Least Squares and MLE (Gaussian Noise)

Least Squares

Objective Function

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2$$

MLE (Gaussian Noise)

Likelihood

$$p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) = \frac{1}{(2\pi\sigma^2)^{N/2}} \prod_{i=1}^{N} \exp\left(-\frac{(y_i - \mathbf{w} \cdot \mathbf{x}_i)^2}{2\sigma^2} \right)$$

For estimating $\mathbf{w}$, the negative log-likelihood under Gaussian noise has the same form as the least squares objective

Alternatively, we can model the data (only y_i-s) as being generated from a distribution defined by exponentiating the negative of the objective function

What Data Model Produces the Ridge Objective?

We have the Ridge Regression Objective, let $\mathcal{D} = \langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$ denote the data

$$\mathcal{L}_{\text{ridge}}(\mathbf{w}; \mathcal{D}) = (\mathbf{y} - \mathbf{X}\mathbf{w})^\mathsf{T}(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w}^\mathsf{T}\mathbf{w}$$

Let's rewrite this objective slightly, scaling by $\frac{1}{2\sigma^2}$ and setting $\lambda = \frac{\sigma^2}{\tau^2}$. To avoid ambiguity, we'll denote this by $\widetilde{\mathcal{L}}$

$$\widetilde{\mathcal{L}}_{\text{ridge}}(\mathbf{w}; \mathcal{D}) = \frac{1}{2\sigma^2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\mathsf{T}(\mathbf{y} - \mathbf{X}\mathbf{w}) + \frac{1}{2\tau^2}\mathbf{w}^\mathsf{T}\mathbf{w}$$

Let $\boldsymbol{\Sigma} = \sigma^2 \mathbf{I}_N$ and $\boldsymbol{\Lambda} = \tau^2 \mathbf{I}_D$, where $\mathbf{I}_m$ denotes the $m \times m$ identity matrix

$$\widetilde{\mathcal{L}}_{\text{ridge}}(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\mathbf{y} - \mathbf{X}\mathbf{w}) + \frac{1}{2}\mathbf{w}^\mathsf{T}\boldsymbol{\Lambda}^{-1}\mathbf{w}$$

Taking the negation of $\widetilde{\mathcal{L}}_{\text{ridge}}(\mathbf{w}; \mathcal{D})$ and exponentiating gives us a non-negative function of $\mathbf{w}$ and $\mathcal{D}$ which after normalisation gives a density function

$$f(\mathbf{w}; \mathbf{D}) = \exp\left(-\frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\mathbf{y} - \mathbf{X}\mathbf{w})\right) \cdot \exp\left(-\frac{1}{2}\mathbf{w}^\mathsf{T}\boldsymbol{\Lambda}^{-1}\mathbf{w}\right)$$

Bayesian Linear Regression (and connections to Ridge)

Let's start with the form of the density function we had on the previous slide and factor it.

$$f(\mathbf{w};\mathbf{D}) = \exp\left(-\frac{1}{2}(\mathbf{y}-\mathbf{X}\mathbf{w})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{y}-\mathbf{X}\mathbf{w})\right)\cdot\exp\left(-\frac{1}{2}\mathbf{w}^{\mathsf{T}}\boldsymbol{\Lambda}^{-1}\mathbf{w}\right)$$

We'll treat σ as fixed and not treat is as a parameter. Up to a constant factor (which does't matter when optimising w.r.t. $\mathbf{w}$), we can rewrite this as

$$\underbrace{p(\mathbf{w} \mid \mathbf{X},\mathbf{y})}_{\text{posterior}} \propto \underbrace{\mathcal{N}(\mathbf{y} \mid \mathbf{X}\mathbf{w},\Sigma)}_{\text{Likelihood}} \cdot \underbrace{\mathcal{N}(\mathbf{w} \mid \mathbf{0},\boldsymbol{\Lambda})}_{\text{prior}}$$

where $\mathcal{N}(\cdot \mid \boldsymbol{\mu},\boldsymbol{\Sigma})$ denotes the density of the multivariate normal distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$

- What the ridge objective is actually finding is the maximum a posteriori or (MAP) estimate which is a mode of the posterior distribution
- The linear model is as described before with Gaussian noise
- The prior distribution on $\mathbf{w}$ is assumed to be a spherical Gaussian

Bayesian Machine Learning

In the discriminative framework, we model the output y as a probability distribution given the input $\mathbf{x}$ and the parameters $\mathbf{w}$, say $p(y \mid \mathbf{w}, \mathbf{x})$

In the Bayesian view, we assume a prior on the parameters $\mathbf{w}$, say $p(\mathbf{w})$

This prior represents a "belief" about the model; the uncertainty in our knowledge is expressed mathematically as a probability distribution

When observations, $\mathcal{D} = \langle(\mathbf{x}_i, y_i)\rangle_{i=1}^{N}$ are made the belief about the parameters $\mathbf{w}$ is updated using Bayes' rule

Bayes Rule

For events A, B, $\Pr[A \mid B] = \dfrac{\Pr[B \mid A] \cdot \Pr[A]}{\Pr[B]}$

The posterior distribution on $\mathbf{w}$ given the data $\mathcal{D}$ becomes:

$$p(\mathbf{w} \mid \mathcal{D}) \propto p(\mathbf{y} \mid \mathbf{w}, \mathbf{X}) \cdot p(\mathbf{w})$$

Full Bayesian Prediction

Let us recall the posterior distribution over parameters $\mathbf{w}$ in the Bayesian approach

$$\underbrace{p(\mathbf{w} \mid \mathbf{X}, \mathbf{y})}_{\text{posterior}} \propto \underbrace{p(\mathbf{y} \mid \mathbf{X}, \mathbf{w})}_{\text{likelihood}} \cdot \underbrace{p(\mathbf{w})}_{\text{prior}}$$

- If we use the MAP estimate, as we get more samples the posterior peaks at the MLE
- When, data is scarce rather than picking a single estimator (like MAP) we can sample from the full posterior

For $\mathbf{x}_{\text{new}}$, we can output the entire distribution over our prediction $\widehat{y}$ as

$$p(y \mid \mathcal{D}) = \int_{\mathbf{w}} \underbrace{p(y \mid \mathbf{w}, \mathbf{x}_{\text{new}})}_{\text{model}} \cdot \underbrace{p(\mathbf{w} \mid \mathbf{D})}_{\text{posterior}} \, \mathrm{d}\mathbf{w}$$

This integration is often computationally very hard!

Summary : Bayesian Machine Learning

In the Bayesian view, in addition to modelling the output y as a random variable given the parameters $\mathbf{w}$ and input $\mathbf{x}$, we also encode prior belief about the parameters $\mathbf{w}$ as a probability distribution $p(\mathbf{w})$.

- If the prior has a parametric form, they are called hyperparameters
- The posterior over the parameters $\mathbf{w}$ is updated given data
- Either pick point (plugin) estimates, *e.g.,* maximum a posteriori
- Or as in the full Bayesian approach use the entire posterior to make prediction (this is often computationally intractable)
- How to choose the prior?

How to Choose Hyper-parameters?

- So far, we were just trying to estimate the parameters $\mathbf{w}$
- For Ridge Regression or Lasso, we need to choose λ
- If we perform basis expansion
  - For kernels, we need to pick the width parameter γ
  - For polynomials, we need to pick degree d
- For more complex models there may be more hyperparameters

Using a Validation Set

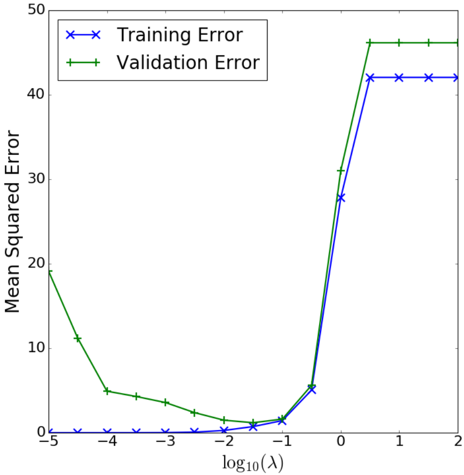
- Divide the data into parts: training, validation (and testing)
- Grid Search: Choose values for the hyperparameters from a finite set
- Train model using training set and evaluate on validation set

λ	training error(%)	validation error(%)
0.01	0	89
0.1	0	43
1	2	12
10	10	8
100	25	27

- Pick the value of λ that minimises validation error
- Typically, split the data as 80% for training, 20% for validation

Training and Validation Curves

- Plot of training and validation error vs λ for Lasso
- Validation error curve is U-shaped



K-Fold Cross Validation

When data is scarce, instead of splitting as training and validation:

- Divide data into K parts
- Use $K-1$ parts for training and 1 part as validation
- Commonly set $K=5$ or $K=10$
- When $K=N$ (the number of datapoints), it is called LOOCV (Leave one out cross validation)

Run 1	train	train	train	train	valid
Run 2	train	train	train	valid	train
Run 3	train	train	valid	train	train
Run 4	train	valid	train	train	train
Run 5	valid	train	train	train	train

Logistic Regression (LR)

- LR builds up on a linear model, composed with a sigmoid function

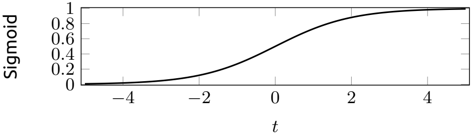
$$p(y \mid \mathbf{w}, \mathbf{x}) = \mathrm{Bernoulli}(\mathrm{sigmoid}(\mathbf{w} \cdot \mathbf{x}))$$

- $Z \sim \mathrm{Bernoulli}(\theta)$

$$Z = \begin{cases} 1 & \text{with probability } \theta \\ 0 & \text{with probability } 1 - \theta \end{cases}$$

- Recall that the sigmoid function is defined by:

$$\mathrm{sigmoid}(t) = \frac{1}{1 + e^{-t}}$$



- As we did in the case of linear models, we assume $x_0 = 1$ for all datapoints, so we do not need to handle the bias term w_0 separately

Prediction Using Logistic Regression

Suppose we have estimated the model parameters $\mathbf{w} \in \mathbb{R}^D$

For a new datapoint $\mathbf{x}_{\text{new}}$, the model gives us the probability

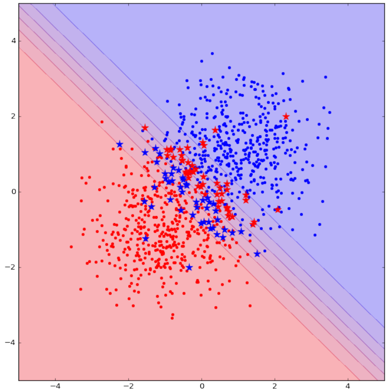
$$p(y_{\text{new}} = 1 \mid \mathbf{x}_{\text{new}}, \mathbf{w}) = \text{sigmoid}(\mathbf{w} \cdot \mathbf{x}_{\text{new}}) = \frac{1}{1 + \exp(-\mathbf{x}_{\text{new}} \cdot \mathbf{w})}$$

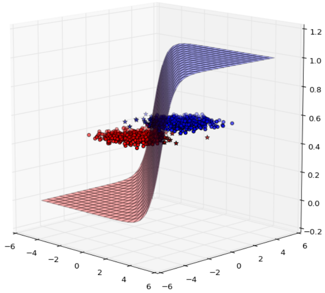
In order to make a prediction we can simply use a threshold at $\frac{1}{2}$

$$\widehat{y}_{\text{new}} = \mathbb{I}(\text{sigmoid}(\mathbf{w} \cdot \mathbf{x}_{\text{new}})) \geq \frac{1}{2}) = \mathbb{I}(\mathbf{w} \cdot \mathbf{x}_{\text{new}} \geq 0)$$

Class boundary is linear (separating hyperplane)

Prediction Using Logistic Regression





Likelihood of Logistic Regression

Data $\mathcal{D} = \langle(\mathbf{x}_i, y_i)\rangle_{i=1}^{N}$, where $\mathbf{x}_i \in \mathbb{R}^D$ and $y_i \in \{0, 1\}$

Let us denote the sigmoid function by σ

We can write the likelihood for of observing the data given model parameters $\mathbf{w}$ as:

$$p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) = \prod_{i=1}^{N} \sigma(\mathbf{w}^{\mathsf{T}}\mathbf{x}_i)^{y_i} \cdot (1 - \sigma(\mathbf{w}^{\mathsf{T}}\mathbf{x}_i))^{1-y_i}$$

Let us denote $\mu_i = \sigma(\mathbf{w}^{\mathsf{T}}\mathbf{x}_i)$

We can write the negative log-likelihood as:

$$\mathrm{NLL}(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) = -\sum_{i=1}^{N}(y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i))$$

Likelihood of Logistic Regression

Recall that $\mu_i = \sigma(\mathbf{w}^\mathsf{T}\mathbf{x}_i)$ and the negative log-likelihood is

$$\mathrm{NLL}(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) = -\sum_{i=1}^{N}(y_i \log \mu_i + (1 - y_i)\log(1 - \mu_i))$$

Let us focus on a single datapoint, the contribution to the negative log-likelihood is

$$\mathrm{NLL}(y_i \mid \mathbf{x}_i, \mathbf{w}) = -(y_i \log \mu_i + (1 - y_i)\log(1 - \mu_i))$$

This is basically the cross-entropy between y_i and μ_i

If $y_i = 1$, then as

- As $\mu_i \to 1$, $\mathrm{NLL}(y_i \mid \mathbf{x}_i, \mathbf{w}) \to 0$
- As $\mu_i \to 0$, $\mathrm{NLL}(y_i \mid \mathbf{x}_i, \mathbf{w}) \to \infty$

Maximum Likelihood Estimate for LR

Recall that $\mu_i = \sigma(\mathbf{w}^\mathsf{T}\mathbf{x}_i)$ and the negative log-likelihood is

$$\mathrm{NLL}(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) = -\sum_{i=1}^{N}(y_i \log \mu_i + (1 - y_i)\log(1 - \mu_i))$$

We can take the gradient with respect to $\mathbf{w}$

$$\nabla_{\mathbf{w}}\mathrm{NLL}(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) = \sum_{i=1}^{N}\mathbf{x}_i(\mu_i - y_i) = \mathbf{X}^\mathsf{T}(\boldsymbol{\mu} - \mathbf{y})$$

And the Hessian is given by,

$$\mathbf{H} = \mathbf{X}^\mathsf{T}\mathbf{S}\mathbf{X}$$

$\mathbf{S}$ is a diagonal matrix where $S_{ii} = \mu_i(1 - \mu_i)$

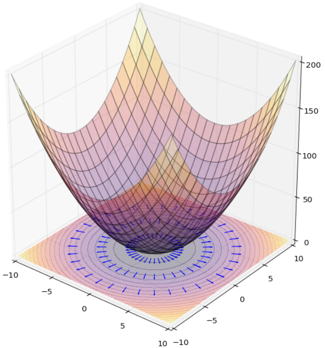
Calculus Background: Gradients

$$z = f(w_1, w_2) = \frac{w_1^2}{a^2} + \frac{w_2^2}{b^2}$$

$$\frac{\partial f}{\partial w_1} = \frac{2w_1}{a^2}$$

$$\frac{\partial f}{\partial w_2} = \frac{2w_2}{b^2}$$

$$\nabla_{\mathbf{w}} f = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \frac{\partial f}{\partial w_2} \end{bmatrix} = \begin{bmatrix} \frac{2w_1}{a^2} \\ \frac{2w_2}{b^2} \end{bmatrix}$$



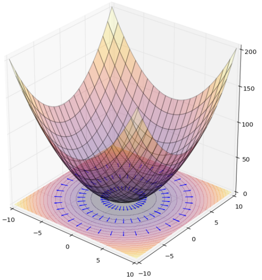
- Gradient vectors are orthogonal to contour curves
- Gradient points in the direction of steepest increase

Calculus Background: Hessians

$$z = f(w_1, w_2) = \frac{w_1^2}{a^2} + \frac{w_2^2}{b^2}$$

$$\nabla_{\mathbf{w}} f = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \frac{\partial f}{\partial w_2} \end{bmatrix} = \begin{bmatrix} \frac{2w_1}{a^2} \\ \frac{2w_2}{b^2} \end{bmatrix}$$

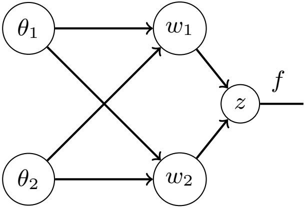
$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial w_1^2} & \frac{\partial^2 f}{\partial w_1 \partial w_2} \\ \frac{\partial^2 f}{\partial w_2 \partial w_1} & \frac{\partial^2 f}{\partial w_2^2} \end{bmatrix} = \begin{bmatrix} \frac{2}{a^2} & 0 \\ 0 & \frac{2}{b^2} \end{bmatrix}$$



- As long as all second derivates exist, the Hessian H is symmetric
- Hessian captures the curvature of the surface

Calculus Background: Chain Rule

$$z = f(w_1(\theta_1, \theta_2), w_2(\theta_1, \theta_2))$$



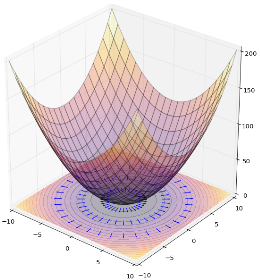
$$\frac{\partial f}{\partial \theta_1} = \frac{\partial f}{\partial w_1} \cdot \frac{\partial w_1}{\partial \theta_1} + \frac{\partial f}{\partial w_2} \cdot \frac{\partial w_2}{\partial \theta_1}$$

We will use this a lot when we study neural networks and back propagation

General Form for Gradient and Hessian

Suppose $\mathbf{w} \in \mathbb{R}^D$ and $f : \mathbb{R}^D \to \mathbb{R}$

The gradient vector contains all first order partial derivatives



$$\nabla_{\mathbf{w}} f(\mathbf{w}) = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \frac{\partial f}{\partial w_2} \\ \vdots \\ \frac{\partial f}{\partial w_D} \end{bmatrix}$$

Hessian matrix of f contains all second order partial derivatives.

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial w_1^2} & \frac{\partial^2 f}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 f}{\partial w_1 \partial w_D} \\ \frac{\partial^2 f}{\partial w_2 \partial w_1} & \frac{\partial^2 f}{\partial w_2^2} & \cdots & \frac{\partial^2 f}{\partial w_2 \partial w_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial w_D \partial w_1} & \frac{\partial^2 f}{\partial w_D \partial w_2} & \cdots & \frac{\partial^2 f}{\partial w_D^2} \end{bmatrix}$$

Gradient Descent Algorithm

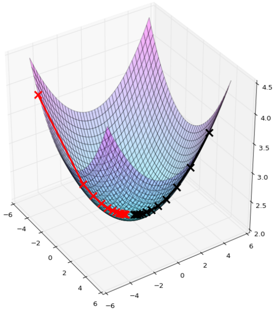
Gradient descent is one of the simplest, but very general algorithm for optimization

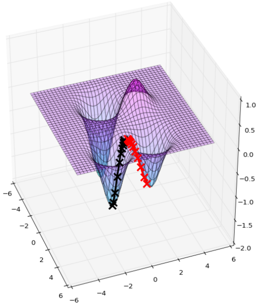
It is an iterative algorithm, producing a new $\mathbf{w}_{t+1}$ at each iteration as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \mathbf{g}_t = \mathbf{w}_t - \eta_t \nabla f(\mathbf{w}_t)$$

We will denote the gradients by $\mathbf{g}_t$

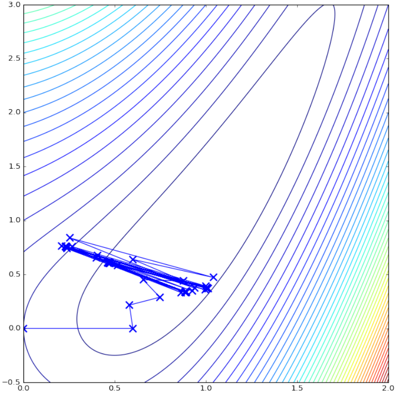
$\eta_t > 0$ is the learning rate or step size

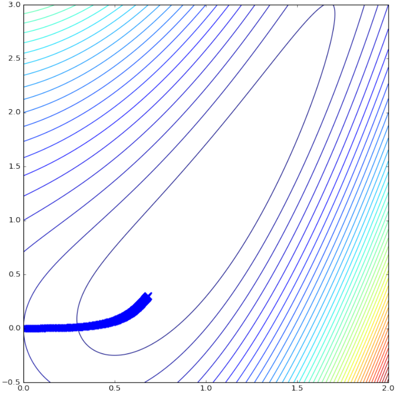




Choosing a Step Size

- Choosing a good step-size is important
- It step size is too large, algorithm may never converge
- If step size is too small, convergence may be very slow
- May want a time-varying step size





Iteratively Re-Weighted Least Squares (IRLS)

Depending on the dimension, we can apply Newton's method to estimate $\mathbf{w}$

Let $\mathbf{w}_t$ be the parameters after t Newton steps.

The gradient and Hessian are given by:

$$\begin{aligned}\mathbf{g}_t &= \mathbf{X}^\mathsf{T}(\boldsymbol{\mu}_t - \mathbf{y}) = -\mathbf{X}^\mathsf{T}(\mathbf{y} - \boldsymbol{\mu}_t) \\ \mathbf{H}_t &= \mathbf{X}^\mathsf{T}\mathbf{S}_t\mathbf{X}\end{aligned}$$

The Newton Update Rule is:

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \mathbf{H}_t^{-1}\mathbf{g}_t \\ &= \mathbf{w}_t + (\mathbf{X}^\mathsf{T}\mathbf{S}_t\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}(\mathbf{y} - \boldsymbol{\mu}_t) \\ &= (\mathbf{X}^\mathsf{T}\mathbf{S}_t\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{S}_t(\mathbf{X}\mathbf{w}_t + \mathbf{S}_t^{-1}(\mathbf{y} - \boldsymbol{\mu}_t)) \\ &= (\mathbf{X}^\mathsf{T}\mathbf{S}_t\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{S}_t\mathbf{z}_t\end{aligned}$$

Where $\mathbf{z}_t = \mathbf{X}\mathbf{w}_t + \mathbf{S}_t^{-1}(\mathbf{y} - \boldsymbol{\mu}_t)$. Then $\mathbf{w}_{t+1}$ is a solution of the following:

Weighted Least Squares Problem

$$\text{minimise} \quad \sum_{i=1}^{N} S_{t,ii}(z_{t,i} - \mathbf{w}^\mathsf{T}\mathbf{x}_i)^2$$

Multiclass Logistic Regression

Multiclass logistic regression is also a discriminative classifier

Let the inputs be $\mathbf{x} \in \mathbb{R}^D$ and $y \in \{1, \ldots, C\}$

There are parameters $\mathbf{w}_c \in \mathbb{R}^D$ for every class $c = 1, \ldots, C$

We'll put this together in a matrix form $\mathbf{W}$ that is $D \times C$

The multiclass logistic model is given by:

$$p(y = c \mid \mathbf{x}, \mathbf{W}) = \frac{\exp(\mathbf{w}_c^\mathsf{T} \mathbf{x})}{\sum_{c'=1}^{C} \exp(\mathbf{w}_{c'}^\mathsf{T} \mathbf{x})}$$

Multiclass Logistic Regression

The multiclass logistic model is given by:

$$p(y = c \mid \mathbf{x}, \mathbf{W}) = \frac{\exp(\mathbf{w}_c^\mathsf{T}\mathbf{x})}{\sum_{c'=1}^{C} \exp(\mathbf{w}_{c'}^\mathsf{T}\mathbf{x})}$$

Recall the softmax function

Softmax

Softmax maps a set of numbers to a probability distribution with mode at the maximum

$$\operatorname{softmax}\left([a_1, \ldots, a_C]^\mathsf{T}\right) = \left[\frac{e^{a_1}}{Z}, \ldots, \frac{e^{a_C}}{Z}\right]^\mathsf{T}$$

where $Z = \sum_{c=1}^{C} e^{a_c}$.

The multiclass logistic model is simply:

$$p(y \mid \mathbf{x}, \mathbf{W}) = \operatorname{softmax}\left(\left[\mathbf{w}_1^\mathsf{T}\mathbf{x}, \ldots, \mathbf{w}_C^\mathsf{T}\mathbf{x}\right]^\mathsf{T}\right)$$

Multiclass Logistic Regression