CS229 lecture 1 notes

James Chuang

January 5, 2017

Contents

My notes on Andrew Ng's CS229 lecture 1 notes.

Supervised learning

Notation:

- $x^{(i)} \rightarrow$ input variables, aka ***features***.
- $y^{(i)} \rightarrow$ output variables, aka the ***target***.
- $\left(x^{(i)}, y^{(i)}\right) \rightarrow$ ***training example***.
- $\left\{\left(x^{(i)}, y^{(i)}\right); i = 1, \dots, m\right\} \rightarrow$ ***training set***, a list of m training examples
- $\mathcal{X} \rightarrow$ the space of input values
- $\mathcal{Y} \rightarrow$ the space of output values

The goal of supervised learning: Given a training set, learn a ***'hypothesis'*** function $h : \mathcal{X} \mapsto \mathcal{Y}$ such that $h(x)$ is a "good" predictor for the corresponding value of y.

- Continuous targets $\rightarrow$ ***regression***.
- Discrete targets $\rightarrow$ ***classification***.

Linear Regression

Linear model: $h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x$, where $x_0 = 1$ to account for an intercept.

Cost function (squared error loss):

$$\begin{aligned} J(\theta) &= \frac{1}{2} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - \left(y^{(i)}\right)\right)^2 \\ &= \frac{1}{2} \left(\mathbf{X}\theta - y\right)^T \left(\mathbf{X}\theta - y\right) \end{aligned}$$

Minimize cost function using ***gradient descent***. The update rule: $\theta_j \leftarrow \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$, i.e. $\theta \leftarrow \theta - \alpha\nabla_\theta J(\theta)$.

Calculate $\frac{\partial}{\partial\theta_j}J(\theta)$:

$$\begin{aligned}\frac{\partial}{\partial\theta_j}J(\theta) &= \frac{\partial}{\partial\theta_j}\frac{1}{2}\left(h_\theta(x)-y\right)^2\\ &= 2\cdot\frac{1}{2}\left(h_\theta(x)-y\right)\cdot\frac{\partial}{\partial\theta_j}\left(h_\theta(x)-y\right)\\ &= \left(h_\theta(x)-y\right)\cdot\frac{\partial}{\partial\theta_j}\left(\theta^T\mathbf{x}-y\right)\\ &= \left(h_\theta(x)-y\right)x_j\end{aligned}$$

Therefore, the update rule is: $\theta \leftarrow \theta + \alpha\left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right)x_j^{(i)}$. This is the ***LMS*** (least mean square) update rule, aka the ***Widrow-Hoff*** learning rule. Note that the magnitude of the update is proportional to the error term.

- ***Batch gradient descent***- use every example in the training set to estimate the gradient
- ***Stochastic gradient descent***- use a subset (minibatch) of the training set to estimate the gradient

The normal equations

In the case of squared error loss, can analytically solve for the minimum instead of resorting to an iterative, numerical algorithm.

Some properties of matrix derivatives

$\nabla_A \text{tr}AB = B^T$: - For $A\in\mathbb{R}^{m\times n}, B\in\mathbb{R}^{n\times m}$

$$\begin{aligned}AB &= \begin{bmatrix} - & a_1^T & - \\ - & a_2^T & - \\ & \vdots & \\ - & a_m^T & - \end{bmatrix}\begin{bmatrix} | & | & & | \\ b_1 & b_2 & \cdots & b_m \\ | & | & & | \end{bmatrix}\\ \text{tr}AB &= \sum_{i=1}^{m} a_i^T b_i\\ \text{tr}AB &= \sum_{i=1}^{m}\sum_{j=1}^{n} A_{i,j}B_{j,i}\\ \nabla_A\text{tr}AB &= \begin{bmatrix} B_{1,1} & B_{2,1} & \cdots & B_{n,1} \\ B_{1,2} & B_{2,2} & \cdots & B_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ B_{1,m} & B_{2,m} & \cdots & B_{n,m}\end{bmatrix}\\ \nabla_A\text{tr}AB &= B^T\end{aligned}$$

$\nabla_{A^T} f(A) = \left(\nabla_A f(A)\right)^T$:

$$\begin{aligned}\nabla_{A^T} f(A) &= \begin{bmatrix} \frac{\partial f}{\partial A_{1,1}} & \frac{\partial f}{\partial A_{2,1}} & \cdots & \frac{\partial f}{\partial A_{m,1}} \\ \frac{\partial f}{\partial A_{1,2}} & \frac{\partial f}{\partial A_{2,2}} & \cdots & \frac{\partial f}{\partial A_{m,2}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{1,n}} & \frac{\partial f}{\partial A_{2,n}} & \cdots & \frac{\partial f}{\partial A_{n,m}} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial f}{\partial A_{1,1}} & \frac{\partial f}{\partial A_{1,2}} & \cdots & \frac{\partial f}{\partial A_{1,n}} \\ \frac{\partial f}{\partial A_{2,1}} & \frac{\partial f}{\partial A_{2,2}} & \cdots & \frac{\partial f}{\partial A_{2,n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{m,1}} & \frac{\partial f}{\partial A_{m,2}} & \cdots & \frac{\partial f}{\partial A_{m,n}} \end{bmatrix}^T \\ &= (\nabla_A f(A))^T\end{aligned}$$

$\nabla_A \text{tr} ABA^T C = CAB + C^T AB^T$: Okay I get that knowing this makes the derivation below easier, but that's not really a reason to learn such a convoluted identity…

Calculate $\nabla_\theta J(\theta)$:

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \left[\frac{1}{2}(\mathbf{X}\theta - \mathbf{y})^T (\mathbf{X}\theta - \mathbf{y})\right] \\ &= \frac{1}{2}\nabla_\theta \left[(\mathbf{X}\theta)^T (\mathbf{X}\theta) - (\mathbf{X}\theta)^T \mathbf{y} - \mathbf{y}^T(\mathbf{X}\theta) + \mathbf{y}^T\mathbf{y}\right] \\ &= \frac{1}{2}\nabla_\theta \left[\theta^T \mathbf{X}^T \mathbf{X}\theta - 2\mathbf{y}^T\mathbf{X}\theta + \mathbf{y}^T\mathbf{y}\right] \\ &= \frac{1}{2}\left[2\mathbf{X}^T\mathbf{X}\theta - 2\mathbf{X}^T\mathbf{y}\right] \\ &= \mathbf{X}^T\mathbf{X}\theta - \mathbf{X}^T\mathbf{y}\end{aligned}$$

To minimize J, set the derivatives to zero to get the ***normal equations***:

$$\begin{aligned}\mathbf{X}^T\mathbf{X}\theta - \mathbf{X}^T\mathbf{y} &= 0 \\ \mathbf{X}^T\mathbf{X}\theta &= \mathbf{X}^T\mathbf{y} \\ \theta &= \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}\end{aligned}$$

Probabilistic interpretation

The least squares cost function is a reasonable choice because it is the ***maximum likelihood estimator***, assuming that the target is linearly related to the inputs, with Gaussian noise arising from unmodeled effects or random noise:

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$$

Assume the $\epsilon^{(i)}$ are ***IID (independent and identically distributed)*** and $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$. Then the density of $\epsilon^{(i)}$ is given by

$$\begin{aligned}p(\epsilon^{(i)}) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-\left(e^{(i)}\right)^2}{2\sigma^2}\right) \\ p(y^{(i)}|x^{(i)};\theta) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}\right)\end{aligned}$$

Likelihood function:

$$\begin{aligned}
L(\theta) &= L(\theta; \mathbf{X}, \mathbf{y}) = p(\mathbf{y} \mid \mathbf{X}; \theta) \\
&= \prod_{i=1}^{m} p\left(y^{(i)} \mid x^{(i)}; \theta\right) && \text{data points are independent} \\
&= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}\right) \\
\ell(\theta) &= \log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}\right) && \text{take log-likelihood} \\
&= \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}\right) \\
&= m \log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^{m} \left(y^{(i)} - \theta^T x^{(i)}\right)^2
\end{aligned}$$

The only term dependent on θ is the rightmost term. Therefore,

$$\underset{\theta}{\text{argmax}}\, \ell(\theta) = \underset{\theta}{\text{argmax}} - \frac{1}{2} \sum_{i=1}^{m} \left(y^{(i)} - \theta^T x^{(i)}\right)^2 = \underset{\theta}{\text{argmin}} \frac{1}{2} \sum_{i=1}^{m} \left(y^{(i)} - \theta^T x^{(i)}\right)^2 = \underset{\theta}{\text{argmin}}\, J(\theta)$$

. I.e., maximizing $\ell(\theta)$ is equivalent to minimizing the least-squares cost function.

Locally weighted linear regression

Simple linear regression algorithm:

- Fit θ to minimize $\sum_i \left(y^{(i)} - \theta^T x^{(i)}\right)^2$.
- Output $\theta^T x$.

Locally weighted linear regression algorithm:

- Fit θ to minimize $\sum_i w^{(i)} \left(y^{(i)} - \theta^T x^{(i)}\right)^2$.
- Output $\theta^T x$.

The $w^{(i)}$ are non-negative valued ***weights***. A standard choice for the weights is

$$w^{(i)} = \exp\left(-\frac{\left(x^{(i)} - x\right)^2}{2\tau^2}\right)$$

, where τ is the ***bandwidth*** parameter controlling how quickly the weight drops off with distance from a training example. Note that if the distance to the training example is small, $w^{(i)} \to e^0 = 1$ and when the distance is large, $w^{(i)} \to e^{-\infty} \approx 0$. Locally weighted linear regression is a ***non-parametric*** algorithm, because it needs the entire training set in order to calculate distances for weights when predicting values for new points. In contrast, unweighted linear regression is ***parametric***, since once the parameters θ are learned from the training set, the training set is no longer needed for prediction.

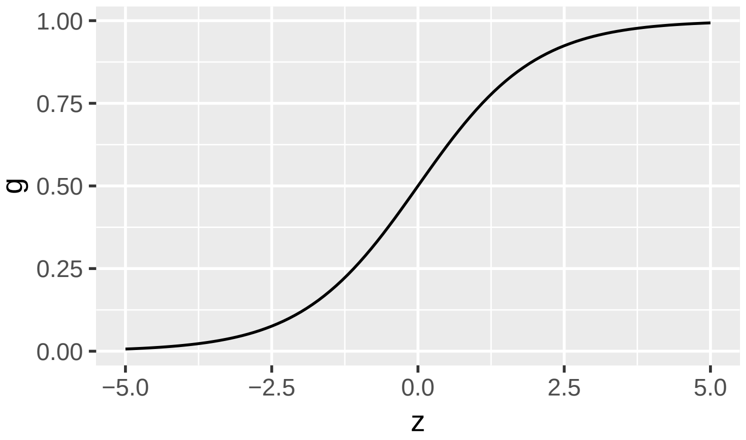
Classification and logistic regression

Start with ***binary classification***, where y can take on two values, 0 or 1, aka the ***negative class (-)*** and the ***positive class (+)***. The $y^{(i)}$ corresponding to a training example $x^{(i)}$ is also called its ***label***.

Logistic Regression

Classification by linear regression runs into the problem that $h_\theta(x)$ can return values outside of $[0,1]$, which cannot be interpreted as valid probabilities. This can be addressed by squashing the $h_\theta(x)$ into the range $(0,1)$ using the ***logistic*** aka ***sigmoid*** function $g(z)$ such that:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}, \quad \text{where } g(z) = \frac{1}{1+e^{-z}}$$



As $z \to \infty, g \to 1$. As $z \to -\infty, g \to 0$.

The derivative of the logistic sigmoid:

$$\begin{aligned} g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\ &= \frac{-1}{(1+e^{-z})^2}\left(-e^{-z}\right) \\ &= \frac{1}{1+e^{-z}}\left(1 - \frac{1}{(1+e^{-z})}\right) \\ &= g(z)\left(1-g(z)\right) \end{aligned}$$

Fitting for θ by maximum likelihood: Assume that:

$$\begin{aligned} P(y=1 \mid x;\theta) &= h_\theta(x) \\ P(y=0 \mid x;\theta) &= 1-h_\theta(x) \end{aligned}$$

i.e.,

$$p(y \mid x;\theta) = (h_\theta(x))^y\,(1-h_\theta(x))^{1-y}$$

Likelihood function, assuming m independent training examples:

$$\begin{aligned} L(\theta) &= p(\mathbf{y} \mid \mathbf{X};\theta) \\ &= \prod_{i=1}^m p\left(y^{(i)} \mid x^{(i)};\theta\right) \\ &= \prod_{i=1}^m \left(h_\theta\left(x^{(i)}\right)\right)^{y^{(i)}}\left(1-h_\theta\left(x^{(i)}\right)\right)^{1-y^{(i)}} \\ \ell(\theta) &= \sum_{i=1}^m y^{(i)}\log h_\theta\left(x^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right) \end{aligned}$$

Maximize the log-likehood $\ell(\theta)$ by gradient ascent with the update rule $\theta \leftarrow \theta + \alpha \nabla_\theta \ell(\theta)$. For one training example:

$$\begin{aligned}
\frac{d}{d\theta_j}\ell(\theta) &= \frac{d}{d\theta_j}\left[y \log h_\theta(x) + (1-y)\log(1-h(x))\right] \\
&= y\frac{1}{g(\theta^T x)}\frac{dg}{d\theta_j} - (1-y)\frac{1}{1-g(\theta^T x)}\frac{dg}{d\theta_j} \\
&= \left(y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)}\right)\frac{d}{d\theta_j}g(\theta^T x) \\
&= \left(y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)}\right) g(\theta^T x)\left(1-g(\theta^T x)\right)\frac{d}{d_{\theta_j}}\theta^T x \\
&= \left(y\left(1-g(\theta^T x)\right) - (1-y)\, g(\theta^T x)\right) x_j \\
&= \left(y - h_\theta(x)\right) x_j
\end{aligned}$$

Therefore, the stochastic gradient ascent rule is:

$$\theta_j \leftarrow \theta_j + \alpha\left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$$

This looks identical to the LMS update rule, but is not the same because $h_\theta(x^{(i)})$ is a non-linear function of $\theta^T x^{(i)}$.

Digression: The perceptron learning algorithm

Consider modifying logistic regression such that it outputs 0 or 1 exactly. This can be done by changing g to be the threshold function:

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Letting $h_\theta(x) = g(\theta^T x)$ with this modified g, and using the update rule

$$\theta_j \leftarrow \theta_j + \alpha\left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right) x_j^{(i)}$$

give the ***perceptron learning algorithm***.

Another algorithm for maximizing $\ell(\theta)$

Consider Newton's method for finding a zero of a function, i.e. suppose a function $f : \mathbb{R} \mapsto \mathbb{R}$, and we want to find a value θ such that $f(\theta) = 0$.

$$\begin{aligned}
y &= f'(\theta_n)\left(\theta_{n+1} - \theta_n\right) + f(\theta_n) && \text{first order (linear) Taylor approx. about } \theta_n \\
0 &= f'(\theta_n)\left(\theta_{n+1} - \theta_n\right) + f(\theta_n) && \text{solve for where linear approx. = 0} \\
f'(\theta_n)\left(\theta_{n+1} - \theta_n\right) &= -f(\theta_n) \\
\theta_{n+1} - \theta_n &= -\frac{f(\theta_n)}{f'(\theta_n)} \\
\theta_{n+1} &= \theta_n - \frac{f(\theta_n)}{f'(\theta_n)} && \text{update rule for Newton's method}
\end{aligned}$$

What if we wanted to use this method to maximize a function ℓ? The maxima of ℓ are the zeros of the first derivative $\ell'(\theta)$. Therefore, we can let $f(\theta) = \ell'(\theta)$ and use the same algorithm to maximize ℓ. The update rule is then:

$$\theta \leftarrow \theta - \frac{\ell'(\theta)}{\ell''(\theta)}$$

Newton's method generalized to the multidimensional setting, aka the ***Newton-Raphson method***:

$$\theta \leftarrow \theta - H^{-1}\nabla_\theta \ell(\theta)$$

, where H is the ***Hessian***.

Newton's method typically converges faster than batch gradient descent and requires fewer iterations to get close to the minimum (because it takes curvature into account). One iteration of Newton's can, however, be more expensive than one iteration of gradient descent since it requires finding and inverting an n by n Hessian; but so long as n is not too large, it is usually much faster overall. Newton's method applied to maximizing the logistic regression log likelihood function $\ell(\theta)$ is called ***Fisher scoring***.

Generalized Linear Models

The regression example with $y \mid x;\theta \sim \mathcal{N}(\mu, \sigma^2)$, and the classification example with $y \mid x;\theta \sim$ Bernoulli(ϕ) are special cases of ***Generalized Linear Models (GLMs)***. Other models in the GLM family can be applied to classification and regression.

The exponential family

A class of distributions is in the exponential family if it can be witten in the form

$$p(y;\eta) = b(y)\exp(\eta^T T(y) - a(\eta))$$

- $\eta \to$ the ***natural parameter***, aka the ***canonical parameter***
- $T(y) \to$ the ***sufficient statistic*** (often, $T(y) = y$)
- $a(\eta) \to$ the ***log partition function***. The $e^{-a(\eta)}$ term normalizes the distribution such that it sums/integrates to 1.

A fixed choice of T, a, and b defines a *family* (set) of distributions parameterized by η.

The Bernoulli and Gaussian distributions are exponential family distributions.

The Bernoulli distribution:

$$\begin{aligned} p(y;\phi) &= \phi^y \left(1-\phi\right)^{1-y} \\ &= \exp\left(y \log\theta + (1-y)\log(1-\phi)\right) \\ &= \exp\left(\left(\log\left(\frac{\phi}{1-\phi}\right)\right) y + \log(1-\phi)\right) \end{aligned}$$

Therefore,

$$\begin{aligned} \eta &= \log\left(\frac{\phi}{1-\phi}\right) \quad \xrightarrow{\text{invert}} \phi = \frac{1}{1+e^{-\eta}} \\ T(y) &= y \\ a(\eta) &= -\log(1-\phi) \\ &= \log\left(1+e^{\eta}\right) \\ b(y) &= 1 \end{aligned}$$

The Gaussian distribution. The value of σ^2 has no effect on the choice of θ and $h_\theta(x)$. Thus, we can set $\sigma^2 = 1$ to simplify the derivation:

$$\begin{aligned} p(y;\mu) &= \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}\left(y-\mu\right)^2\right) \\ &= \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}y^2 + \mu y - \frac{1}{2}\mu^2\right) \\ &= \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}y^2\right)\cdot\exp\left(\mu y - \frac{1}{2}\mu^2\right) \end{aligned}$$

Therefore,

$$\begin{aligned} \eta &= \mu \\ T(y) &= y \\ a(\eta) &= \frac{\mu^2}{2} \\ &= \frac{\eta^2}{2} \\ b(y) &= \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-y^2}{2}\right) \end{aligned}$$

Other members of the exponential family:

- multinomial
- Poisson (for count data)
- gamma and exponential (for continuous, non-negative random variables, e.g. time-intervals)
- beta and Dirichlet (for distributions over probabilities)

Constructing GLMs

Consider a classification or regression problem with the goal of predicting the value of a random variable y as a function of x. To derive a GLM for this problem, make three assumptions about $P(y \mid x)$ and the model:

1. $y \mid x; \theta \sim$ ExponentialFamily(η)
2. Goal: predict E $[T(y)]$ given x.
    - In most examples, $T(y) = y$, so we want the prediction $h(x) =$ E $[y \mid x]$
3. The natural parameter η and the inputs x are linearly related: $\eta = \theta^T x$ ($\eta_i = \theta_i^T x$)

Ordinary Least Squares as a GLM

- target variable (aka ***response variable***) y is continuous
- model $P(y \mid x) \sim \mathcal{N}(\mu, \sigma^2)$

Then,

$$\begin{aligned} h_\theta(x) &= \text{E}\left[y \mid x; \theta\right] && \text{assumption 2} \\ &= \mu && \text{expected value of } \mathcal{N}(\mu, \sigma^2) \\ &= \eta && \mu = \eta \text{ from Gaussian as exp. family dist.} \\ &= \theta^T x && \text{assumption 3} \end{aligned}$$

Logistic Regression as a GLM

- target variable y is binary-valued ($y \in \{0, 1\}$)
- model $P(y \mid x) \sim$ Bernoulli(ϕ)

Then,

$$\begin{aligned} h_\theta(x) &= \text{E}\left[y \mid x; \theta\right] && \text{assumption 2} \\ &= \phi && \text{expected value of Bernoulli}(\phi) \\ &= \frac{1}{1 + e^{-\eta}} && \phi(\eta) \text{ from Bernoulli as exp. family dist.} \\ &= \frac{1}{1 + e^{-\theta^T x}} && \text{assumption 3} \end{aligned}$$

So, the hypothesis function above arises from the definitions of GLMs and exponential family distributions, assuming that $y \mid x \sim$ Bernoulli.

Some terminology:

- the function g giving the expected value as a function of the natural parameter η is called the ***canonical response function***:

$$g(\eta) = \mathsf{E}\left[T(y); \eta\right]$$

  - for the Gaussian family, $g(\eta)$ is the identity function
  - for the Bernoulli, $g(\eta)$ is the logistic function

- its inverse, g^{-1} is called the ***canonical link function***

Softmax Regression as a GLM

- classification, $y \in \{1, 2, \cdots, k\}$.
- model $P(y \mid x) \sim$ Multinomial$(\phi_1, \cdots, \phi_{k-1})$ (parameterized by $k-1$ parameters since ϕ_k must satisfy $\sum_{i=1}^{k-1} \phi_i$).
- define $T(y) \in \mathbb{R}^{k-1}$ as follows:

$$T(1) = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad T(2) = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \quad \cdots, \quad T(k-1) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}, \quad T(k) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

It is useful to now introduce the ***indicator function*** $1\{\cdot\}$:

$$\begin{cases} 1\{\text{True}\} = 1 \\ 1\{\text{False}\} = 0 \end{cases}$$

Using the indicator function, we can write

$$(T(y))_i = 1\{y = i\}$$

Also,

$$\mathsf{E}\left[(T(y))_i\right] = P(y = i) = \phi_i$$

Show that softmax regression is a member of the exponential family

$$\begin{aligned}
p(y; \phi) &= \phi_1^{1\{y=1\}} \phi_2^{1\{y=2\}} \dots \phi_k^{1\{y=k\}} \\
&= \phi_1^{1\{y=1\}} \phi_2^{1\{y=2\}} \dots \phi_k^{1 - \sum_{i=1}^{k-1} 1\{y=k\}} \\
&= \phi_1^{(T(y))_1} \phi_2^{(T(y))_2} \dots \phi_k^{(T(y))_i} \\
&= \exp\left((T(y))_1 \log(\phi_1) + (T(y))_2 \log(\phi_2) + \cdots + \left(1 - \sum_{i=1}^{k-1} (T(y))_i \right) \log(\phi_k) \right) \\
&= \exp\left((T(y))_1 \log\left(\frac{\phi_1}{\phi_k}\right) + (T(y))_2 \log\left(\frac{\phi_2}{\phi_k}\right) + \cdots + (T(y))_{k-1} \log\left(\frac{\phi_{k-1}}{\phi_k}\right) + \log(\phi_k) \right) \\
&= b(y) \exp\left(\eta^T T(y) - a(\eta)\right)
\end{aligned}$$

where

$$\begin{aligned}
\eta &= \begin{bmatrix} \log\left(\frac{\phi_1}{\phi_k}\right) \\ \log\left(\frac{\phi_2}{\phi_k}\right) \\ \vdots \\ \log\left(\frac{\phi_{k-1}}{\phi_k}\right) \end{bmatrix}, \\
a(\eta) &= -\log\left(\phi_k\right), \\
b(y) &= 1
\end{aligned}$$

The link function is given by

$$\eta_i = \log\left(\frac{\phi_i}{\phi_k}\right), \qquad \text{for } i = 1, \dots, k$$

The response function is found by inverting the link function, first solving for ϕ_k:

$$\begin{aligned} e^{\eta_i} &= \frac{\phi_i}{\phi_k} \\ \phi_k e^{\eta_i} &= \phi_i \\ \phi_k \sum_{i=1}^{k} e^{\eta_i} &= \sum_{i=1}^{k} \phi_i = 1 \\ \phi_k &= \frac{1}{\sum_{i=1}^{k} e^{\eta_i}} \end{aligned}$$

Then, substituting to solve for ϕ_i:

$$\phi_i = \frac{e^{\eta_i}}{\sum_{j=1}^{k} e^{\eta_j}}$$

This response function is called the ***softmax*** function.

To complete the GLM, we use assumption 3, i.e. the $\eta_i = \theta^T x$ for $i = 1, \dots, k-1$. We can also define $\theta_k = 0$, so that $\eta_k = \theta_k^T x = 0$. Hence,

$$\begin{aligned} p(y = i \mid x; \theta) &= \phi_i \\ &= \frac{e^{\eta_i}}{\sum_{j=1}^{k} e^{\eta_j}} \\ &= \frac{e^{\theta_i^T x}}{\sum_{j=1}^{k} e^{\theta^T x}} \end{aligned}$$

This is the ***softmax regression*** model, a generalization of logistic regression to k classes.

The hypothesis function will output:

$$\begin{aligned} h_\theta(x) &= \mathsf{E}\left[T(y) \mid x; \theta\right] \\ &= \left[\begin{matrix} 1\{y = 1\} \\ 1\{y = 2\} \\ \vdots \\ 1\{y = k-1\} \end{matrix} \quad \middle| \quad x; \theta\right] \\ &= \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{k-1} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\exp\left(\theta_1^T x\right)}{\sum_{j=1}^{k} \exp\left(\theta_j^T x\right)} \\ \frac{\exp\left(\theta_2^T x\right)}{\sum_{j=1}^{k} \exp\left(\theta_j^T x\right)} \\ \vdots \\ \frac{\exp\left(\theta_{k-1}^T x\right)}{\sum_{j=1}^{k} \exp\left(\theta_j^T x\right)} \end{bmatrix} \end{aligned}$$

I.e., the hypothesis will output the estimated probability that $p(y = i \mid x; \theta)$, for every value of $i = 1, \dots, k$.

How do we fit to learn the parameters θ_i? Given a training set of m examples $\left\{\left(x^{(i)}, y^{(i)}\right); i = 1, \ldots, m\right\}$, we can do it by maximizing the log-likelihood

$$\begin{aligned}\ell(\theta) &= \sum_{i=1}^{m} \log p\left(y^{(i)} \mid x^{(i)}; \theta\right) \\ &= \sum_{i=1}^{m} \log \prod_{p=1}^{k} \left(\frac{e^{\theta_p^T x^{(i)}}}{\sum_{j-1}^{k} e^{\theta_j^T} x^{(i)}}\right)^{1\left\{y^{(i)}=p\right\}}\end{aligned}$$

The log-likelihood can then be maximized using a method such as gradient ascent or Newton's method.