = 1|x) = \frac{exp(\beta^T x)}{1 + exp(\beta^T x)}$$

And

$$p(y = 0|x) = \frac{1}{1 + exp(\beta^T x)}$$

For our two feature example we have

$$p(y = 1|x) = \frac{exp(\beta_1 x_1 + \beta_2 x_2)}{1 + exp(\beta_1 x_1 + \beta_2 x_2)}$$

Logistic Function Pictures

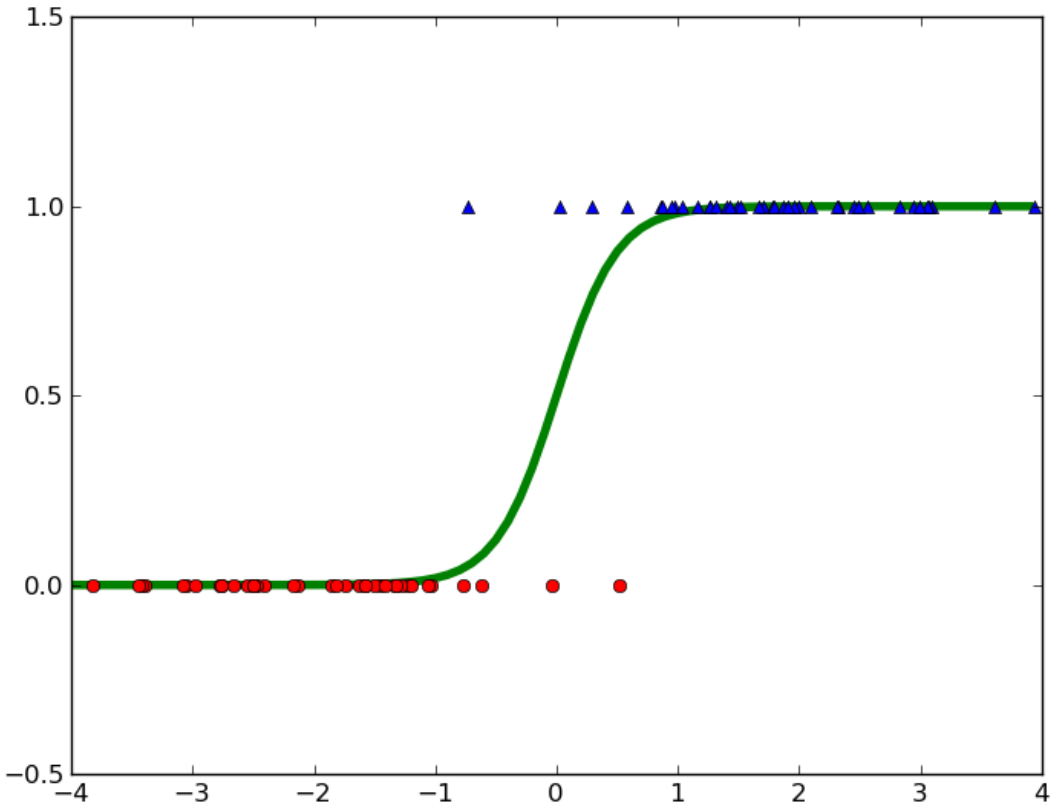


Figure: Logistic function in one variable.

Adding in a Bias Term

Once we can compute $p(y|x)$ we can get a minimum error-rate classifier by assigning all feature vectors with $p(y = 1|x) > 0.5$ to class $y = 1$ and 0 otherwise.

What is $p(y = 1|x)$ when $x = \mathbf{0}$?

$p(y = 1|x) = \frac{exp(0)}{1+exp(0)} = 0.5$

Therefore the classification boundary always passes through the origin.

We change this behavior by augmenting our model with a bias term.

Adding in a Bias Term

So we write $p(y = 1|x) = \frac{exp(\beta_0 + \beta^T x)}{1 + exp(\beta_0 + \beta^T x)}$

For the two feature example it becomes

$p(y = 1|x) = \frac{exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2)}{1 + exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2)}$

We can rewrite this as

$p(y = 1|x) = \frac{exp(\acute{\beta}^T \acute{x})}{1 + exp(\acute{\beta}^T \acute{x})}$

where $\acute{\beta} = [\beta_0, \beta_1, \beta_2]$ and $\acute{x} = [1, x_1, x_2]$

Logistic Function Pictures

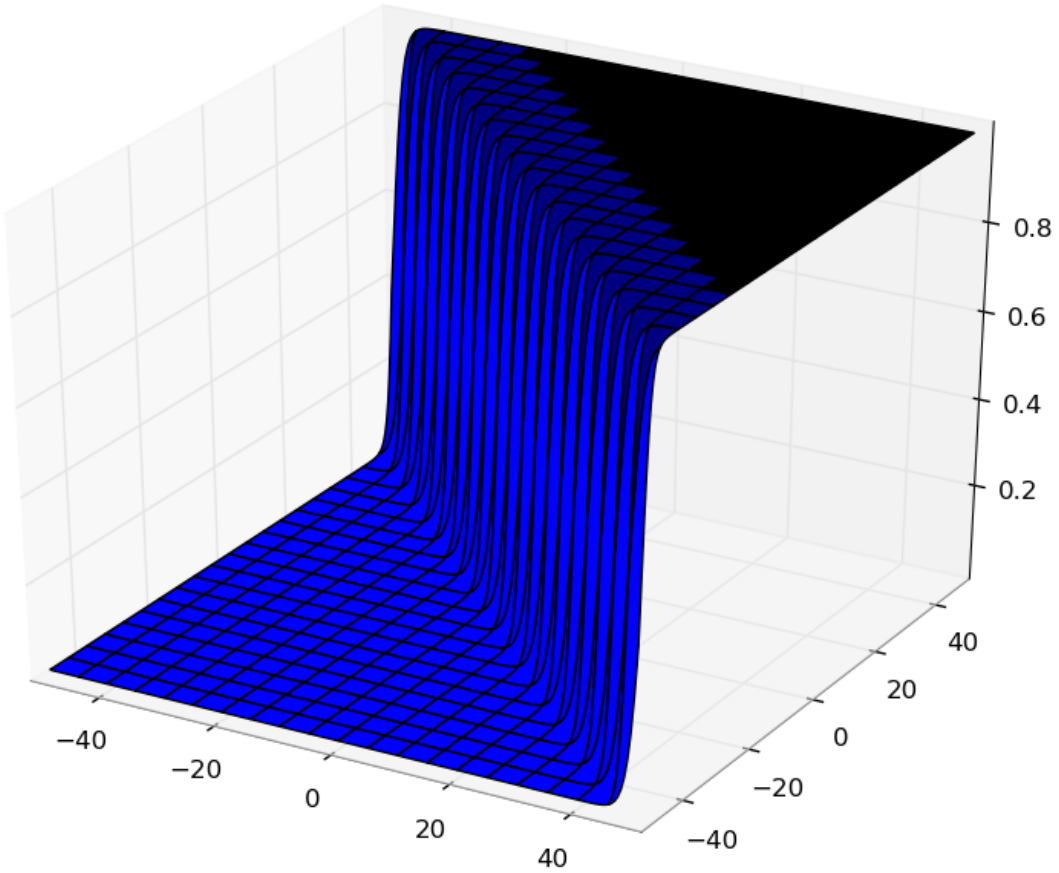


Figure: Logistic function in two variables, $\beta_0 = 0$, $\beta_1 = 1.0$, $\beta_2 = 1.0$

Logistic Function Pictures

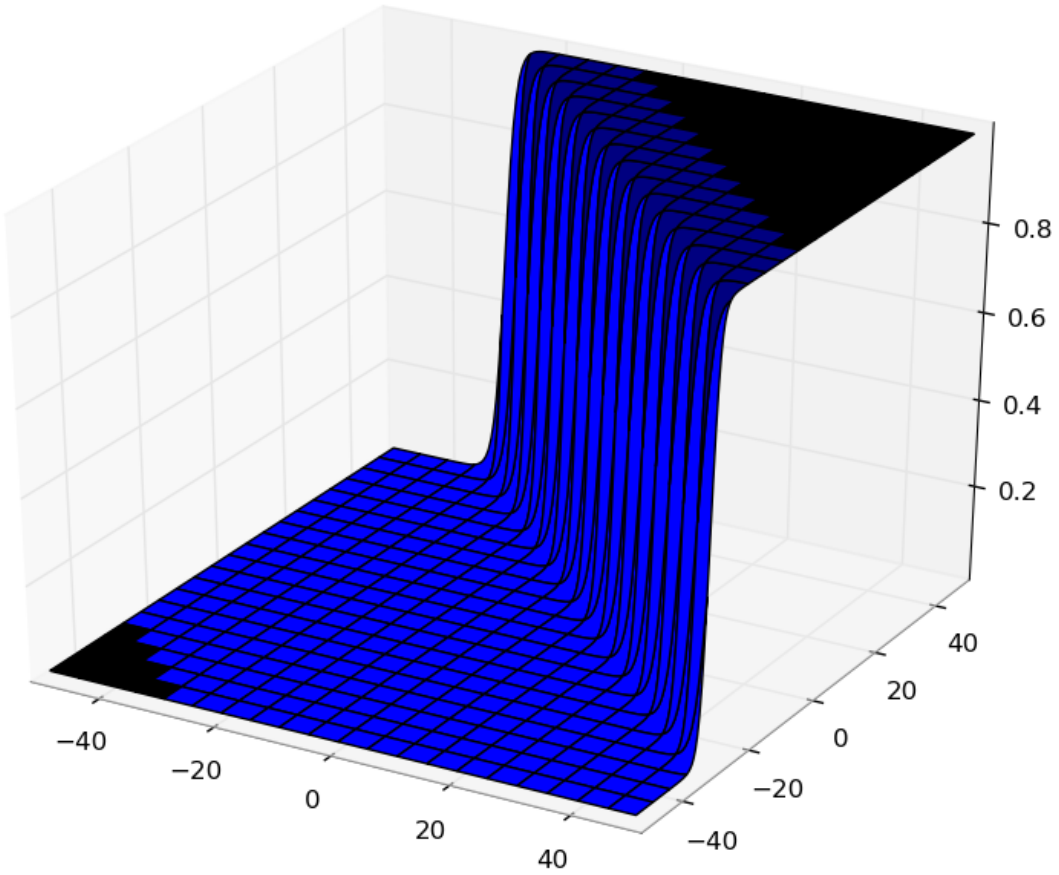


Figure: Logistic function in two variables, $\beta_0 = -20.0$, $\beta_1 = 1.0$, $\beta_2 = 1.0$

Logistic Function Pictures

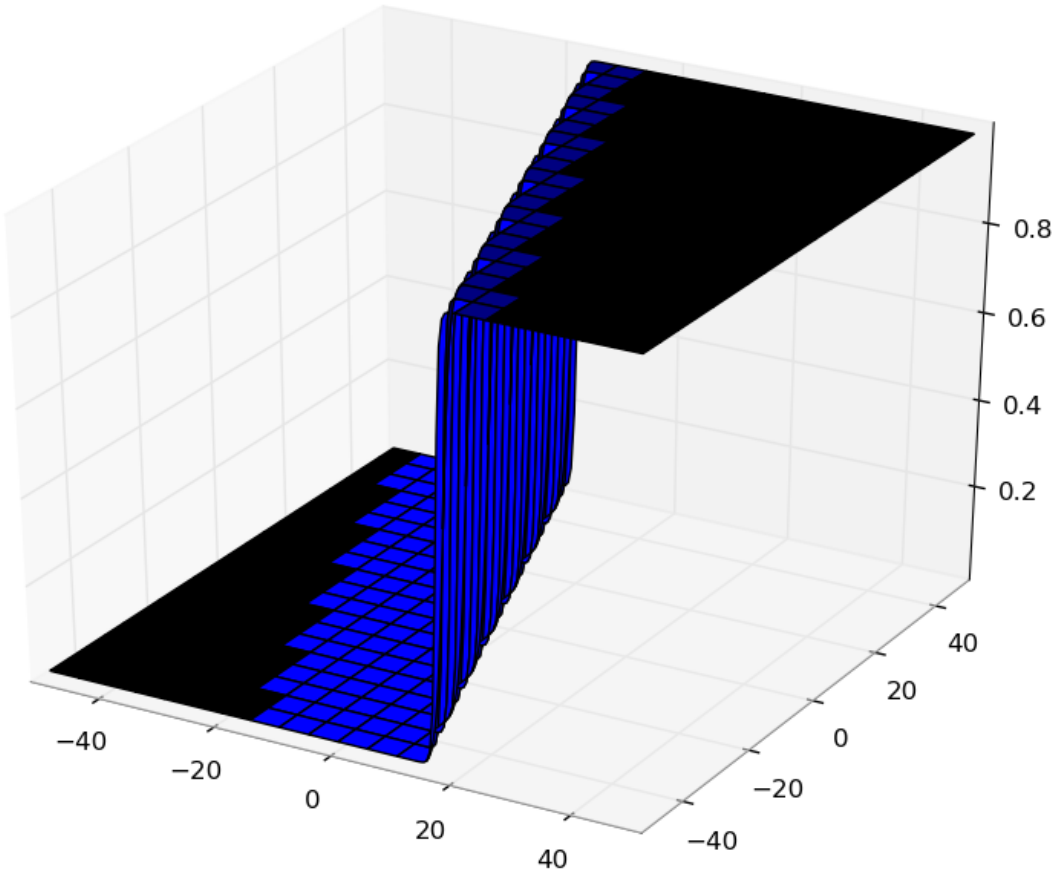


Figure: Logistic function in two variables, $\beta_0 = 0.0$, $\beta_1 = 3.0$, $\beta_2 = 1.0$

Logistic Function Pictures

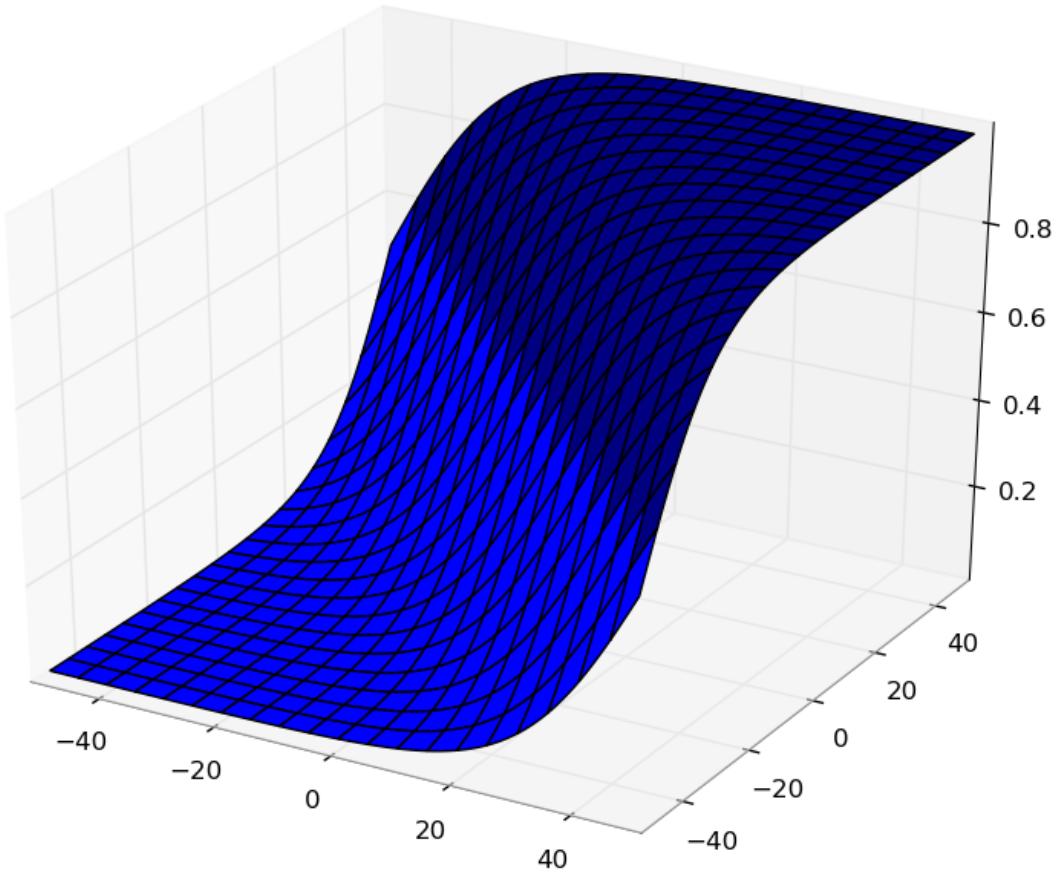


Figure: Logistic function in two variables, $\beta_0 = 0$, $\beta_1 = 0.1$, $\beta_2 = 0.1$

Logistic Function Pictures

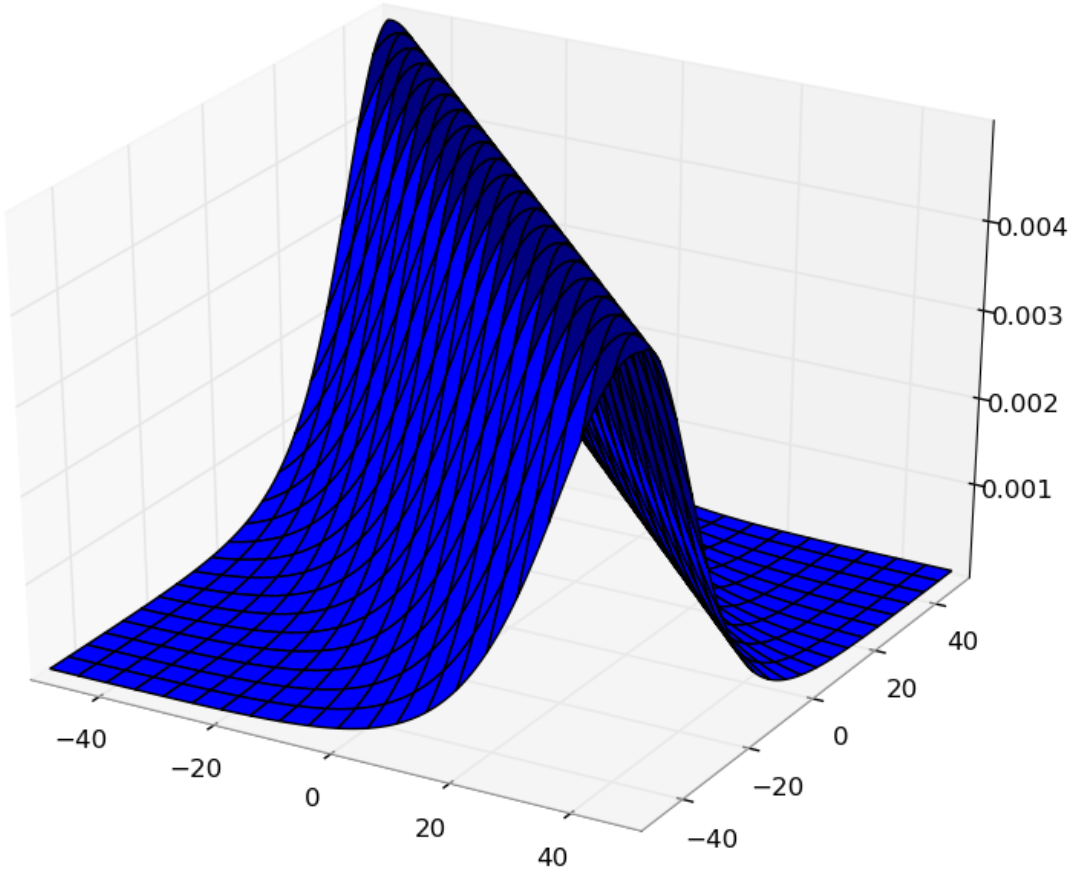


Figure: Derivative of the logistic function in two variables, $\beta_0 = 0$, $\beta_1 = 0.1$, $\beta_2 = 0.1$

Decoding and Inference

At test time the given feature vector x is classified (decoded) as class 1 if

$$p(y = 1|x) > p(y = 0|x)$$

$$\frac{exp(\beta^T x)}{1 + exp(\beta^T x)} > \frac{1}{1 + exp(\beta^T x)}$$

$$exp(\beta^T x) > 1$$

$$\beta^T x > 0$$

Inference: If we need the probability, we need to compute the partition function $(1 + exp(\beta^T x))$

Some Justification for the Model

- Why exp()?
- The log-odds ratio
  $\log\left(\frac{p(y=1|x)}{p(y=0|x)}\right) = \beta^T x$
  is an *affine* function of the observations.
- The log-odds ratio is an affine function of the sufficient statistics if the class-conditional distributions are from a fixed exponential family.
- The converse also true[3].
- Logistic regression same as MAXENT classification (under the right constraints on the features).

[3] Banerjee (2007), *An Analysis of Logistic Models: Exponential Family Connections and Online Performance.*

1. Logistic Regression (Plain and Simple)
   - Introduction to logistic regression

2. Learning Algorithm
   - Log-Likelihood
   - Gradient Descent
   - Lazy Updates

3. Kernelization

4. Sequence Tagging

Learning

Suppose we are given a labeled data set

$$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

From this data we wish to learn the logistic regression parameter (weight) vector β.

Learning

If we assume that the labeled examples are i.i.d., the likelihood of the data is

$$l(D;\beta) = p(D|\beta) = \prod_{i=1}^{N} p(y = y_i|x_i;\beta)$$

Or the log-likelihood is

$$L(D;\beta) = \log l(D;\beta) = \sum_{i=1}^{N} \log p(y = y_i|x_i;\beta)$$

Maximum Likelihood (ML) Estimation

The Maximum Likelihood (ML) estimate of the parameter vector is given by

$$\hat{\beta} = \arg\max_{\beta} p(D|\beta) = \arg\max_{\beta} L(D|\beta)$$

$$= \arg\max_{\beta} \sum_{i=1}^{N} \log p(y = y_i|x_i; \beta)$$

What happens if there is a feature which is zero for all the examples except one?

Maximum A Posteriori (MAP) Estimation

We place a prior on β that penalizes large values.

For example a Gaussian prior: $p(\beta) = N(\beta; \mathbf{0}, \sigma^2\mathbf{I})$

The posterior probability of the β is given by

$$p(\beta|D) = \frac{p(D|\beta)p(\beta)}{p(D)}$$

So instead of ML we can obtain the Maximum A Posteriori (MAP) estimate

$$\begin{aligned}\hat{\beta} &= \arg\max_{\beta} p(\beta|D) = \arg\max_{\beta} p(D|\beta)p(\beta) \\ &= \arg\max_{\beta} \left(\log p(D|\beta) + \log p(\beta)\right) \\ &= \arg\max_{\beta} \left(\sum_{i=1}^{N} \log p(y = y_i|x_i; \beta) + \log p(\beta)\right)\end{aligned}$$

How is this maximization actually done?

Gradient Ascent

- Start with $\beta = \mathbf{0}$ and iteratively move along the direction of steepest ascent.
- Since the function we are optimizing is concave, any local maximum we find is also the global maximum.
- We need to be able to compute the gradient ($\nabla_\beta p(\beta|D)$) at any point.
- The choice of the size of the step is important.

Gradient Ascent

$$\text{Gradient} = \nabla_\beta p(\beta|D) = (\frac{\partial}{\partial \beta_0} p(\beta|D), \frac{\partial}{\partial \beta_1} p(\beta|D), \ldots)$$

$$\frac{\partial}{\partial \beta_j} p(\beta|D) = \frac{\partial}{\partial \beta_j} \{\sum_{i=1}^{N} \log p(y = y_i|x_i; \beta) + \log p(\beta)\}$$

So the update rule would be

$$\beta_j^{(t+1)} = \beta_j^{(t)} + \zeta \frac{\partial}{\partial \beta_j} p(\beta|D)$$

Stochastic Gradient Ascent

Gradient $= \nabla_\beta p(\beta|D) = (\frac{\partial}{\partial \beta_0} p(\beta|D), \frac{\partial}{\partial \beta_1} p(\beta|D), \ldots)$

We can write

$$\begin{aligned}
\frac{\partial}{\partial \beta_j} p(\beta|D) &= \frac{\partial}{\partial \beta_j} \{\sum_{i=1}^{N} \log p(y = y_i|x_i;\beta) + \log p(\beta)\} \\
&= \sum_{i=1}^{N} \frac{\partial}{\partial \beta_j} (\log p(y = y_i|x_i;\beta)) + \frac{\partial}{\partial \beta_j} \log p(\beta) \\
&= \sum_{i=1}^{N} \left(\frac{\partial}{\partial \beta_j} \log p(y = y_i|x_i;\beta) + \frac{1}{N} \frac{\partial}{\partial \beta_j} \log p(\beta) \right) \\
&= \sum_{i=1}^{N} g(x_i, y_i) \qquad \text{(At a fixed value of } \beta)
\end{aligned}$$

Stochastic Gradient Ascent [4]

The update rule is

$$\begin{aligned}
\beta_j^{(t+1)} &= \beta_j^{(t)} + \zeta \frac{\partial}{\partial \beta_j} p(\beta|D) \\
&= \beta_j^{(t)} + \zeta \sum_{i=1}^{N} g(x_i, y_i) \\
&= \beta_j^{(t)} + \underbrace{N\zeta}\underbrace{\left(\frac{1}{N} \sum_{i=1}^{N} g(x_i, y_i) \right)} \\
&\approx \beta_j^{(t)} + \eta E[g(x, y)]
\end{aligned}$$

[4] Usually talked about as Stochastic Gradient Descent (SGD), because minimization is canonical.

Stochastic Gradient Ascent

- We could randomly sample a small batch of L examples from the training set to estimate $E[g(x, y)]$.
- What if we went to extreme and made $L = 1$?
- We obtain online (incremental) learning.
- Can be shown to converge to the same optimum under some reasonable assumptions.

Stochastic Gradient Ascent

The learning algorithm in its entirety is
vspace0.1in

Algorithm 1: SGD

Input: $D = \{x_i, y_i\}_{i=1...N}$, η, MaxEpochs, Convergence Threshold
Output: β
$\beta \leftarrow \mathbf{0}$
for $e = 1 : MaxEpochs \,||\, Log\text{-}Likelihood\ Convergence$ **do**
  for $i = 1 : N$ **do**
    $\beta \leftarrow \beta + \eta g(x_i, y_i)$
return β

If $\mathbf{x} \in R^d$, complexity is $O(Nd)$.

Gradient Calculation

Let us now compute $g(x_i, y_i)$.

$$g(x_i, y_i) = \frac{\partial}{\partial \beta_j} \log p(y = y_i | x_i; \beta) + \frac{1}{N} \frac{\partial}{\partial \beta_j} \log p(\beta)$$

Gradient Calculation

If $y_i = 1$

$$\begin{aligned}
\frac{\partial}{\partial \beta_j} \log p(y = y_i | x_i; \beta) &= \frac{\partial}{\partial \beta_j} \log \left(\frac{exp(\beta^T x_i)}{1 + exp(\beta^T x_i)} \right) \\
&= \frac{\partial}{\partial \beta_j} \beta^T x_i - \frac{\partial}{\partial \beta_j} \log(1 + exp(\beta^T x_i)) \\
&= x_{ij} - \frac{exp(\beta^T x)}{1 + exp(\beta^T x)} x_{ij} \\
&= x_{ij} - p(y = 1 | x_i; \beta) x_{ij} \\
&= x_{ij}(1 - p(y = 1 | x_i; \beta))
\end{aligned}$$

Gradient Calculation

Similarly, if $y_i = 0$

$$\begin{aligned}
\frac{\partial}{\partial \beta_j} \log p(y = y_i | x_i; \beta) &= \frac{\partial}{\partial \beta_j} \log \left(\frac{1}{1 + exp(\beta^T x_i)} \right) \\
&= -\frac{\partial}{\partial \beta_j} \log(1 + exp(\beta^T x_i)) \\
&= -\frac{exp(\beta^T x_i)}{1 + exp(\beta^T x_i)} x_{ij} \\
&= -x_{ij}(1 - p(y = 0 | x_i; \beta))
\end{aligned}$$

Gradient Calculation

Therefore we may write

$$\frac{\partial}{\partial \beta_j} \log p(y = y_i | x_i; \beta) = (-1)^{y_i+1} x_{ij} (1 - p(y = y_i | x_i; \beta))$$

Note: β is in the span of the example feature vectors. (Representer Theorem)

Question: How many β_j will be updated if for an example all its feature values are zero ($x_i = 0$)?

Lazy Update

- If we ignore the prior,
  for each example, we only need to update the coordinates of β where the feature vector is non-zero.
- Makes learning very fast if the feature vectors are sparse.
- Complexity reduces to $O(Ns)$, where s is the average number of non-zero feature values.
- What happens when we add the prior?

Lazy Update

If the prior is Gaussian

$$p(\beta) = A \times exp(-0.5\frac{\|\beta\|^2}{\sigma^2})$$

where A does not depend on β. Therefore

$$\log p(\beta) = C + \log exp(-0.5\frac{\|\beta\|^2}{\sigma^2}) = C - 0.5\frac{\|\beta\|^2}{\sigma^2}$$

$$\begin{aligned}\frac{\partial}{\partial \beta_j} \log p(\beta) &= \frac{\partial}{\partial \beta_j}\left(C - 0.5\frac{\|\beta\|^2}{\sigma^2}\right) \\ &= -\frac{1}{\sigma^2}\beta_j\end{aligned}$$

Because of the contribution of the prior to the gradient, *all* the coordinates of β need to be updated.

Lazy Update

- The solution is to update lazily[5].
- Still only update the coordinates corresponding to non-zero features.
- For each coordinate keep track of number of examples u seen since last update.
- For each non-zero feature coordinate penalize the β value u times.
- Technical detail – clip to avoid crossing zero.

[5] Carpenter (2008), *Lazy sparse stochastic gradient descent for regularized multinomial logistic regression*

One Slide about Multi-Class Logistic Regression

- Say the class labels come from a finite set $y \in \{0, 1, \ldots, M-1\}$.
- We define [6]
  $p(y|x) \propto exp(\beta_y^T x)$
- Therefore $p(y|x) = \frac{exp(\beta_y^T x)}{\sum_y exp(\beta_y^T x)}$.
- The above discussion about learning/SGD/lazy updates generalizes trivially.
- To classify, pick the y with the highest $\beta^T x$.
- The partition function has M terms.

[6] One of the βs is redundant. We can make one of them 0 by subtracting it from the rest.

1. Logistic Regression (Plain and Simple)
   - Introduction to logistic regression
2. Learning Algorithm
   - Log-Likelihood
   - Gradient Descent
   - Lazy Updates
3. Kernelization
4. Sequence Tagging

Logistic Regression with Kernels

- As we said before, β is in the span of the examples, i.e., $\beta = \sum_{i=1}^{N} \alpha_i x_i$
- So in order to classify a test feature vector x we compute $\beta^T x = \sum_{i=1}^{N} \alpha_i x_i^T x$
- If we have a high-dimensional feature map $\phi(x)$ and learn a logistic regression model in this high-dimensional space, we will compute $\beta^T \phi(x) = \sum_{i=1}^{N} \alpha_i \phi(x_i)^T \phi(x) = \sum_{i=1}^{N} \alpha_i k(x_i, x)$
- So if we have a positive definite kernel $k(.,.)$, we can kernelize logistic regression.

  e.g. Gaussian radial basis function (rbf) kernel
  $k(x_i, x) = exp(-\lambda \|x_i - x\|^2)$

Logistic Regression with Kernels

- For SVMs most of the $\alpha_i = 0$ (sparse model).
- For logistic regression the model is not sparse, which is a problem because
- Learning involves estimating all the N α_is, and classification involves computing the kernel of the test vector with *all* the training vectors.
- Training is slow, the model is very large and testing will be very slow.

Sparse Kernel Logistic Regression: Two Common Approaches

- Greedily select a subset of the α_i to be non-zero, so the full model is best approximated by the reduced model (Import Vector Machine[7]).
- Allow a *random* subset of α_i to be non-zero and maximize the log-likelihood on the entire training data.

[7] Zhu & Hastie (2001), *Kernel Logistic Regression and the Import Vector Machine*

Sparse Kernel Logistic Regression: Our Approach

- What are we doing when we pick a random subset of αs (say n of them)?
- We are writing

$$\begin{aligned}\beta_i^T \phi(x) &= \sum_{i=1}^{N} \alpha_i \phi(x_i)^T \phi(x) \\ &= \sum_{i=1}^{n} \alpha_i k(x_i, x) = \sum_{i=1}^{n} \alpha_i z_i = \alpha^T z\end{aligned}$$

- Essentially we *transformed* the feature vector x into an n-dimensional feature vector z.
- In this transformed feature space we are doing a plain linear logistic regression.

Sparse Kernel Logistic Regression: Our Approach

$$\beta_i^T \phi(x) = \sum_{i=1}^{N} \alpha_i \phi(x_i)^T \phi(x)$$
$$= \sum_{i=1}^{n} \alpha_i k(x_i, x) = \sum_{i=1}^{n} \alpha_i z_i = \alpha^T z_i$$

- Each coordinate in this *transformed* space is the kernel value of the test vector to one example in our subset.
- We can think of our n training feature vectors as *prototypes*

Sparse Kernel Logistic Regression: Our Approach

- There is no reason why the prototypes should be from the training data.
- We can pick any points in our feature space to serve as prototypes.
- We can even learn these prototypes.
- For differentiable kernels we can learn them by gradient ascent.
- In fact we go further and learn the kernel parameters as well (e.g. the scale parameter for the rbf kernel).

Sparse Prototype Kernel Logistic Regression

Assume we have n prototypes $\{u_1, \ldots, u_n\}$. We have

$$p(y_i = y|x_i; \alpha) = \frac{exp(\alpha_y^T k(x, u_i))}{\sum_y exp(\alpha_y^T k(x, u_i))}$$
$$= \frac{exp(\alpha_y^T z_i)}{\sum_y exp(\alpha_y^T z_i)}$$

- If the kernel is differentiable w.r.t. to u_i and its parameter ...
- we can compute the gradient of the log-likelihood $\log p(y = y_i|x_i; \alpha)$ and ...
- update both the prototypes and the kernel params by stochastic gradient ascent.

Gradient Calculation w.r.t. Prototypes

$$\frac{\partial}{\partial u_l} \log p(y = c|x_i) = \frac{\partial}{\partial u_l} \left(\alpha_c^T z_i - \log(\sum_{y=1}^{M} exp(\alpha_y^T z_i)) \right)$$

$$= \frac{\partial}{\partial u_l} \left(\sum_{j=1}^{n} \alpha_{cj} z_{ij} - \log(\sum_{y} exp(\sum_{j=1}^{n} \alpha_{yj} z_{ij})) \right)$$

$$= \alpha_{cl} \frac{\partial}{\partial u_l} z_{il} - \sum_{y} p(y|x_i) \sum_{j=1}^{n} \alpha_{yj} \frac{\partial}{\partial u_l} z_{ij}$$

$$= \left(\alpha_{cl} - \sum_{y} p(y|x_i) \alpha_{yl} \right) \frac{\partial}{\partial u_l} z_{ij}$$

Gradient Calculation w.r.t. Kernel Parameter

If the kernel $k(.,.)$ is parameterized by λ,

$$\begin{aligned}
\frac{\partial}{\partial \lambda} \log p(y = c|x_i) &= \frac{\partial}{\partial \lambda} \left(\alpha_c^T z_i - \log(\sum_{y=1}^{M} exp(\alpha_y^T z_i)) \right) \\
&= \frac{\partial}{\partial \lambda} \left(\sum_{j=1}^{n} \alpha_{cj} z_{ij} - \log(\sum_{y} exp(\sum_{j=1}^{n} \alpha_{yj} z_{ij})) \right) \\
&= \sum_{j=1}^{n} \alpha_{cj} \frac{\partial}{\partial \lambda} z_{ij} - \sum_{y} p(y|x_i) \sum_{j=1}^{n} \alpha_{yj} \frac{\partial}{\partial \lambda} z_{ij} \\
&= \sum_{j=1}^{n} \left(\alpha_{cj} - \sum_{y} p(y|x_i) \alpha_{yj} \right) \frac{\partial}{\partial \lambda} z_{ij}
\end{aligned}$$

Gradient Calculation – Gaussian Radial Basis Function Kernel

For the Gaussian r.b.f. kernel $z_{il} = k(x_i, u_l) = exp(-\lambda \|x_i - u_l\|^2)$.

$$\begin{aligned} \frac{\partial}{\partial u_l} z_{ij} &= \frac{\partial}{\partial u_l} exp(-\lambda \|x_i - u_l\|^2) \\ &= 2\lambda (x_i - u_l) z_{il} \end{aligned}$$

$$\begin{aligned} \frac{\partial}{\partial \lambda} z_{ij} &= \frac{\partial}{\partial \lambda} exp(-\lambda \|x_i - u_l\|^2) \\ &= -\|x_i - u_l\|^2 z_{il} \end{aligned}$$

Sparse Prototype Kernel Logistic Regression

Algorithm 2: Prototype Kernel Logistic Regression

Input: $D = \{x_i, y_i\}_{i=1...N}$, η, MaxEpochs, Convergence Threshold

$\beta \leftarrow \mathbf{0}$

Initialize prototypes and kernel param.

for $e = 1 : MaxEpochs \,||\, Log\text{-}Likelihood\ Convergence$ **do**

  for $i = 1 : N$ **do**

    Update β by gradient ascent.

  for $i = 1 : N$ **do**

    Update prototypes by gradient ascent.

  for $i = 1 : N$ **do**

    Update kernel param by gradient ascent.

return β, prototypes, kernel param

Sparse Prototype Kernel Logistic Regression

- Done naively the updates cannot take advantage of feature vector sparsity anymore.
- Since our kernelization approach can be extended to any similarity function, we compute the rbf kernel value based on the distance along only the subset of features that are non-zero.
- That means that we need not visit the coordinates of the prototypes corresponding to non-zero feature values.
- We also update the prototypes lazily.
- For the r.b.f. kernel, the parameter update worked best when the kernel was parameterized as $k(x_i, u_l) = exp(-exp(\theta)\|x_i - u_l\|^2)$.

Demo Here

1. Logistic Regression (Plain and Simple)
   - Introduction to logistic regression
2. Learning Algorithm
   - Log-Likelihood
   - Gradient Descent
   - Lazy Updates
3. Kernelization
4. Sequence Tagging

Non-I.I.D. Data

- We've be assuming that the examples are generated i.i.d. from the distribution $p(x, y)$.
- In particular our model assumes $p(y_1 = a_1, \ldots, y_N = a_N | x_1, \ldots, x_N) = p(y_1 = a_1 | x_1) \times \ldots \times p(y_N = a_N | x_N)$
- For some problems (like Named Entity Tagging) such a model is too weak because we know that the label for an example depends on the labels for the neighboring ones.
- How can we incorporate dependencies among the labels into our model?
- In particular, say we want to model Markov dependencies.

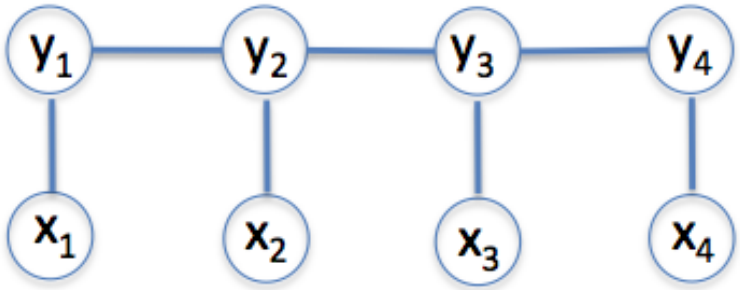
Markov Random Field

- We can describe conditional independences by a Markov Random Field (MRF).
- An MRF is a graph over the random variables of interest such that *a variable is conditionally independent of all other variables given its neighbors.*
- The Hammersley-Clifford theorem states that the density can be written as a product of *clique potentials* over all maximal cliques.
- Clique potentials are positive functions of the random variables that comprise the clique.
- A Conditional Random Field is a special kind of MRF, where we are interested in dependences (between labels) conditioned on some global random vector (observations)[8].

[8] Lafferty et. al. (2001), *Conditional random fields: Probabilistic models for segmenting and labeling sequence data*

Conditional Random Fields: Example

Consider the CRF in the following example

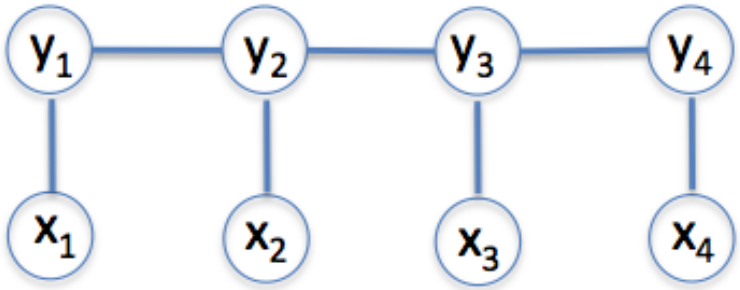


The probability that $P = p(y_1 = 0, y_2 = 1, y_3 = 1, y_4 = 0 | x_1, x_2, x_3, x_4)$ is given by
$P \propto \psi_{12}(0, 1, x) \times \psi_{23}(1, 1, x) \times \psi_{34}(1, 0, x)$

where $\psi_{i,j}(y_1, y_2, x)$ are the potential functions.

Conditional Random Fields: Example

Consider the CRF in the following example



Often we pick the same potential function $\psi_{i,j}(y_1, y_2, x) = \psi(y_1, y_2, x)$.

Moreover, since the potential is positive $\psi(y_1, y_2, x) = exp(f(y_1, y_2, x))$

If $f()$ is linear in x, $\psi(y_1, y_2, x) = exp(\beta^T_{y_1 y_2} x)$

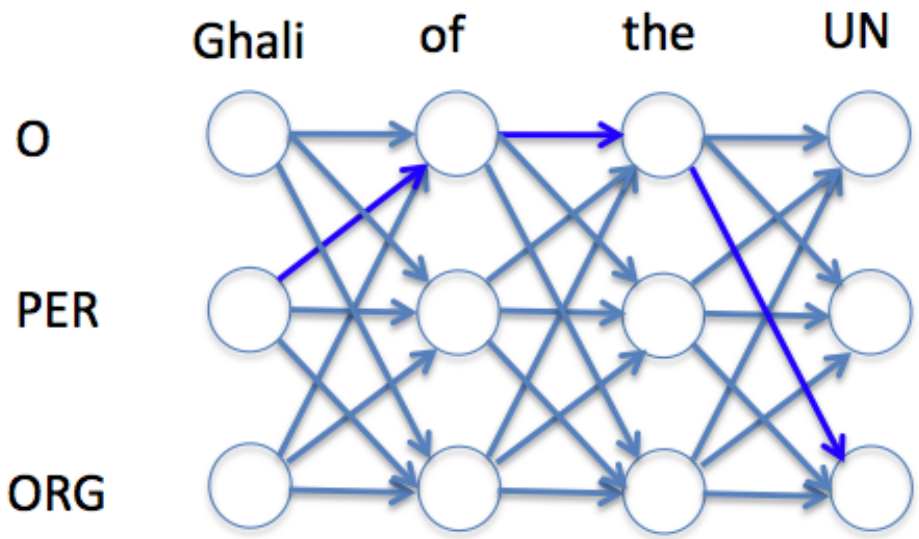
Looks very much like a logistic regression model.

How many terms does the partition function have?

Learning and Decoding for CRFs

- To learn CRFs by gradient descent, we need to compute the derivative of the log-likelihood.
- Again the derivative involves the partition function.
- The calculation of the partition function is efficient for chain CRFs (forward-backward algorithm).
- Although MALLET uses L-BFGS, CRFs can also be learned by stochastic gradient descent.
- For chain CRFs decoding can be done efficiently by dynamic programming (Viterbi algorithm).

Viterbi Decoding



- The weight on the arrows represents the benefit of transitioning from the previous label to the current label for that particular token.
- It is the logarithm of the potential function value for that clique ($\beta^T_{y_1 y_2} x$).
- To decode, we find the path with the maximum benefit by dynamic programming.

Sequence Tagging without CRFs

- We can use Viterbi decoding without the additional cost of forward-backward at training time by just learning a logistic regression to pairs of classes.
- The training sequence data is used to assign a class-pair label to each example.
- The class-pair model is smoothed with a single class model.
- At decoding time pretend that the model was learned under a CRF assumption.
- Often gives a better model because the assumption made by CRFs that the training sequences are drawn i.i.d. is violated.

Sequence Tagging using Logistic Regression

- Our approach gives similar (and sometimes better) accuracy for NER than CRFs (MALLET).
- Our un-optimized implementation trains 30-50 times faster.
- The code is being used in projects requiring incremental learning.
- The barrier for implementing active learning systems is now much lower.

Current and Near Term

- Minimax & Type-specific Priors for domain independence.
- Active Learning
- Ranking
- Semi-Supervised
- ....