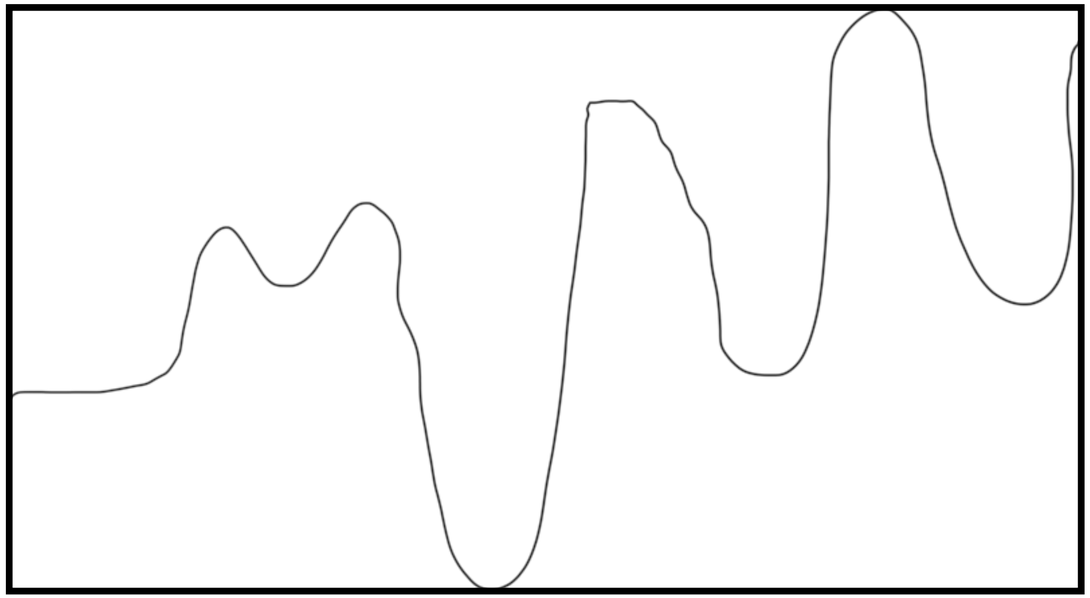
Logistic Regression Part 2

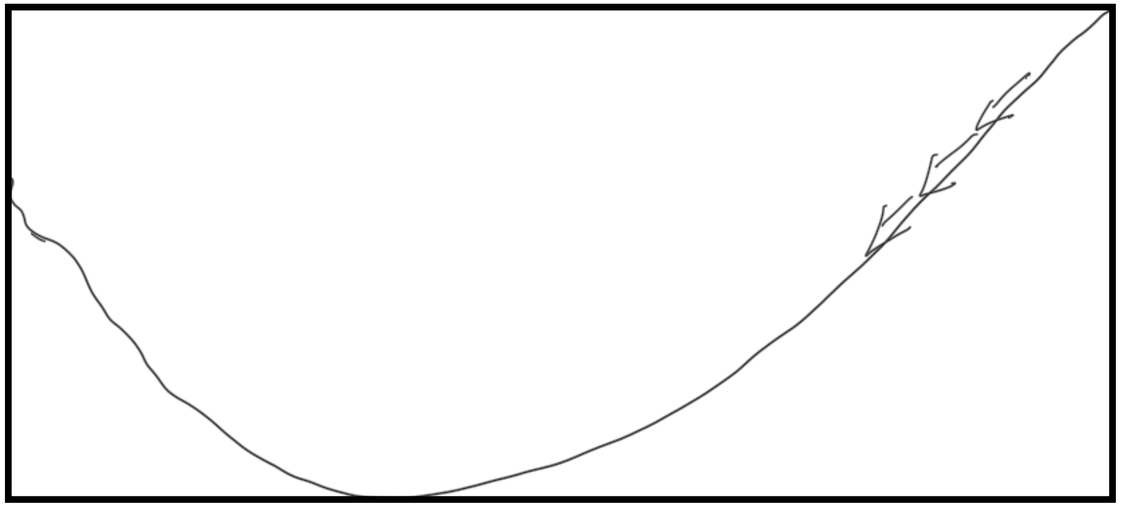
CS 584: Data Mining

Prof. Sanmay Das (Adapted from notes of Prof. Malik Magdon-Ismail)

Minimizing Cross-Entropy Error

- Remember, the cross-entropy error: $E_{\text{in}}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ln \frac{1}{\sigma(y_i \mathbf{w} \,.\, \mathbf{x}_i)} = \frac{1}{n}\sum_{i=1}^{n} \ln(1 + e^{-y_i \mathbf{w} \cdot \mathbf{x}_i})$
- Condition: Set $\nabla E_{\text{in}}(\mathbf{w}) = 0$
- Gradient descent: General technique for minimizing a twice-differentiable function
- In general: Can get stuck in local minima. But $E_{\text{in}}(\mathbf{w})$ is convex in this case, so there is a global minimum





Gradient Descent Idea

- Take small steps in the direction that minimizes the error (in $\mathbf{w}$ space)
- A Taylor expansion argument will tell you that this is the direction of the negative gradient at the $\mathbf{w}$ you are at
- Basic algorithm:
  - Initialize $\mathbf{w}(0)$ at $t = 1$
  - For t = 1:max_iterations
    - Compute $g_t = \nabla E_{\text{in}}(\mathbf{w}(t))$
    - Set $\mathbf{w}(t+1) = \mathbf{w}(t) - \eta g_t$
  - Return $\mathbf{w}(t+1)$
- Behavior dependent on η. In practice, good at getting close to the min quite quickly, then slow

Computing the Gradient for Logistic Regression

- $E_{\text{in}}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ln(1 + e^{-y_i \mathbf{w} \,.\, \mathbf{x}_i})$
- Chain rule: $\nabla_{\mathbf{w}} f(g(\mathbf{w})) = f'(g(w))\, \nabla_{\mathbf{w}} g(\mathbf{w})$
- Using this:

$$\begin{aligned}
\nabla_{\mathbf{w}} E_{\text{in}}(\mathbf{w}) &= \frac{1}{n}\sum_{i=1}^{n} \frac{1}{1 + e^{-y_i \mathbf{w}.\mathbf{x}_i}} \nabla_{\mathbf{w}}(1 + e^{-y_i \mathbf{w}.\mathbf{x}_i}) \\
&= \frac{1}{n}\sum_{i=1}^{n} \frac{e^{-y_i \mathbf{w}.\mathbf{x}_i}}{1 + e^{-y_i \mathbf{w}.\mathbf{x}_i}} \nabla_{\mathbf{w}}(-y_i \mathbf{w} \,.\, \mathbf{x}_i) \\
&= \frac{1}{n}\sum_{i=1}^{n} \frac{1}{1 + e^{y_i \mathbf{w}.\mathbf{x}_i}} (-y_i \mathbf{x}_i) \\
&= -\frac{1}{n}\sum_{i=1}^{n} \frac{y_i \mathbf{x}_i}{1 + e^{y_i \mathbf{w}.\mathbf{x}_i}}
\end{aligned}$$

- Think about these updates:
- If a point is classified correctly
  - $y_i \mathbf{w} \,.\, \mathbf{x}_i > 0$. Denominator is higher, so this point contributes less to the gradient
- If a point is classified incorrectly
  - $y_i \mathbf{w} \,.\, \mathbf{x}_i < 0$. Contribution is higher, and gets closer to $-y_i \mathbf{x}_i$ as this number gets smaller

Gradient Descent Practicalities

- Initialization: OK for logistic regression to initialize $\mathbf{w} = 0$ but in general bad starting points can hurt. Typically, small zero-mean random numbers instead.
- Termination: Max iterations, or gradient has become small enough.
- For logistic regression, in smaller problems, methods like Newton-Raphson work well. For large problems, use stochastic gradient descent early on, and then a specialized solver when closer to the min.
  - Pick a training point and just compute the gradient at that point.
    $$\nabla e_i(\mathbf{w}) = \frac{-y_i \mathbf{x}_i}{1 + e^{y_i \mathbf{w}.\mathbf{x}_i}}$$
  - In expectation, the weight update is the same as the batch method
  - Easy to implement and tune
  - Less memory
  - V. Efficient in terms of learning (vs. optimization) performance
- Using for classification: Simple to set probability threshold to 0.5. Can be tuned for cost-sensitive learning.
  - Even though it's trained for a different criterion, classification accuracy tends to be good!