Today: Multinomial (multiclass) logistic regression
Regularization
Convex functions & sets w/ examples

NB: Hw2 is posted, due next Monday.
You will need to:
- use Jupyter, Scikit-learn, matplotlib, numpy

Multinomial Logistic Reg

Given features $x \in \mathbb{R}^d$ goal is to predict class $y \in [K] = \{1, \dots, K\}$

Ex: given image $x \in \mathbb{R}^{m \times n}$ determine what object it contains

Model: assume the log-odds (logits or scores) of each class given x relative to class K are linear:

$$\log \frac{\mathbb{P}(Y = 1 \mid X = x)}{\mathbb{P}(Y = K \mid X = x)} = b_1 + w_1^T x$$

$$\vdots$$

$$\log \frac{\mathbb{P}(Y=2 \mid X=x)}{\mathbb{P}(Y=K \mid X=x)} = b_2 + \omega_2^T x$$

$$\vdots$$

$$\log \frac{\mathbb{P}(Y=K-1 \mid X=x)}{\mathbb{P}(Y=K \mid X=x)} = b_{K-1} + \omega_{K-1}^T x$$

These $K-1$ equations plus the constraint that $\sum_{i=1}^{K} \mathbb{P}(Y=i) = 1$

imply

$$\mathbb{P}(Y=i \mid X=x) = \frac{\exp(b_i + \omega_i^T x)}{1 + \sum_{l=1}^{K-1} \exp(b_l + \omega_l^T x)} \qquad i = 1, \dots, K-1$$

$$\mathbb{P}(Y=K \mid X=x) = \frac{1}{1 + \sum_{l=1}^{K-1} \exp(b_l + \omega_l^T x)}$$

For ease of use, we can introduce some unnecessary parameters b_K, ω_K to get the simpler expressions

$$\mathbb{P}(y=i \mid X=x) = \frac{\exp(b_i + \omega_i^T x)}{\sum_{l=1}^{K} \exp(b_l + \omega_l^T x)} = \frac{\exp(\omega x + b)_i}{1^T \exp(\omega x + b)}$$

where $\underline{b \in \mathbb{R}^K}$ and $\omega \in \mathbb{R}^{K \times d}$ $\omega = \begin{bmatrix} \omega_1^T \\ \vdots \\ \omega_K^T \end{bmatrix}$ are the model parameters

and if $v \in \mathbb{R}^K$ is a vector then $\exp(v) \in \mathbb{R}^K$ is the componentwise exponentiation of v, and $1 = 1_K$ is the K-dim vector of all ones

$$\mathbb{P}(Y = i \mid X = x) = \underline{\text{softmax}}(\underline{Wx + b})_i$$

where $\text{softmax} : v \mapsto \frac{\exp(v)}{1^T \exp(v)}$

$$\mathbb{P}(Y = j \mid X = x_i) = \text{softmax}(Wx_i + b)_j$$

softmax is a continuous approximation of the function that maps a vector v to a vector of zeros and ones where there is a one corresp to the largest entry in v, and a zero everywhere else.

Notice that softmax(v) is always a probability vector, and softmax is defined on all of $\mathbb{R}^k$. (multidim analog of the sigmoid)

Ex:

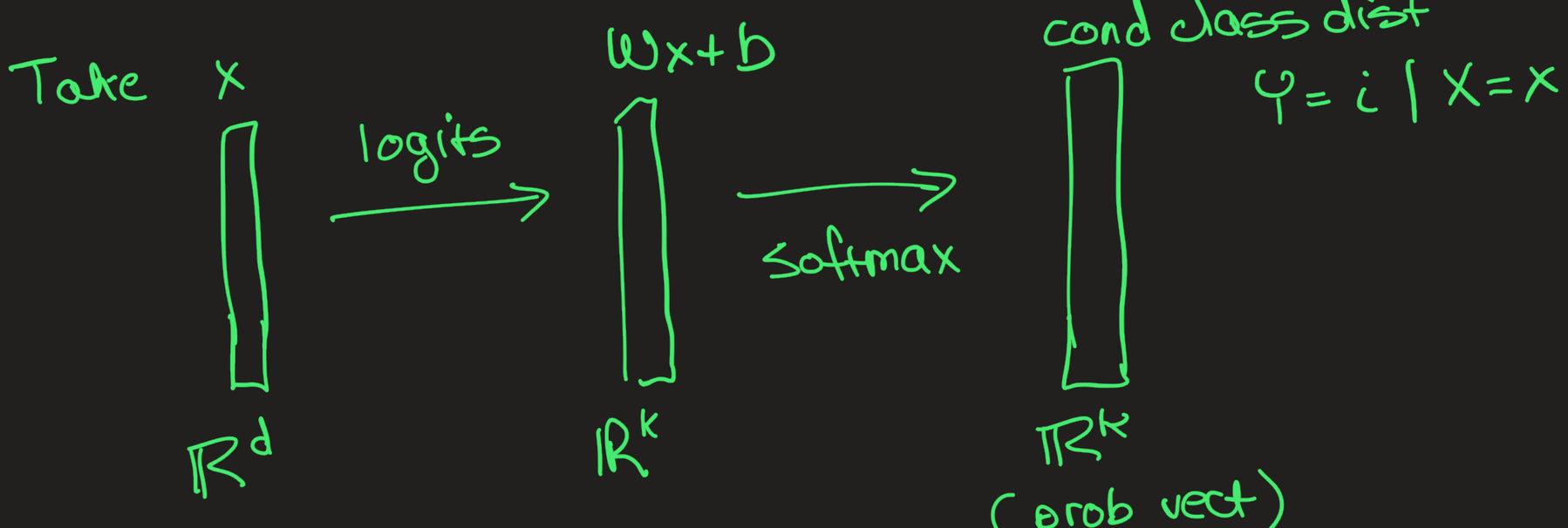
$$\begin{bmatrix} 30 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{\text{softmax}} \begin{bmatrix} 1 \\ 2.54 \times 10^{-13} \\ 9.36 \times 10^{-14} \end{bmatrix}$$

logits → prob dist over classes

$$\begin{bmatrix} .9 \\ .05 \\ .05 \end{bmatrix} \longrightarrow \begin{bmatrix} 0.539 \\ 0.230 \\ 0.230 \end{bmatrix}$$

logits

Using multiclass logistic reg. to predict given new data, assuming we know W & b.



Then classify x as having class

$$i^* = \underset{i}{\operatorname{argmax}}\ \mathbb{P}(Y=i \mid X=x)$$

$$= \underset{i}{\operatorname{argmax}}\ \operatorname{softmax}(Wx+b)_i$$

How do we determine w & b?

As before, we maximize the likelihood of our model parameters given the training data; we do this by minimizing the negative log-likelihood (NLL)

$$\mathcal{L}(w, b) := \sum_{i=1}^{n} -\log \mathbb{P}(y = y_i \mid X = x_i; w, b)$$

$$w^*, b^* = \underset{w, b}{\operatorname{argmin}} \; \mathcal{L}(w, b)$$

To write the NLL more conveniently, encode the class observations as one-hot vectors. Instead of $y_i \in [K]$ consider $y_i \in \{e_1, \dots, e_K\} \subseteq \mathbb{R}^K$ e.g. if 2 was the class of training example i,

$$y_i = e_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^K$$

Now we see that if the ith training example has class j, then

$$\log \mathbb{P}(Y = y_i \mid X = x_i) = \log \left[\text{softmax}(Wx_i + b)_j \right]$$

$$= y_i^T \log(\text{softmax}(Wx_i + b))$$

$$= y_i^T \log\left(\frac{\exp(Wx_i + b)}{1^T \exp(Wx_i + b)} \right)$$

$$= y_i^T (Wx_i + b) - \log\left(\sum_{l=1}^{k} \exp(Wx_i + b)_l \right)$$

$$= y_i^T (Wx_i + b) - \text{logsumexp}(Wx_i + b)$$

where $\text{logsumexp}(v) = \log\left(\sum_{i=1}^{k} \exp(v_i) \right)$ when $v \in \mathbb{R}^k$

Example

$k = 3$

$$y^T \log\left(\frac{\exp(Wx+b)}{I^T \exp(Wx+b)}\right) = \left\langle \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \log(\exp(w_1^T x + b_1)/Z) \\ \log(\exp(w_2^T x + b_2)/Z) \\ \log(\exp(w_3^T x + b_3)/Z) \end{bmatrix} \right\rangle$$

$$= \log(\exp(w_1^T x + b_1)/Z)$$

$$= (w_1^T x + b_i) - \log(Z)$$

We end up with the ERM problem:

$$w^*, b^* = \underset{w,b}{\operatorname{argmin}} - \sum_{i=1}^{n} \left[y_i^T (Wx_i + b) - \log \left(\sum_{l=1}^{k} \exp(Wx_i + b)_l \right) \right]$$

Recall: $x^T y = \sum_{i=1}^{k} x_i y_i = \langle x, y \rangle$

Again: there is no reason why you have to model the logits/scores as linear in x. We could use an arbitrary function

$$f: \mathbb{R}^d \to \mathbb{R}^k$$

$$f^* = \underset{f \in \vec{f}}{\operatorname{argmin}} - \sum_{i=1}^{n} \left[y_i^T f(x_i) - \log \left(\sum_{l=1}^{k} \exp(f(x_i))_l \right) \right]$$

Regularization & Overfitting

In ML we often don't have a strong idea of what is the ideal model for our data, so we try a bunch of different models and choose the one that gives better performance. If we pick <u>too</u> complicated/powerful of a class of models, we can <u>overfit</u>: have $R_n(\hat{f}_T) \ll R(f^*)$

a large generalization gap. False confidence that your model has low risk.

To avoid overfitting:

1) use Occam's razor: use very simple models when we can (structural minimization)

2) use <u>regularization</u>

Regularized ERM is obtained by adding a regularization term to your objective to encourage your complicated model to be simple

Ex: we can encourage the model to have small coeffs by adding an l_2 regularization term

$$w_* = \underset{w}{\operatorname{argmin}} \underbrace{\frac{1}{n}\sum_{i=1}^{n}(y_i - w^T x_i)^2}_{\text{data-fit term}} + \underbrace{\lambda \|w\|_2^2}_{\text{regularizer}}$$

$\lambda \rightarrow$ regularization parameter, > 0

This formulation (l_2 regularization + l_2 loss) is called ridge regression

Recall OLS estimation : Assume $y \sim \mathcal{N}(\underline{w^T x}, \sigma^2)$

↓

ordinary least squares

then determine w^* using MLE

$$\Rightarrow w^* = \underset{w}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (y_i - w^T x_i)^2$$

$$= \underset{w}{\operatorname{argmin}} \frac{1}{n} \| X_{tr} w - y_{tr} \|_2^2$$

$$= (X_{tr}^T X_{tr})^{-1} X_{tr}^T y_{tr}$$

$$X_{tr} = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix} \in \mathbb{R}^{n \times d}$$

$$y_{tr} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \in \mathbb{R}^n$$

notice Bayes optimal estimator (b/c we're using l_2-loss) for y given x is $\mathbb{E}[y|x] = w^T x$

Ridge Regression (l_2 loss + l_2 regularization)

$$w^* = \underset{w}{\operatorname{argmin}} \frac{1}{n} \| X_{tr} w - y_{tr} \|_2^2 + \lambda \|w\|_2^2$$

$$= \left(X_{tr}^T X_{tr} + n\lambda I \right)^{-1} X_{tr}^T y_{tr}$$

l_1-regularization (aka Lasso regularization)

Goal: we want only a few nonzero model weights

$$\|w\|_1 = \sum_{i=1}^{d} |w_i|$$

LASSO (Least Absolute Shrinkage & Selection Operator)

l_2 loss + l_1 regularization

$$w^* = \underset{w}{\operatorname{argmin}} \quad \frac{1}{n} \|X_{tr} w - y_{tr}\|_2^2 + \lambda \|w\|_1$$

This does <u>not</u> have a closed-form solution.

Regularization is a powerful technique that is ubiquitous in ML. In general we use the formulation

$$\hat{f}_{\mathcal{F}} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \; R_n(f) + \lambda \mathcal{R}(f)$$

where $\mathcal{R} : \mathcal{F} \to \mathbb{R}$ is the regularizer and λ is the regularization parameter. This is called regularized empirical risk minimization (RERM)

Ex: polynomial (quadratic) models vs linear models

regression against $[\underline{1}; x]$ — linear model

vs

regression against $[\underline{1}; x; x \otimes x]$ where

$$x \otimes x = \begin{bmatrix} x_1^2 \\ x_1 x_2 \\ x_1 x_3 \\ \vdots \\ x_1 x_d \\ x_2 x_1 \\ \vdots \\ x_2 x_d \\ \vdots \\ x_d x_1 \\ \vdots \\ x_d^2 \end{bmatrix} \in \mathbb{R}^{d^2}$$

See the sample code on the course website

How to chroge the regularization parameter λ?

In theory as $n \to \infty$, "let the data speak" by taking $\lambda = \Theta\left(\frac{1}{\sqrt{n}}\right)$. The idea is that even if $\mathcal{F}$ is complicated, we can avoid overfitting by seeing enough samples from the population $\mathbb{P}$.

In practice try various values of λ and choose the one that minimizes the gap in the risk between the train & test (really validation) data sets.