Lecture 7 - More on Classification

Fall 2009
First, Some Review
Looking at the confidence in classification

- We can translate this response into a probability.
- We will use the *logistic function* to transform the response into a probability of the correct classification.
- We will assume that each point will be labeled with a label \( l \). \( l \) can take values +1 or −1.

\[
P[l = +1|\mathbf{x}] = \frac{1}{1 + \exp(-(ax + by + c))}
\]
Finding the line parameters

Now, for any set of line constants, we can find out the probability assigned to the correct label of each item in the training set.

\[
\prod_{i=1}^{N} P[l_i|x_i] = \prod_{i=1}^{N} \frac{1}{1 + \exp(-l_i(ax_i + by_i + c))}
\]

We’ve inserted \( l_i \) into the exponent because, if \( l_i = -1 \)

\[
P[l = -1|x] = \frac{1}{1 + \exp((ax + by + c))}
\]

We multiply the probabilities because we believe the points are drawn independently.

Note that for each item, this number tells us how much the model defined by that particular line believes that the ground-truth right answer is the actual right answer.
Finding the line

\[
\prod_{i=1}^{N} P[l_i | x_i] = \prod_{i=1}^{N} \frac{1}{1 + \exp(-l_i(ax_i + by_i + c))}
\]

▶ This is also called the likelihood of the data
▶ Ideally, this likelihood should be as close to 1 as possible.
▶ We can find the parameters of the line by looking for the line parameters that maximize this likelihood.
▶ In other words, find the set of line parameters that make the right answers have as high a probability as possible.
Finding the line

\[
\prod_{i=1}^{N} P[l_i | x_i] = \prod_{i=1}^{N} \frac{1}{1 + \exp(-l_i(ax_i + by_i + c))}
\]

▶ Before going on, we are going to do a quick math trick.

▶ Because the \( \log \) (natural logarithm or \( \ln \)) function is monotonically increasing (i.e. it is always increasing), the parameters that maximize the above equation will also maximize

\[
\log \left( \prod_{i=1}^{N} P[l_i | x_i] \right) = \sum_{i=1}^{N} - \log \left( 1 + \exp(-l_i(ax_i + by_i + c)) \right)
\]
Finding the line

\[
\log \left( \prod_{i=1}^{N} P[l_i | x_i] \right) = \sum_{i=1}^{N} -\log \left( 1 + \exp \left( -l_i(ax_i + by_i + c) \right) \right)
\]

▶ Similarly, we can multiply this equation by $-1$ to change from a maximization problem to a minimization problem
▶ Leading to a function that we will call a “loss function” or “cost function” and can be denoted as $L$:

\[
L = \sum_{i=1}^{N} \log \left( 1 + \exp \left( -l_i(ax_i + by_i + c) \right) \right)
\]

▶ Our goal is to find the line parameters that minimize $L$. 
Minimizing $L$

- We can minimize $L$ by using its gradient:

$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial a} \\ \frac{\partial L}{\partial b} \\ \frac{\partial L}{\partial c} \end{bmatrix}$$

- The gradient can be viewed as a vector that points in the direction of steepest ascent.
- So, the negative gradient points in the direction of steepest descent.
An algorithm for minimizing $L$

- This leads to a simple algorithm for optimizing the line parameters.
- We’ll create a vector

$$p = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

- The steps are:
  1. Initialize $p$ to some value $p_0$
  2. Repeat these steps:
     2.1 Calculate $\nabla L$ using the current value of $p$ (The value of the gradient depends on $p$!)
     2.2 $p \leftarrow p + \eta(-\nabla L)$

- We go in the direction of the negative gradient because that is the direction of steepest descent.
More on Loss Functions

- Let’s look at the loss function that we are minimizing

\[ L(x) = \log(1 + e^{-x}) \]

- If the label is +1 then we are encouraging our linear classifier to return a positive value.
- Loss grows approximately linearly as it gets more and more negative.
The Log-Loss

\[ L(x) = \log(1 + e^{-x}) \]

- This can be thought of as a modification of the zero-one loss
- The 0-1 loss, says, “I only care if I make a mistake!”
The Exponential Loss

\[ L(x) = e^{-x} \]

▶ This is an upper-bound on the 0-1 loss
Boosting Approach to Finding Classification Parameters

In the logistic regression approach discussed in the previous lecture, we gathered all of our features, then optimized the weights.

What if we had millions of features? We couldn’t load those into memory and optimize using gradient descent?

What if we added features greedily? We could then consider tons of features.

This often called boosting.
Describing This Mathematically

▶ To begin with, we will minimize the exponential loss.
▶ We will also define a new term, $F(\vec{x}_i)$:

$$F(\vec{x}_i) = \sum_{j=1}^{N_f} a_j \phi_j(\vec{x}_i)$$

▶ $F(\vec{x}_i)$ is the classifier. The predicted label of any sample is $\text{sign}(F(\vec{x}_i))$
▶ Each $\phi(\cdot)$ is a feature. Each $a_j$ is a weight.
▶ In the boosting algorithm, we will build up $F(\cdot)$ one feature at a time.
Adding Features

Let’s assume that we have defined \( F(\vec{x}_i) \) for \( j \) features and have chosen \( \phi_{j+1}(\cdot) \).

\[
F(\vec{x}_i) = \sum_{j=1}^{N_f} a_j \phi_j(\vec{x}_i)
\]

Now we need to choose \( a_{j+1} \).

We will minimize the exponential loss for \( N \) training examples, with respect to \( a_{j+1} \):

\[
L(a_{j+1}) = \sum_{i=1}^{N} \exp \left( -l_i [F(\vec{x}_i) + a_{j+1} \phi_{j+1}(\vec{x}_i)] \right)
\]
Finding the parameter

\[ L(a_{j+1}) = \sum_{i=1}^{N} \exp (-l_i [F(\tilde{x}_i) + a_{j+1} \phi_{j+1}(\tilde{x}_i)]) \]

- We need to find \( a_{j+1} \)
- We could use gradient descent, or we could do a Newton step
- In a Newton step, we will use a Taylor series to approximate \( L(a_{j+1}) \) with a quadratic function, then solve that quadratic function to find \( a_{j+1} \).

\[ f(x) \approx f(b) + f'(b)(x - b) + \frac{f''(b)}{2}(x - b)^2 \]

- This is similar to Newton’s algorithm for minimizing functions.
Finding the parameter

\[ L(a_{j+1}) = \sum_{i=1}^{N} \exp \left( -l_i[F(\bar{x}_i) + a_{j+1}\phi_{j+1}(\bar{x}_i)] \right) \]

\[ L'(a_{j+1}) = \sum_{i=1}^{N} -l_i\phi_{j+1}(\bar{x}_i) \exp \left( -l_i[F(\bar{x}_i) + a_{j+1}\phi_{j+1}(\bar{x}_i)] \right) \]

\[ L''(a_{j+1}) = \sum_{i=1}^{N} \phi_{j+1}^2(\bar{x}_i) \exp \left( -l_i[F(\bar{x}_i) + a_{j+1}\phi_{j+1}(\bar{x}_i)] \right) \]

If we do the Taylor expansion around \( a_{j+1} = 0 \), then

\[ L(a_{j+1}) \approx \sum_{i=1}^{N} \exp \left( -l_i[F(\bar{x}_i)] \right) + a_{j+1}L'(0) + a_{j+1}^2 \frac{L''(0)}{2} \]

Differentiating this, we can solve for \( a_{j+1} \)

\[ a_{j+1} = -\frac{L'(0)}{L''(0)} = \frac{\sum_{i=1}^{N} l_i\phi_{j+1}(\bar{x}_i) \exp \left( -l_i[F(\bar{x}_i)] \right)}{\sum_{i=1}^{N} \phi_{j+1}^2(\bar{x}_i) \exp \left( -l_i[F(\bar{x}_i)] \right)} \]
A Regression View

\[ L(a_{j+1}) \approx \sum_{i=1}^{N} \exp (-l_i[F(\vec{x}_i)]) + a_{j+1}L'(0) + a_{j+1}^2 \frac{L''(0)}{2} \]

- This can also be looked at from a regression point of view:

\[ L(a_{j+1}) \approx \sum_{i=1}^{N} \exp (-l_i[F(\vec{x}_i)]) (\phi_{j+1}(\vec{x}_i)a_{j+1} - l_i)^2 \]

- We minimize this with respect to \( a_{j+1} \).
- This term \( \exp (-l_i[F(\vec{x}_i)]) \) can be thought of as a weight for each training example that will be updated at each round.
Boosting

- This is called the GentleBoost algorithm
- We have to choose $\phi_{j+1}(\cdot)$. Usually choose that greedily by searching over a bunch of different features that have been thresholded.