$\LaTeX$ command declarations here.

EECS 545: Machine Learning
Lecture 07: Logistic Regression & Perceptron

- Instructor: **Jacob Abernethy**
- Date: January 27, 2016

Lecture Exposition Credit: Benjamin Bray, Saket Dewangan

Outline

- Concept of Classification
- Perceptron
- Logistic Regression
  - Intuition, Motivation
  - Newton's Method

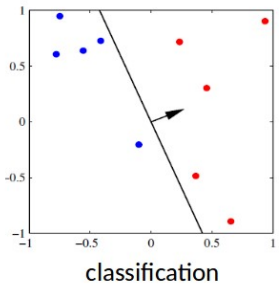
Reading List

- Required:
  - **[MLAPP]**, Chapter 8: Logistic Regression

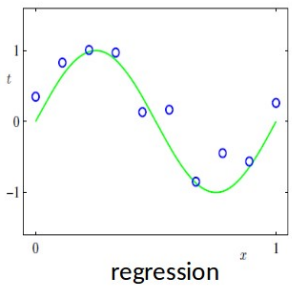
> In this lecture, we will move from regression to classification. Unlike of predicting some value for data in regression, we predict what category data belongs to in classification. And we will introduce two classifiers in this lecture: perceptron and logistic regression. In logistic regression, we will show how to find the optimal coefficients $\mathbf{w}$ using Newton's method.

Review: Supervised Learning

- Goal
  - Given data X in feature space and the labels Y
  - Learn to predict Y from X
- Labels could be discrete or continuous
  - Discrete-valued labels: Classification
  - Continuous-valued labels: Regression



classification



regression

Classification Problem

Classification Problem: Basics

- Given an input vector $\mathbf{x}$, assign it to one of K distinct classes C_k, where $k = 1, \ldots, K$.
- The case $K = 2$ is **Binary Classification**
  - Label $t = 1$ means $x \in C_1$
  - Label $t = 0$ means $x \in C_2$ (or sometimes $t = -1$)
- **Training:** Learn a classifier $y(\mathbf{x})$ from data,

$$\text{Training Data} \quad \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\} \implies \text{Classifier } y(\mathbf{x})$$

- **Prediction:** Predict labels of new data,

$$\text{New Data} \quad \{(\mathbf{x}_1^{new}, t_1^{new}), \ldots, (\mathbf{x}_m^{new}, t_m^{new})\} \overset{h}{\implies} \{y(\mathbf{x}_1^{new}), \ldots, y(x_m^{new})\}$$

- **Performance Evaluation:** Evaluate learned classifier on test data,

$$\text{Test Data} \quad \{(\mathbf{x}_1^{test}, t_1^{test}), \ldots, (\mathbf{x}_m^{test}, t_m^{test})\} \overset{y}{\implies} \{y(\mathbf{x}_1^{test}), \ldots, y(\mathbf{x}_m^{test})\} \implies \text{Error Estimate}$$

  - To estimate **classification error**, we could use e.g. *zero-one loss*:

$$E = \frac{1}{m} \sum_{j=1}^{m} 1[y(\mathbf{x}_j^{test}) \neq t_j^{test})]$$

    i.e. number of misclassified data.

Classification Problems: Strategies

- **Nearest-Neighbors:** Given query data $\mathbf{x}$, find closest training points and do a majority vote.
- **Discriminant Functions:** Learn a function $y(\mathbf{x})$ mapping $\mathbf{x}$ to some class C_k.
- **Probabilistic Model:** Learn the distributions $P(C_k|\mathbf{x})$
  - *Discriminative Models* directly model $P(C_k|\mathbf{x})$ and learn parameters from the training set.
  - *Generative Models* learn class-conditional densities $P(\mathbf{x}|C_k)$ and priors $P(C_k)$

Perceptron Algorithm

- The Perceptron is the most basic model of a training a linear predictor with sequential update steps:
- Given data $\{\mathbf{x}_n, t_n\}_{n=1}^N$, $t_n \in \{-1, 1\}$, here is how perceptron works:
  - **Initialize**: Set $\mathbf{w}_1 = \mathbf{0}$;
  - **For:** $n = 1, 2, \ldots, N$
    - Observe $\mathbf{x}_n$, predict $y_n = \text{sign}(\mathbf{w}_t^\top \phi(\mathbf{x}_n))$
    - Receive $t_n \in \{-1, 1\}$, update:

$$\mathbf{w}_{n+1} = \begin{cases} \mathbf{w}_n & \text{if } t_n \mathbf{w}_n^T \phi(\mathbf{x}_n) > 0 \quad \text{Correct Prediction} \\ \mathbf{w}_n + t_n \phi(\mathbf{x}_n) & \text{otherwise} \quad \text{Incorrect Prediction} \end{cases}$$

  - **End**
- Note that we could repeat the for-loop multiple loops until classification error is less than certain threshold.

Perceptron: Intuition

- We have update

$$\mathbf{w}_{n+1} = \begin{cases} \mathbf{w}_n & \text{if } t_n \mathbf{w}_n^T \phi(\mathbf{x}_n) > 0 \quad \text{Correct Prediction} \\ \mathbf{w}_n + t_n \phi(\mathbf{x}_n) & \text{otherwise} \quad \text{Incorrect Prediction} \end{cases}$$

- The more *positive* $t_n \mathbf{w}_n^T \phi(\mathbf{x}_n)$ is, the more robust performance $\mathbf{w}_n$ has on data $\mathbf{x}_n$
- **Intuition**
  - When $\mathbf{w}_n$ gives incorrect prediction for $\mathbf{x}_n$, i.e. $t_n \mathbf{w}_n^T \phi(\mathbf{x}_n) \leq 0$, above update tells us

$$\begin{aligned} t_n \mathbf{w}_{n+1}^T \phi(\mathbf{x}_n) &= t_n \mathbf{w}_n^T \phi(\mathbf{x}_n) + t_n^2 \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_n) \\ &= t_n \mathbf{w}_n^T \phi(\mathbf{x}_n) + \underbrace{t_n^2 \|\phi(\mathbf{x}_n)\|^2}_{\text{Non-negative}} \end{aligned}$$

  - **Non-negative** term $t_n^2 \|\phi(\mathbf{x}_n)\|^2$ makes $t_n \mathbf{w}_{n+1}^T \phi(\mathbf{x}_n)$ more likely to be positive.
  - Therefore, $\mathbf{w}_{n+1}$ is likely to have better performance on $\mathbf{x}_n$

Perceptron: Essentially a form of Stochastic Gradient Descent

- Define error function:

$$E(\mathbf{w}) = \sum_{n=1}^{N} \max(0, -t_n \mathbf{w}^T \phi(\mathbf{x}_n))$$

- The derivative of $E(\mathbf{w})$ on data $\mathbf{x}_n$ is

$$\nabla_{\mathbf{w}} E(\mathbf{w}|\mathbf{x}_n) = \begin{cases} 0 & \text{if } t_n \mathbf{w}^T \phi(\mathbf{x}_n) > 0 \\ -t_n \phi(\mathbf{x}_n) & \text{otherwise} \end{cases}$$

- Perceptron is equivalent to the following stochastic gradient descent
  - **For**: $n = 1, 2, \ldots, N$
    - $\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} - \eta \nabla_{\mathbf{w}} E(\mathbf{w}_{\text{old}}|\mathbf{x}_n)$
  - **End**
- Notice the "step size" is $\eta = 1$! This is atypical.
- Perceptron was (originally) viewed as building block of the *neural network* (NN). Indeed, NN often called the Multi-Layer Perceptron (MLP).

Perceptron: A magical property

- If problem is *linearly separable*, i.e. a hyperplane separates positives/negatives, then Perceptron *will find a separating* $\mathbf{w}^*$.
- **Theorem**:
  - Assume that $\|\phi(\mathbf{x}_n)\| \leq 1$ for all n
  - Assume $\exists \mathbf{w}$, with $\|\mathbf{w}\|_2 = 1$, such that for all $(\mathbf{x}_n, t_n)$ that $t_n \mathbf{w}^T \phi(\mathbf{x}_n) > \gamma$ for some $\gamma > 0$.
  - Then the Perceptron algorithm will find some $\mathbf{w}^*$ which perfectly classifies all examples
  - The number of updates/mistakes in learning is bounded by $\frac{1}{\gamma^2}$
- This is a *margin bound*, notice that it depends on γ not the dimension of $\phi(\mathbf{x})$
- Proof is in the notes

Remark

- **Proof Sketch**
- Let $\mathbf{w}_*$ be perfect classifier scaled by $\frac{1}{\gamma}$.

$$\begin{aligned}
\frac{1}{\gamma^2} = \|\mathbf{w}_*\|_2^2 &\geq \|\mathbf{w}_* - \mathbf{0}\|_2^2 - \|\mathbf{w}_* - \mathbf{w}_{T+1}\|_2^2 \\
&= \sum_{n=1}^{T} \|\mathbf{w}_* - \mathbf{w}_n\|_2^2 - \|\mathbf{w}_* - \mathbf{w}_{n+1}\|_2^2 \\
&= \sum\nolimits_{n\,:\,t_n \mathbf{w}_n^T \phi(\mathbf{x}_n) < 0} \|\mathbf{w}_* - \mathbf{w}_n\|_2^2 - \|\mathbf{w}_* - (\mathbf{w}_n + t_n \phi(\mathbf{x}_n)\|_2^2 \\
&= \sum\nolimits_{n\,:\,t_n \mathbf{w}_n^T \phi(\mathbf{x}_n) < 0} 2 \left(\underbrace{t_n(\mathbf{w}_*^T \phi(\mathbf{x}_n))}_{\geq 1} - \underbrace{t_n(\mathbf{w}_n^T \phi(\mathbf{x}_n))}_{\geq 0} \right) \underbrace{-t_n^2 \|\phi(\mathbf{x}_n)\|_2^2}_{\geq -1} \\
&\geq \sum\nolimits_{n\,:\,t_n \mathbf{w}_n^T \phi(\mathbf{x}_n) < 0} 1 \quad = \quad \#\text{mistakes}[\text{Perceptron}]
\end{aligned}$$

- See learning theory lecture notes (http://web.eecs.umich.edu/~jabernet/eecs598course/fall2015/web/notes/lec16_110515.pdf) for full details. (Note that we have changed notations a little bit. Index n, label t_n and data feature vector $\phi(\mathbf{x})$ each corresponds to index t, label y_n and data $\mathbf{x}$ in this reference)

Logistic Regression

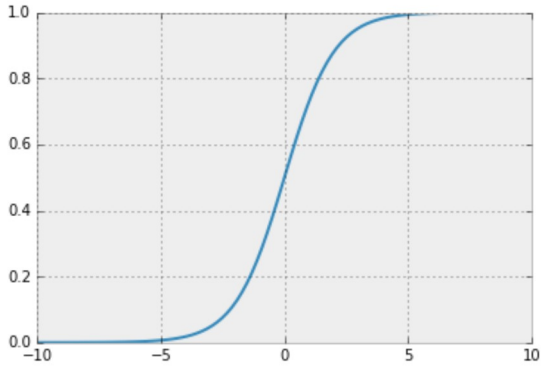
- Logistic Regression is a technique for **classification**!
- We will focus on *binary* classification

Logistic Regression: Preliminary—Logistic Sigmoid Function

- The **logistic sigmoid function** is

$$\sigma(a) = \frac{1}{1+\exp(-a)} = \frac{\exp(a)}{1+\exp(a)}$$

- Sigmoid function $\sigma(a)$ maps $(-\infty, +\infty) \to (0, 1)$



Logistic Regression: Why use Logistic Sigmoid Function?

- Prediction is picking the larger one of $P(y=1|\mathbf{x})$ and $P(y=0|\mathbf{x})$.
- This can be implemented by evaluating **log odds**

$$a = \ln \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})}$$

- So the prediction is

$$y = \begin{cases} 1 & a \geq 0 \\ 0 & a < 0 \end{cases}$$

- Since $P(y=1|\mathbf{x}) + P(y=0|\mathbf{x}) = 1$, we could solve for

$$P(y=1|\mathbf{x}) = \frac{\exp(a)}{1+\exp(a)} = \sigma(a)$$

  Logistic Function appears!

- A heuristic choice for log odds is a separating **hyper plane** $a = \mathbf{w}^T\phi(\mathbf{x})$. So the criterion becomes

$$\boxed{y = \begin{cases} 1 & \mathbf{w}^T\phi(\mathbf{x}) \geq 0, \quad \text{i.e.} \quad \sigma(\mathbf{w}^T\phi(\mathbf{x}) \geq 0) \geq 0.5 \\ 0 & \mathbf{w}^T\phi(\mathbf{x}) < 0, \quad \text{i.e.} \quad \sigma(\mathbf{w}^T\phi(\mathbf{x}) \geq 0) < 0.5 \end{cases}}$$

- In this case, $P(y=1|\mathbf{x}) = \sigma(\mathbf{w}^T\phi(\mathbf{x}))$ and $P(y=0|\mathbf{x}) = 1 - \sigma(\mathbf{w}^T\phi(\mathbf{x}))$.

Logistic Regression: Underlying Model

- We already have

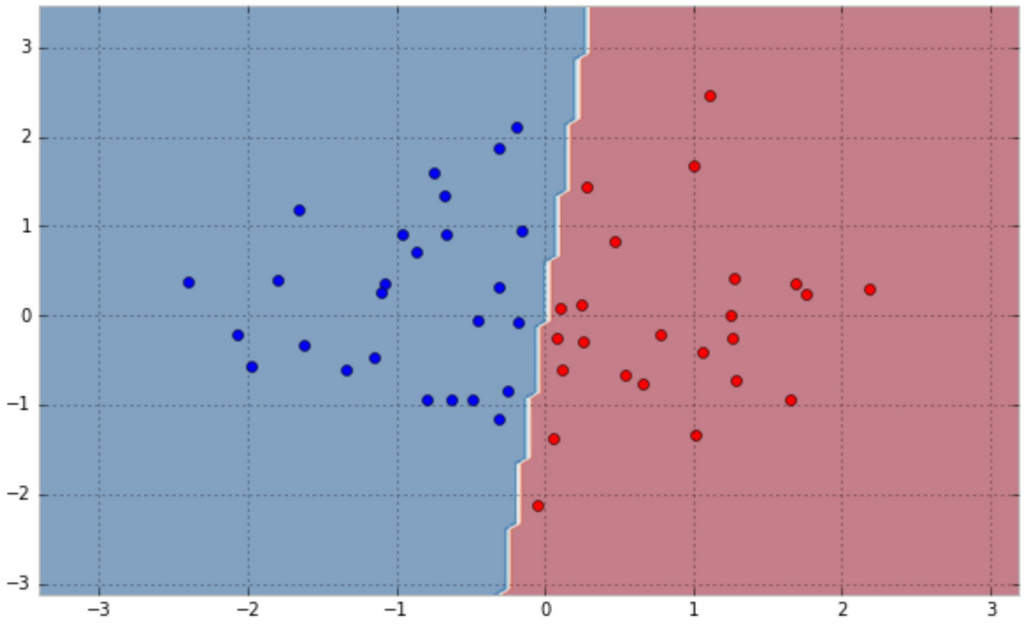
$$P(y=1|\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^T\phi(\mathbf{x}))$$
$$P(y=0|\mathbf{x}, \mathbf{w}) = 1 - \sigma(\mathbf{w}^T\phi(\mathbf{x}))$$

- So we could model **class posterior** using Bernoulli random variable

$$y|\mathbf{x}, \mathbf{w} \sim \text{Bernoulli}(\sigma(\mathbf{w}^T\phi(\mathbf{x})))$$

- We can obtain the best parameter $\mathbf{w}$ by maximizing the likelihood of the training data.(Later)
- Logistic regression is simplest discriminative model that is **linear** in the parameters.

Logistic Regression: Example



- We could clearly see the linear boundary corresponding to $\mathbf{w}^T\phi(\mathbf{x})$

Logistic Regression: Likelihood

- We saw before that the **likelihood** for each binary label is:

$$P(y=1|\mathbf{x},\mathbf{w}) = \sigma(\mathbf{w}^T\phi(\mathbf{x}))$$
$$P(y=0|\mathbf{x},\mathbf{w}) = 1-\sigma(\mathbf{w}^T\phi(\mathbf{x}))$$

- With a clever trick, this

$$P(y|x,w) = \sigma(\mathbf{w}^T\phi(\mathbf{x}))^y \cdot (1-\sigma(\mathbf{w}^T\phi(\mathbf{x})))^{1-y}$$

- For a data set $\{(\mathbf{x}_n, t_n)\}_{n=1}^N$ where $t_n \in \{0,1\}$, the **likelihood function** is

$$P(\mathbf{y}=\mathbf{t}|\mathcal{X},\mathbf{w}) = \prod_{n=1}^N P(y=t_n|\mathbf{x}_n,\mathbf{w}) = \prod_{n=1}^N \sigma(\mathbf{w}^T\phi(\mathbf{x}_n))^{t_n}[1-\sigma(\mathbf{w}^T\phi(\mathbf{x}_n))]^{1-t_n}$$

  - where $\mathcal{X} = \{\mathbf{x}_n\}_{n=1}^N$

- The optimal $\mathbf{w}$ can be obtained by maximizing this likelihood.
- Maximum likelihood estimate $\mathbf{w}_{ML}$ makse sense because $\mathbf{w}_{ML}$ is the coefficient that are most likely to produce $\{t_n\}_{n=1}^N$ given $\mathcal{X}$.
- Define **negative log-likelihood** as the **loss**

$$E(\mathbf{w}) \triangleq -\ln P(\mathbf{y}=\mathbf{t}|\mathcal{X},\mathbf{w})$$

- Maximizing **likelihood** is equivalent to minimizing **loss** $E(\mathbf{w})$

Logistic Regression: Gradient of Loss

- Loss function $E(\mathbf{w})$ can be transformed:

$$\begin{aligned}
E(\mathbf{w}) &= -\ln P(\mathbf{y}=\mathbf{t}|\mathcal{X},\mathbf{w}) \\
&= -\ln \textstyle\prod_{n=1}^N \sigma(\mathbf{w}^T\phi(\mathbf{x}_n))^{t_n}[1-\sigma(\mathbf{w}^T\phi(\mathbf{x}_n))]^{1-t_n} \\
&= -\textstyle\sum_{n=1}^N \left[t_n \ln \sigma(\mathbf{w}^T\phi(\mathbf{x}_n)) + (1-t_n)\ln(1-\sigma(\mathbf{w}^T\phi(\mathbf{x}_n)))\right] \\
&= -\sum_{n=1}^N \left[t_n \ln \frac{\exp(\mathbf{w}^T\phi(\mathbf{x}_n))}{1+\exp(\mathbf{w}^T\phi(\mathbf{x}_n))} + (1-t_n)\ln(\frac{1}{1+\exp(\mathbf{w}^T\phi(\mathbf{x}_n))})\right] \\
&= -\sum_{n=1}^N \left[t_n \ln \frac{1}{1+\exp(-\mathbf{w}^T\phi(\mathbf{x}_n))} + (1-t_n)\ln(\frac{1}{1+\exp(\mathbf{w}^T\phi(\mathbf{x}_n))})\right] \\
&= \boxed{\textstyle\sum_{n=1}^N \left[t_n \ln(1+\exp(-\mathbf{w}^T\phi(\mathbf{x}_n))) + (1-t_n)\ln(1+\exp(\mathbf{w}^T\phi(\mathbf{x}_n)))\right]}
\end{aligned}$$

- Gradient of loss $\nabla_{\mathbf{w}} E(\mathbf{w})$

$$\begin{aligned}\nabla_{\mathbf{w}} E(\mathbf{w}) &= \nabla_{\mathbf{w}} \sum_{n=1}^{N} \left[t_n \ln(1+\exp(-\mathbf{w}^T\phi(\mathbf{x}_n))) + (1-t_n)\ln(1+\exp(\mathbf{w}^T\phi(\mathbf{x}_n)))\right] \\ &= \sum_{n=1}^{N} \left[-t_n \frac{\exp(-\mathbf{w}^T\phi(\mathbf{x}_n))}{1+\exp(-\mathbf{w}^T\phi(\mathbf{x}_n))}\phi(\mathbf{x}_n) + (1-t_n)\frac{\exp(\mathbf{w}^T\phi(\mathbf{x}_n))}{1+\exp(\mathbf{w}^T\phi(\mathbf{x}_n))}\phi(\mathbf{x}_n)\right] \\ &= \sum_{n=1}^{N} \left[-t_n(1-\sigma(\mathbf{w}^T\phi(\mathbf{x}_n))) + (1-t_n)\sigma(\mathbf{w}^T\phi(\mathbf{x}_n))\right]\phi(\mathbf{x}_n) \\ &= \sum_{n=1}^{N} \left[\sigma(\mathbf{w}^T\phi(\mathbf{x}_n)) - t_n\right]\phi(\mathbf{x}_n) \\ &= \boxed{\Phi^T(\sigma(\Phi\mathbf{w}) - \mathbf{t})}\end{aligned}$$

  of which

$$\Phi = \begin{bmatrix} - & \phi(\mathbf{x}_1)^T & - \\ & \vdots & \\ - & \phi(\mathbf{x}_N)^T & - \end{bmatrix}_{N\times M} \qquad \sigma(\Phi\mathbf{w}) = \begin{bmatrix} \sigma(\mathbf{w}^T\phi(\mathbf{x}_1)) \\ \vdots \\ \sigma(\mathbf{w}^T\phi(\mathbf{x}_N)) \end{bmatrix}_{N\times 1} \qquad \mathbf{t} = \begin{bmatrix} t_1 \\ \vdots \\ t_N \end{bmatrix}_{N\times 1}$$

- With the gradient of loss, we could perform *gradient descent* to find $\mathbf{w}_{ML}$.
- But we will use a new method by finding roots of first order derivative!

> Remark
>
> - Note that this gradient resembles the gradient in linear regression with least squares (Check Lecture 4)
>
> $$\text{Logistic Regression} \quad \nabla_{\mathbf{w}} E(\mathbf{w}) = \Phi^T(\sigma(\Phi\mathbf{w}) - \mathbf{t})$$
> $$\text{Linear Regression} \quad \nabla_{\mathbf{w}} E(\mathbf{w}) = \Phi^T(\Phi\mathbf{w} - \mathbf{t})$$

Newton's Method: Overview

- First let's consider one dimension case.
- **Goal:** Finding *root* of a general function $f(x)$, i.e. solve for x such that

$$f(x) = 0$$

- **Newton's Method:** Repeat until convergence:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

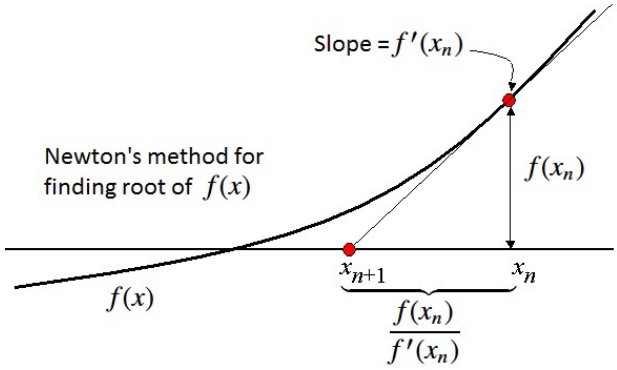
Newton's Method: Geometric Intuition

- Find the roots of $f(x)$ by following its **tangent lines**. The tangent line of $f(x)$ at x_n has equation

$$\ell(x) = f(x_n) + (x - x_n)f'(x_n)$$

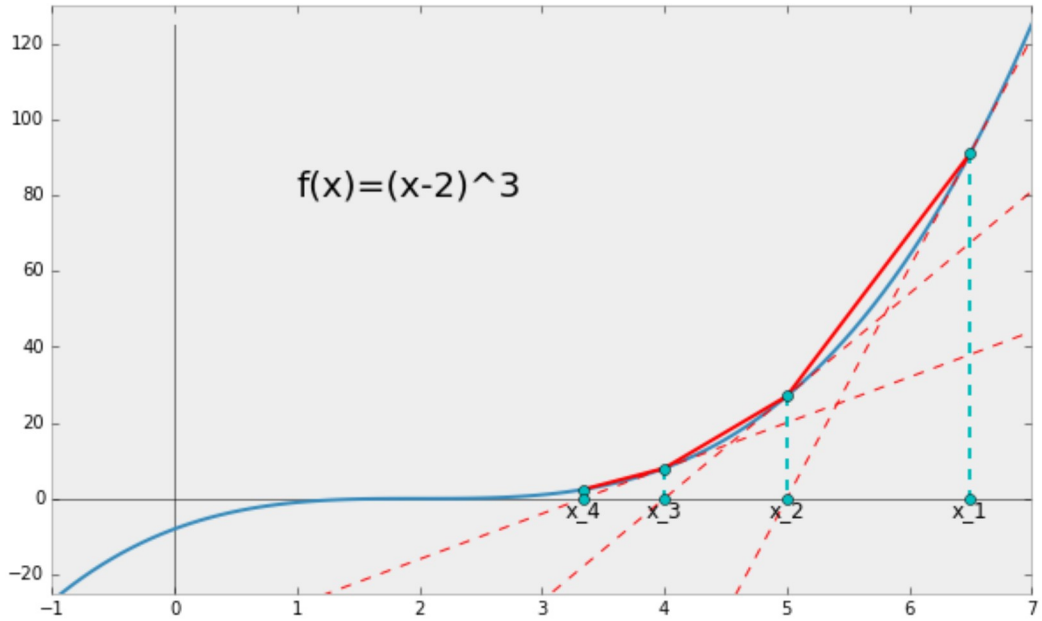
- Set next iterate x_{n+1} to be **root** of tangent line:

$$f(x_n) + (x - x_n)f'(x_n) = 0$$
$$\implies \boxed{x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}}$$



Newton's Method: Example

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_10
Iteration Process	6.500	5.000	4.000	3.333	2.889	2.593	2.395	2.263	2.176	2.117



Remark

- Here is how Newton's method works
  - Given some initial point x_1, we first find the tangent line $\ell_1(x)$ of $f(x)$ at x_1.
  - Let x_2 denote the root of $\ell_1(x)$, i.e. $\ell_1(x_2) = 0$
  - Find the tangent line $\ell_2(x)$ of $f(x)$ at x_2.
  - Let x_3 denote the root of $\ell_2(x)$, i.e. $\ell_1(x_3) = 0$

Newton's Method: Finding Stationary Point

- We have shown how to use Newton's method to find the root for $f(x)$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

- **Note** that Stationary point of $f(x)$ is equivalent to root of $f'(x)$
- So, we could find stationary point of $f(x)$ by finding root of $f'(x)$ using Newton's method.
- The iteration steps becomes

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

- For *multi-dimension* case, this iteration turns into

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \left(\nabla^2 f(\mathbf{x}_n)\right)^{-1} \nabla_{\mathbf{x}} f(\mathbf{x}_n)$$

of which $\nabla^2 f(\mathbf{x}_n)$ is **Hessian matrix** which is the *second order derivative*

$$\nabla^2 f = \begin{bmatrix} \frac{\partial f}{\partial x_1 \partial x_1} & \cdots & \frac{\partial f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial x_n \partial x_1} & \cdots & \frac{\partial f}{\partial x_n \partial x_n} \end{bmatrix}$$

Logistic Regression: Applying Newton's Method

- Back to logistic regression!
- Recall our goal to minimize $E(\mathbf{w})$ and we already have its gradient

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \sum_{n=1}^{N} \left[\sigma(\mathbf{w}^T \phi(\mathbf{x}_n)) - t_n\right] \phi(\mathbf{x}_n) = \Phi^T \left(\sigma(\Phi \mathbf{w}) - \mathbf{t}\right)$$

- To minimize of $E(\mathbf{w})$, we could use Newton's method to find its *stationary point*!
- To use Newton's method, we need the *Hessian matrix*.

Logistic Regression: Hessian Matrix

$$\begin{aligned}
\nabla^2_{\mathbf{w}} E(\mathbf{w}) &= \nabla_{\mathbf{w}} \nabla_{\mathbf{w}} E(\mathbf{w}) \\
&= \nabla_{\mathbf{w}} \sum_{n=1}^{N} \left[\sigma(\mathbf{w}^T \phi(\mathbf{x}_n)) - t_n\right] \phi(\mathbf{x}_n) \\
&= \sum_{n=1}^{N} \nabla_{\mathbf{w}} \sigma(\mathbf{w}^T \phi(\mathbf{x}_n)) \phi(\mathbf{x}_n) \\
&= \sum_{n=1}^{N} \nabla_{\mathbf{w}} \frac{1}{1+\exp(-\mathbf{w}^T \phi(\mathbf{x}_n))} \phi(\mathbf{x}_n) \\
&= \sum_{n=1}^{N} \phi(\mathbf{x}_n) \frac{\exp(-\mathbf{w}^T \phi(\mathbf{x}_n))}{(1+\exp(-\mathbf{w}^T \phi(\mathbf{x}_n)))^2} \phi(\mathbf{x}_n)^T \\
&= \sum_{n=1}^{N} \phi(\mathbf{x}_n) \frac{1}{1+\exp(-\mathbf{w}^T \phi(\mathbf{x}_n))} \frac{\exp(-\mathbf{w}^T \phi(\mathbf{x}_n))}{1+\exp(-\mathbf{w}^T \phi(\mathbf{x}_n))} \phi(\mathbf{x}_n)^T \\
&= \sum_{n=1}^{N} \phi(\mathbf{x}_n) [\sigma(\mathbf{w}^T \phi(\mathbf{x}_n)) \cdot (1 - \sigma(\mathbf{w}^T \phi(\mathbf{x}_n)))] \phi(\mathbf{x}_n)^T \\
&= \sum_{n=1}^{N} \phi(\mathbf{x}_n) r_n(\mathbf{w}) \phi(\mathbf{x}_n)^T
\end{aligned}$$

- of which $r_n(\mathbf{w}) = \sigma(\mathbf{w}^T \phi(\mathbf{x}_n)) \cdot (1 - \sigma(\mathbf{w}^T \phi(\mathbf{x}_n)))$

$$\begin{aligned}
\nabla^2_{\mathbf{w}} E(\mathbf{w}) &= \sum_{n=1}^{N} \phi(\mathbf{x}_n) r_n(\mathbf{w}) \phi(\mathbf{x}_n)^T \\
&= \begin{bmatrix} | & & | \\ \phi(\mathbf{x}_1) & \cdots & \phi(\mathbf{x}_N) \\ | & & | \end{bmatrix} \begin{bmatrix} r_1(\mathbf{w}) & & \\ & \ddots & \\ & & r_N(\mathbf{w}) \end{bmatrix} \begin{bmatrix} - & \phi(\mathbf{x}_1)^T & - \\ & \vdots & \\ - & \phi(\mathbf{x}_N)^T & - \end{bmatrix} \\
&= \boxed{\Phi^T R(\mathbf{w}) \Phi}
\end{aligned}$$

- of which

$$R(\mathbf{w}) = \begin{bmatrix} r_1(\mathbf{w}) & & & \\ & r_2(\mathbf{w}) & & \\ & & \ddots & \\ & & & r_N(\mathbf{w}) \end{bmatrix}$$

Logistic Regression: Applying Newton's Method

- We already have

$$\begin{aligned}
\nabla_{\mathbf{w}} E(\mathbf{w}) &= \Phi^T \left(\sigma(\Phi \mathbf{w}) - \mathbf{t}\right) \\
\nabla^2_{\mathbf{w}} E(\mathbf{w}) &= \Phi^T R(\mathbf{w}) \Phi
\end{aligned}$$

- So the iteration step is

$$\begin{aligned}
\mathbf{w}_{n+1} &= \mathbf{w}_n - \left(\nabla^2_{\mathbf{w}} E(\mathbf{w}_n)\right)^{-1} \nabla_{\mathbf{w}} f(\mathbf{w}_n) \\
&= \boxed{\mathbf{w}_n - \left(\Phi^T R(\mathbf{w}_n) \Phi\right)^{-1} \Phi^T \left(\sigma(\Phi \mathbf{w}_n) - \mathbf{t}\right)}
\end{aligned}$$

- Repeat until convergence and we could get maximum likelihood estimate $\mathbf{w}_{ML}$ which minimizes the loss function $E(\mathbf{w})$ and maximizes likelihood function $P(\mathbf{y} = \mathbf{t} | \mathcal{X}, \mathbf{w})$

Logistic Regression: Do we have closed-form solution?

- Recall for **ordinary least squares** and **regularized least squares**, we have closed-form solution:

	Ordinary Least Squares	Regularized Least Squares
Derivate of Loss Function	$\Phi^T\Phi\mathbf{w} - \Phi^T\mathbf{t}$	$(\Phi^T\Phi + \lambda I)\mathbf{w} - \Phi^T\mathbf{t}$
Closed-form Solution	$(\Phi^T\Phi)^{-1}\Phi^T\mathbf{t}$	$(\Phi^T\Phi + \lambda I)^{-1}\Phi^T\mathbf{t}$

- They are obtained by finding the closed-form root of derivative of loss function。
- For logistic regression, we have

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = 0$$
$$\Downarrow$$
$$\Phi^T\left(\sigma(\Phi\mathbf{w}) - \mathbf{t}\right) = 0$$

- Existence of sigmoid function makes $\nabla_{\mathbf{w}} E(\mathbf{w})$ **nonlinear** and no closed-form solution exists.
- So we must **iterate**!

Appendix: Multi-class Classification using Logistic Regression

- We have seen sigmoid function enables us to do binary classification with logistic regression.
- What if we want have multiple classes?
- We will resort to **softmax** aka **normalized exponential** function
- **Softmax Function**

$$p_k = \frac{\exp(q_k)}{\sum_j \exp(q_j)}$$

  Given any real numbers $q_1, \ldots, q_n$, we can generate a distribution on them using softmax function.
- Recall in binary case, we have

$$P(y = 1|\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^T\phi(\mathbf{x}))$$

- For K-class classification, we define $\mathcal{W} = \mathbf{w}_{k=1}^K$.
- The probability data $\mathbf{x}$ belongs to class j is

$$P(y = j|\mathbf{x}, \mathcal{W}) = \frac{\exp(\mathbf{w}_j^T\phi(\mathbf{w}))}{\sum_{k=1}^K \exp(\mathbf{w}_k^T\phi(\mathbf{w}))}$$

- We classify using

$$y = \underset{j\in\{1,\ldots,K\}}{\arg\max} P(y = j|\mathbf{x}, \mathcal{W})$$

- Similarly, $\mathcal{W} = \mathbf{w}_{k=1}^K$ is learned by maximizing likelihood function.
- For details, please refer to this (http://ufldl.stanford.edu/wiki/index.php/Softmax_Regression)