Categorical distributions; Discriminative models

CPSC 440/550: Advanced Machine Learning

cs.ubc.ca/~dsuth/440/24w2

University of British Columbia, on unceded Musqueam land

2024-25 Winter Term 2 (Jan–Apr 2025)

Admin

admin

- If you need a form signed, post it (privately) on Piazza
- Assignment 1 due Friday 11:59pm!
- Quiz 1 happening now through Saturday; schedule a slot ASAP

Last time

- Generative classifiers: model $p(x, y)$ and predict with e.g. $\arg\max_y p(y \mid x) = \arg\max_y p(x, y)$
- Multivariate models: product of Bernoullis, assumes X_j are all independent
- Naïve Bayes: assume the X_j are independent given Y

"Galaxy Brain Bayes"

- Naïve Bayes models $p(y)$ as Bernoulli, $p(x \mid y)$ as product of Bernoullis
  - Makes a strong assumption: all the X_j are independent given Y
- What if we avoided that assumption entirely?
- Could model $p(x \mid y)$ with a full categorical distribution:

$$\begin{aligned}
\Pr(X_1 = 0, X_2 = 0, \ldots, X_d = 0 \mid Y = 0) &= \theta_{00\cdots0|0} \\
\Pr(X_1 = 0, X_2 = 0, \ldots, X_d = 1 \mid Y = 0) &= \theta_{00\cdots1|0} \\
&\vdots \\
\Pr(X_1 = 1, X_2 = 1, \ldots, X_d = 1 \mid Y = 0) &= \theta_{11\cdots1|0}
\end{aligned}$$

  ... and the same for probabilities given $Y = 1$
- 2^d possible binary vectors, so need $2^d - 1$ parameters for each condition
- MLE is again counting, $\theta_{x|y} = n_{x|y}/n_y$, as we'll see in a moment
- Different kind of "naïvety" than naïve Bayes: each bit-vector is totally separate

Outline

Motivating problem: political polling

- Want to know support for political parties among a voter group
  - Helps candidates/parties target campaigning, etc
- Where I live, the last election results:
  - ~~40.4%~~ 23.0% Liberal
  - ~~30.7%~~ 17.5% NDP
  - ~~21.6%~~ 12.31% Conservative
  - ~~3.9%~~ 2.2% Green
  - ~~3.2%~~ 1.8% PPC
  - 43% no vote
- We want to estimate these quantities based on a sample (a poll)

General problem: categorical density estimation

- Special case of density estimation with a categorical variable:
  - Input: n iid samples of categorical values $x^{(1)}, x^{(2)}, \ldots, x^{(n)} \in \{1, 2, \ldots, k\}$
  - Output: a probability model for $\Pr(X = 1)$, $\Pr(X = 2)$, $\ldots$, $\Pr(X = k)$
- We'll remember, but not usually write down, that $1 = \texttt{Lib}$, $2 = \texttt{NDP}$, $\ldots$
- As a picture: $\mathbf{X} \in \mathbb{R}^{n \times 1}$ contains our sample data
  X is a random variable over $\{1, 2, \ldots, k\}$ from the distribution

$$\mathbf{X} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \\ 3 \end{bmatrix} \xrightarrow{\text{density estimator}} \begin{aligned} \Pr(X = 1) &= 0.4 \\ \Pr(X = 2) &= 0.2 \\ \Pr(X = 3) &= 0.4 \end{aligned}$$

- We'll start by revisiting previous concepts, but introduce some more

Other applications of categorical density estimation

- Some other questions we might ask:
  1. What portion of my customers use cash, credit, debit?
  2. What's the probability that a random patient will be able to receive this type of blood?
  3. How many random tweets should I expect to look at before I see this particular word?

- For categorical variables, we do not assume an ordering
  - Category 4 isn't "closer" to category 3 than it is to category 1

Ordinal variables

bonus!

- Ordered categorical variables are called ordinal
  - Results of rolling dice, if you're trying to beat a specific number
  - Survey results ("strongly disagree," "disagree," "neutral," ...)
  - Ratings (1 star, 2 stars, ...)
  - Tumour severity (Grade I, ..., Grade IV)

- We won't cover these for now, but lots of methods exist
- "Ordinal logistic regression": a loss function where "2 stars" is closer to "3 stars" than "4 stars"
  - But there might be a bigger "gap" between 2 and 3 stars than between 3 and 4
- Can use this "ordinal loss" in neural nets

Parametrizing categorical probabilities

- We typically use the categorical distribution (aka "multinoulli" (ugh))
- For k categories, have k parameters, $\theta_1, \ldots, \theta_k \geq 0$

$$\Pr(X = 1 \mid \theta_1, \ldots, \theta_k) = \theta_1 \quad \cdots \quad \Pr(X = k \mid \theta_1, \ldots, \theta_k) = \theta_k$$

- Categories are mutually exclusive: can only pick one
  - Require that $\sum_{c=1}^{k} \theta_c = 1$

- More succinctly: if $X \sim \mathrm{Cat}(\boldsymbol{\theta})$ with $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_k)$,

$$p(x \mid \boldsymbol{\theta}) = \theta_1^{\mathbb{1}(x=1)} \theta_2^{\mathbb{1}(x=2)} \cdots \theta_k^{\mathbb{1}(x=k)}$$

Outline

Inference task: union

- Inference task: given $\boldsymbol{\theta}$, compute probability of unions
- For example: $\Pr(X = \texttt{Lib} \cup X = \texttt{NDP} \mid \boldsymbol{\theta})$

- Can't be both, so: $\Pr(X = 2 \cup X = 4 \mid \boldsymbol{\theta}) = \theta_2 + \theta_4$

- Variation: $\Pr(X \leq c)$ for some c is $\theta_1 + \theta_2 + \cdots + \theta_c$
- Why do we care, since the categories are unordered?
- $F(c) = \Pr(X \leq c)$ is the cumulative distribution function (cdf)
  - Depends on (arbitrary) ordering, but very useful function as we'll see soon!

Inference task: mode (decoding)

- Inference task: given $\boldsymbol{\theta}$, find the mode, $\arg\max_x p(x \mid \boldsymbol{\theta})$
  - "Who's going to win the election?"
- Also very easy: $\arg\max_c \theta_c$

Inference task: likelihood

- Inference task: given $\boldsymbol{\theta}$ and data $\mathbf{X}$, find $p(\mathbf{X} \mid \boldsymbol{\theta})$
- Assuming data is iid from $\mathrm{Cat}(\boldsymbol{\theta})$,

$$
\begin{aligned}
p(\mathbf{X} \mid \boldsymbol{\theta}) &= p(x^{(1)}, \ldots, x^{(n)} \mid \boldsymbol{\theta}) = \prod_{i=1}^{n} p(x^{(i)} \mid \boldsymbol{\theta}) \\
&= \prod_{i=1}^{n} \theta_1^{\mathbb{1}(x^{(i)}=1)} \theta_2^{\mathbb{1}(x^{(i)}=2)} \cdots \theta_k^{\mathbb{1}(x^{(i)}=k)} \\
&= \theta_1^{\sum_{i=1}^{n} \mathbb{1}(x^{(i)}=1)} \theta_2^{\sum_{i=1}^{n} \mathbb{1}(x^{(i)}=2)} \cdots \theta_k^{\sum_{i=1}^{n} \mathbb{1}(x^{(i)}=k)} \\
&= \theta_1^{n_1} \theta_2^{n_2} \cdots \theta_k^{n_k}
\end{aligned}
$$

- ... defining at the end n_c as the number of cs in $\mathbf{X}$, like n_0 and n_1 for binary data
- Like Bernoulli, the likelihood only depends on the counts

Code for categorical likelihood

```
counts = np.zeros(k)
for x in X:
    count[x] += 1
p = 1
for theta_c, n_c in zip(theta, counts):
    p *= theta_c ** n_c
```

Better version:

```
counts = np.bincount(X,
↪  minlength=k)
log_p = counts @ log_theta
```

- Computational complexity (either way) is $\mathcal{O}(n + k)$
  - Usual case: $n \gg k$ (many samples, few categories), this is just $\mathcal{O}(n)$
  - If $k \gg n$, could also easily get $\mathcal{O}(n)$ by only tracking categories with nonzero counts

Inference task: sampling

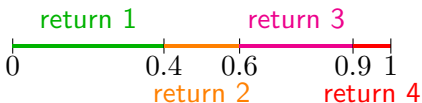
- Inference task: given $\boldsymbol{\theta}$, generate samples from $X \sim \mathrm{Cat}(\boldsymbol{\theta})$

$$\begin{array}{l}\Pr(X=1)=0.4\\ \Pr(X=2)=0.2\\ \Pr(X=3)=0.4\end{array} \quad \xrightarrow{\text{sampling}} \quad \mathbf{X} = \begin{bmatrix}1\\3\\3\end{bmatrix}$$

- Notice: not sampling "one value per class"; each sample is in one category
  - Who will this voter (say they'll) vote for?

Categorical sampling algorithm

- Will use a uniform sample from $[0, 1]$ to construct a sample from $\mathrm{Cat}(\boldsymbol{\theta})$
- Example: sample from $\boldsymbol{\theta} = (0.4, 0.2, 0.3, 0.1)$ based on a single $u \sim \mathrm{Unif}([0, 1])$
  - Want $X = 1$ 40% of the time: if $u < 0.4$, return 1
  - Want $X = 2$ 20% of the time: if $0.4 \leq u < 0.6$, return 2
  - Want $X = 3$ 30% of the time: if $0.6 \leq u < 0.9$, return 3
  - Want $X = 4$ 10% of the time: if $0.9 \leq u$, return 4



- Use CDF, $\Pr(X \leq c) = \theta_1 + \cdots + \theta_c$:
  - Generate $u \sim \mathrm{Unif}([0, 1])$
  - if $u \leq \Pr(X \leq 1)$, return 1
  - else if $u \leq \Pr(X \leq 2)$, return 2
  - ...
  - else return k

- Computing $\Pr(X \leq c)$ from $\boldsymbol{\theta}$ costs $\mathcal{O}(k)$
  - Would get $\mathcal{O}(k^2)$ total time... but can save it

```
cdf = np.cumsum(theta)
u = rng.random_sample(n_to_samp)
samp = cdf.searchsorted(u, side='right')
```

- Takes $\mathcal{O}(k)$ upfront, $\mathcal{O}(\log k)$ per sample

Faster categorical sampling algorithms

bonus!

- Previous method is sometimes called "roulette wheel sampling"
  - $\mathcal{O}(k)$ preprocessing (computing the CDF), $\mathcal{O}(\log k)$ time per sample
- "Vose's alias method": $\mathcal{O}(k)$ preprocessing but only $\mathcal{O}(1)$ time per sample
- Really nice (long) article developing many variations:
  Darts, Dice, and Coins: Sampling from a Discrete Distribution by Keith Schwarz

Outline

MLE for categorical distribution

- How do we learn a categorical model?

$$\mathbf{X} = \begin{bmatrix} \texttt{NDP} \\ \texttt{Lib} \\ \texttt{Lib} \\ \texttt{CPC} \\ \vdots \end{bmatrix} \xrightarrow{\text{density estimator}} \boldsymbol{\theta} = \begin{bmatrix} \Pr(X = \texttt{Lib}) = 0.404 \\ \Pr(X = \texttt{NDP}) = 0.307 \\ \Pr(X = \texttt{CPC}) = 0.216 \\ \Pr(X = \texttt{Grn}) = 0.039 \\ \Pr(X = \texttt{PPC}) = 0.032 \end{bmatrix}$$

- Like before, start with maximum likelihood estimation (MLE):

$$\hat{\boldsymbol{\theta}} \in \arg\max_{\theta} p(\mathbf{X} \mid \boldsymbol{\theta})$$

- Like before, MLE will be $\theta_c = \frac{n_c}{n}$ (the portion of cs in the data)
- Like before, derivation is more complicated than the result

Derivation of the MLE that doesn't work

- The likelihood is
$$p(\mathbf{X} \mid \boldsymbol{\theta}) = \theta_1^{n_1} \cdots \theta_k^{n_k}$$
- So, the log-likelihood is
$$\log p(\mathbf{X} \mid \boldsymbol{\theta}) = n_1 \log \theta_1 + \cdots + n_k \log \theta_k$$
- Take the derivative for a particular θ_c:
$$\frac{\partial}{\partial \theta_c} \log p(\mathbf{X} \mid \boldsymbol{\theta}) = \frac{n_c}{\theta_c}$$
- Set the derivative to zero:
$$\frac{n_c}{\theta_c} = 0$$
- . . . huh?

Fixing the derivation

- Setting the derivative to zero doesn't work
  - Ignores the constraint that $\sum_c \theta_c = 1$
- Some ways to enforce constraints (see e.g. this StackExchange thread):
  - Use Lagrange multipliers to find stationary point of the Lagrangian
  - Define $\theta_k = 1 - \sum_{c=1}^{k-1} \theta_c$, replace in the objective function
- We'll take a different way here:
  - Use a different parameterization $\tilde{\theta}_c$ that doesn't have this constraint
  - Compute the MLE for the $\tilde{\theta}_c$ by setting derivative to zero
  - Convert from the $\tilde{\theta}_c$ to θ_c

Unnormalized parameterization

- Let's have $\tilde{\theta}_c$ be unnormalized:

$$\Pr(X = c \mid \tilde{\theta}_1, \ldots, \tilde{\theta}_k) \propto \tilde{\theta}_c$$

  - Still need each $\tilde{\theta}_c \geq 0$
- Can then find

$$p(c \mid \tilde{\boldsymbol{\theta}}) = \frac{\tilde{\theta}_c}{\sum_{i=1}^{k} \tilde{\theta}_c} = \frac{\tilde{\theta}_c}{Z_{\tilde{\boldsymbol{\theta}}}}$$

- The "normalizing constant" $Z_{\tilde{\boldsymbol{\theta}}}$ makes the total probability 1
  - Don't need the explicit sum-to-1 constraint anymore
  - Note: constant for different x; **not** constant for different $\boldsymbol{\theta}$
- To convert from unnormalized to normalized: $\theta_c = \tilde{\theta}_c / Z_{\tilde{\boldsymbol{\theta}}}$

Derivation of the MLE that does work

- The likelihood in terms of the unnormalized parameters is

$$p(\mathbf{X} \mid \tilde{\boldsymbol{\theta}}) = \left(\frac{\tilde{\theta}_1}{Z_{\tilde{\boldsymbol{\theta}}}}\right)^{n_1} \cdots \left(\frac{\tilde{\theta}_k}{Z_{\tilde{\boldsymbol{\theta}}}}\right)^{n_k} = \frac{1}{Z_{\tilde{\boldsymbol{\theta}}}^n} \tilde{\theta}_1^{n_1} \cdots \tilde{\theta}_k^{n_k}$$

- So, the log-likelihood is

$$\log p(\mathbf{X} \mid \tilde{\boldsymbol{\theta}}) = n_1 \log \tilde{\theta}_1 + \cdots + n_k \log \tilde{\theta}_k - n \log Z_{\tilde{\boldsymbol{\theta}}}$$

- Take the derivative for a particular $\tilde{\theta}_c$:

$$\frac{\partial}{\partial \tilde{\theta}_c} \log p(\mathbf{X} \mid \tilde{\boldsymbol{\theta}}) = \frac{n_c}{\tilde{\theta}_c} - \frac{n}{Z_{\tilde{\boldsymbol{\theta}}}} \frac{\partial Z_{\tilde{\boldsymbol{\theta}}}}{\partial \tilde{\boldsymbol{\theta}}_c} = \frac{n_c}{\tilde{\theta}_c} - \frac{n}{Z_{\tilde{\boldsymbol{\theta}}}} \qquad \text{since } \frac{\partial}{\partial \tilde{\theta}_c}\left(\tilde{\theta}_1 + \cdots + \tilde{\theta}_k\right) = 1$$

- Set the derivative to zero:

$$\frac{n_c}{\tilde{\theta}_c} = \frac{n}{Z_{\tilde{\boldsymbol{\theta}}}} \quad \text{so} \quad \frac{\tilde{\theta}_c}{Z_{\tilde{\boldsymbol{\theta}}}} = \frac{n_c}{n}$$

  - Can check this objective is concave, so this is a max; also satisfies $\tilde{\theta}_c \geq 0$ constraint
  - Many solutions, but all the same after normalizing

MAP estimate, Dirichlet prior

- As before, might prefer MAP estimate over MLE
- Often becomes more important for large k: lots of parameters!

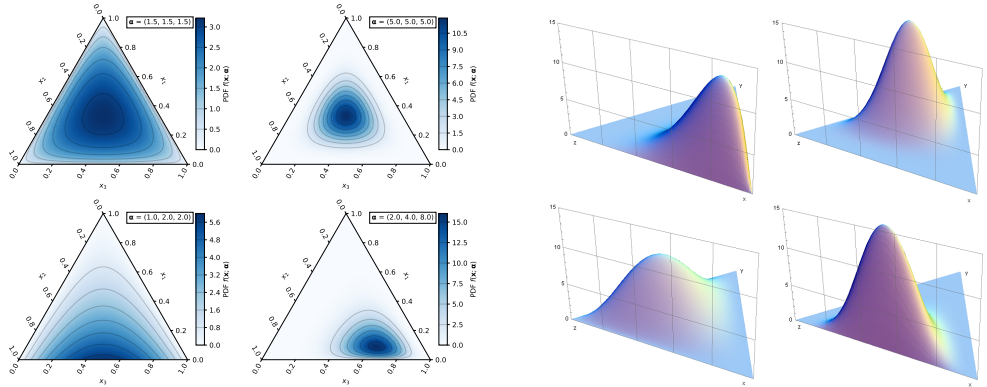
- Most common prior is the Dirichlet distribution:

$$p(\theta_1, \ldots, \theta_k \mid \alpha_1, \ldots, \alpha_k) \propto \theta_1^{\alpha_1 - 1} \cdots \theta_k^{\alpha_k - 1}$$

  - Generalization of the beta distribution to k classes
  - Requires each $\alpha_c > 0$
- This is a distribution over $\boldsymbol{\theta}$
  - Probability distribution over possible (categorical) probability distributions

Dirichlet distribution

- Wikipedia's visualizations for $k = 3$:



https://en.wikipedia.org/wiki/Dirichlet_distribution

MAP estimate for Dirichlet-Categorical

- Reason to use the Dirichlet: again because posterior is simple

$$p(\boldsymbol{\theta} \mid \mathbf{X}, \boldsymbol{\alpha}) \propto p(\mathbf{X} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \boldsymbol{\alpha}) \propto \theta_1^{n_1} \cdots \theta_k^{n_k} \; \theta_1^{\alpha_1 - 1} \cdots \theta_k^{\alpha_k - 1}$$
$$= \theta_1^{(n_1 + \alpha_1) - 1} \cdots \theta_k^{(n_k + \alpha_k) - 1}$$

  i.e. it's Dirichlet again with parameters $\tilde{\alpha}_c = n_c + \alpha_c$

- A few more steps show MAP for a categorical with Dirichlet prior is

$$\hat{\theta}_c = \frac{n_c + \alpha_c - 1}{\sum_{c'=1}^{k} (n_{c'} + \alpha_{c'} - 1)}$$

  (again as long as all $n_c + \alpha_c > 1$)

- Dirichlet has k hyper-parameters α_c
  - Often use $\alpha_c = \alpha$ for some $\alpha \in \mathbb{R}$: one hyperparameter
  - Makes the MLE $\hat{\theta}_c = \dfrac{n_c + \alpha - 1}{n + k(\alpha - 1)}$
  - $\alpha = 2$ gives Laplace smoothing (add 1 "fake" count for each class)

Conjugate priors

- This is our second example where prior and posterior have the same form
  - Beta prior + Bernoulli likelihood gives a Beta posterior
    - Also happens with binomial, geometric, . . . likelihoods
  - Dirichlet prior + categorical likelihood gives a Dirichlet posterior
    - Also happens with multinomial likelihood

- When this happens, we say prior is conjugate to the likelihood
- Prior and posterior come from the same "family" of distributions

$$X \sim L(\theta) \quad \theta \sim P(\alpha) \qquad \text{implies} \quad \theta \mid X \sim P(\alpha^+)$$

  - Updated parameters α^+ will depend on the data

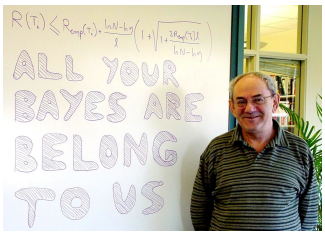
- Many computations become easier if we have a conjugate prior
- But not all distributions have conjugate priors
  - And even when one exists, might not be convenient / a good choice

Outline

Discriminative classifiers

- Generative classifiers model $p(x, y)$, then use that to get $p(y \mid x)$
  - Often model $p(y)$ (usually simple) and then $p(x \mid y)$ (harder)

> "When solving a problem of interest, do not solve a more general problem as an intermediate step."
>
> — *Vladimir Vapnik*



- An alternative philosophy: just directly model $p(y \mid x)$
  - Or even further: just directly learn a classification function
- Modeling $p(x)$ can be hard
  - Discriminative: "which pixels show me this picture is a dog?"
  - Generative: "what do pictures of dogs look like?"

Hierarchy of predictor types

- Different types of models can answer different types of questions:

type	example	$p(x, y)$	$p(y \mid x)$	$f(x) \approx y$
Generative	naïve Bayes	✓	✓	✓
Discriminative (prob.)	logistic regression	✗	✓	✓
Discriminative (non-prob.)	SVM	✗	✗	✓

- Problem usually gets "easier" as you model less
- But you can't do as much with it
  - Discriminative models can't sample, do outlier detection, ...
  - "Pure classifiers" can't easily combine into broader inference (e.g. decision theory)

Discriminative models, binary data

- Discriminative model with a full categorical parameterization:

$$\Pr(\text{spam} \mid \texttt{aardvark} = 0, \ldots, \texttt{lotto} = 0, \ldots, \texttt{zyzzyva} = 0) = \theta_{0 \cdots 0 \cdots 0}$$

$$\vdots$$

$$\Pr(\text{spam} \mid \texttt{aardvark} = 1, \ldots, \texttt{lotto} = 1, \ldots, \texttt{zyzzyva} = 1) = \theta_{1 \cdots 1 \cdots 1}$$

- Can represent **any** conditional distribution on binary data
- Needs 2^d parameters (versus $2(2^d - 1)$ for "galaxy brain Bayes")
  - (Why not $2^d - 1$?)

- Fitting: $y \mid x$ is a separate Bernoulli for each x; can just MLE/MAP for each one
- But probably don't see very many emails per x (and many have $n_x = 0$)
  - Will probably overfit for almost every x
  - Want to share information across similar xs!

Linear parameterization of conditionals

- Generally: would like to use a "parsimonious" parameterization
  - Full categorical distribution: can model anything, very many parameters
  - Making stronger assumptions: can't model everything, much less complex model

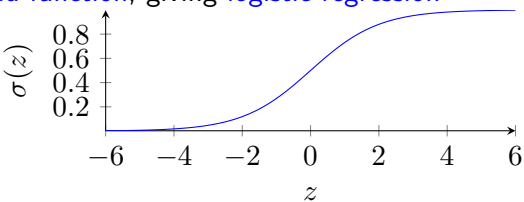
- Standard basic choice: assume a linear model, i.e. one of the form

$$p(y = 1 \mid x_1, \ldots, x_d, w) = f(w_1 x_1 + \cdots + w_d x_d) = f(w^\mathsf{T} x)$$

  where w is our vector of d parameters and f is some function from $\mathbb{R}$ to $[0, 1]$

- Standard basic choice for f: sigmoid function, giving logistic regression

$$f(z) = \frac{1}{1 + \exp(-z)}$$



Logistic regression inference

- For a given w and x, logistic regression gives us a Bernoulli distribution over y:

$$\Pr(Y = 1 \mid X = x, w) = \frac{1}{1 + \exp(-w^{\mathsf{T}} x)}$$

- Usually just take the mode to predict most likely y
- But can also:
  - Set a different confidence threshold, e.g. based on "decision theory"
  - Sample conditional ys given this x
  - Compute probability of seeing 5 positives out of 10 examples with this x
  - Compute the expected number of samples with this x to see a single positive
  - Ask how likely *both* an x and an independent x' are to be positive
  - ...

Maximum conditional likelihood

- MLE for generative models: $\arg\max_w p(\mathbf{X}, \mathbf{y} \mid w)$
  - Can't do that for discriminative models!
- When we say MLE for discriminative models, we mean $\arg\max_w p(\mathbf{y} \mid \mathbf{X}, w)$
  - Treat $\mathbf{X}$ as fixed, maximize conditional likelihood

- Logistic regression also makes sense for continuous x
  - Even though it's only using binary probabilities!
- Different than naïve Bayes:
  - Models $X \mid Y$, so continuous X needs to use a continuous distribution

Logistic (negative log-)likelihood

- Logistic regression uses

$$p(\mathbf{y} \mid \mathbf{X}, w) = \prod_{i=1}^{n} p\left(y^{(i)} \mid \mathbf{X}, w\right) = \prod_{i=1}^{n} p\left(y^{(i)} \mid x^{(i)}, w\right)$$

  so $-\log p(\mathbf{y} \mid \mathbf{X}, w) = \sum_{i=1}^{n} -\log p(y^{(i)} \mid x(i), w)$
- Each $-\log p(y^{(i)} \mid x(i), w)$ term is $\log\left(1 + \exp\left(-\tilde{y}^{(i)} w^{\mathsf{T}} x^{(i)}\right)\right)$, for $\tilde{y} \in \{-1, 1\}$:

$$\begin{cases} -\log \frac{1}{1+\exp\left(-w^{\mathsf{T}} x^{(i)}\right)} & \text{if } y^{(i)} = 1 \\ -\log\left(1 - \frac{1}{1+\exp\left(-w^{\mathsf{T}} x^{(i)}\right)}\right) & \text{if } y^{(i)} = 0 \end{cases} = \begin{cases} \log\left(1 + \exp\left(-w^{\mathsf{T}} x^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ \log\left(1 + \exp\left(w^{\mathsf{T}} x^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

  - Usually convenient to use $y \in \{-1, 1\}$ instead of $\{0, 1\}$ for binary linear classifiers

MLE for logistic regression

review

- MLE is equivalent to minimizing $f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y^{(i)} w^{\mathsf{T}} x^{(i)}))$
  - Using $y^{(i)} \in \{-1, 1\}$ here
  - Equivalent to "binary cross-entropy"
  - Computational cost: need to compute the $w^{\mathsf{T}} x^{(i)}$, aka $\mathbf{X}w$, in time $\mathcal{O}(nd)$
  - $\nabla f(w) = -\mathbf{X}^{\mathsf{T}} \frac{\mathbf{y}}{1+\exp(\mathbf{y} \odot \mathbf{X}w)}$, with elementwise operations for the y; also $\mathcal{O}(nd)$

- Convex function: no bad local minima
- No closed-form solution in general from setting $\nabla f(w) = 0$
- But can solve with gradient descent or other iterative optimization algorithms
  - Best choice depends on n, d, desired accuracy, computational setup, ...

MAP for logistic regression $\approx$ regularization

review

- MAP with a Gaussian prior, $w_j \sim \mathcal{N}\left(0, \frac{1}{\lambda}\right)$, adds $\frac{1}{2}\lambda\|w\|^2$ to the objective
  - Now "strongly convex": optimization is usually faster
- Typically gives better test error when λ is appropriate

- MAP here is $\arg\max_w p(w \mid \mathbf{X}, \mathbf{y}) = \arg\max_w p(\mathbf{y} \mid \mathbf{X}, w)p(w)$
  - As opposed to generative MAP, $\arg\max_w p(w \mid \mathbf{X}, \mathbf{y}) = \arg\max_w p(\mathbf{X}, \mathbf{y} \mid w)p(w)$

Binary naïve Bayes is a linear model

bonus!

$$
\begin{aligned}
\Pr(Y = 1 \mid X = x) &= \frac{p(x \mid y = 1)p(y = 1)}{p(x \mid y = 1)p(y = 1) + p(x \mid y = 0)p(y = 0)} \\
&= \frac{1}{1 + \frac{p(x|y=0)p(y=0)}{p(x|y=1)p(y=1)}} = \frac{1}{1 + \exp\left(-\log \frac{p(x|y=1)p(y=1)}{p(x|y=0)p(y=0)}\right)} \\
&= \sigma\left(\sum_{j=1}^{d} \log \frac{p(x_j \mid y = 1)}{p(x_j \mid y = 0)} + \log \frac{p(y = 1)}{p(y = 0)}\right) \\
&= \sigma\left(\sum_{j=1}^{d} \log \frac{\theta_{j|1}^{x_j}(1 - \theta_{j|1})^{1-x_j}}{\theta_{j|0}^{x_j}(1 - \theta_{j|0})^{1-x_j}} + \log \frac{p(y = 1)}{p(y = 0)}\right) \\
&= \sigma\left(\sum_{j=1}^{d} \left[x_j \log \frac{\theta_{j|1}}{\theta_{j|0}} + (1 - x_j) \log \frac{1 - \theta_{j|1}}{1 - \theta_{j|0}}\right] + \log \frac{p(y = 1)}{p(y = 0)}\right) \\
&= \sigma\left(\sum_{j=1}^{d} x_j \underbrace{\log \frac{\theta_{j|1}}{\theta_{j|0}} \frac{1 - \theta_{j|0}}{1 - \theta_{j|1}}}_{w_j} + \underbrace{\sum_{j=1}^{d} \log \frac{1 - \theta_{j|1}}{1 - \theta_{j|0}} + \log \frac{p(y = 1)}{p(y = 0)}}_{b}\right) = \sigma(w^\top x + b)
\end{aligned}
$$

Not generally the parameters that logistic regression would pick (so, lower likelihoods in logreg model)

Adding intercepts to linear models

review

- Often we only talk about homogeneous linear models, $f(w^\mathsf{T} x)$
- More generally inhomogeneous models, $f(w^\mathsf{T} x + b)$, are very useful in practice

- Two usual ways to do this:
  - Treat b as another parameter to fit and put it in all the equations
  - Add a "dummy feature" $X_0 = 1$; then corresponding weight w_0 acts like b

- Both of these ways make sense in probabilistic framing, too!
- Just be careful about if you want to use the same prior on b/w_0 or not
  - Often makes sense to "not care about y location," i.e. use improper prior $p(w_0) \propto 1$
- Another generally-reasonable scheme:
  - First centre the ys so $\frac{1}{n}\sum_{i=1}^{n} y^{(i)} = 0$, then put some prior on w_0 not being too big

Recap: tabular versus logistic regression

- Tabular parameterization of a categorical:
  - Each $\theta_{y|x}$ is totally separate
  - 2^d parameters when everything is binary
  - Can model any binary conditional parameter
  - Tends to overfit unless $2^d \ll n$
- Logistic regression parameterization of a categorical:
  - Each $\theta_{y|x}$ is given by $\sigma(w^\mathsf{T} x + b)$
  - d or $d + 1$ parameters (depending on offset)
  - Can only model linear conditionals
  - Tends to underfit unless d is big or truth is linear
- Simple versus complex model: subject of learning theory

"Fundamental trade-off"

review

- Tabular and logistic models on different sides of the "fundamental trade-off":

$$\text{generalization error} = \text{train error} + \underbrace{\text{generalization error - train error}}_{\text{generalization gap (overfitting)}} \geq \text{irreducible error}$$

- If irreducible error > 0, small train error implies some overfitting / vice versa
- Simple models, like logistic regression with few features:
  - Tend to have small generalization gaps: don't overfit much
  - Tend to have larger training error (can't fit data very well)
- Complex models, like tabular conditionals with many features:
  - Tend to have small training error (fit data very well)
  - Tend to overfit more

Nonlinear feature transformations

review

- Can go between linear and tabular with non-linear feature transforms:
  - Transform each $x^{(i)}$ into some new $z^{(i)}$
  - Train a logistic regression model on $z^{(i)}$
  - At test time, do the same transformation for the test features
- Examples: polynomial features, radial basis functions, periodic basis functions, ...
- Can also frame kernel methods in this way

- More complex features tend to decrease training error, increase overfitting
  - Performance is better if the features match the "true" conditionals better!

- Gaussian RBF features/Gaussian kernels, with appropriate regularization (λ and lengthscale σ chosen on a validation set), is often an excellent baseline

Learning nonlinear feature transformations with deep networks

review

- Not always clear which feature transformations are "right"
- Generally, deep learning tries to learn good features
  - Use "parameterized" features, optimize those parameters too
  - Use a flexible-enough class of features
- Assuming you've seen fully-connected networks: one-layer version is

$$\hat{y}(x) = v^{\mathsf{T}} h(Wx)$$

where W is an $m \times d$ matrix (the "first layer" of feature transformation)
h is an element-wise activation function, e.g. $\mathrm{ReLU}(z) = \max\{0, z\}$ or sigmoid,
v is a linear function of "activations"

- Without h (e.g. $h(z) = z$), becomes a linear model: $v^{\mathsf{T}}(Wx) = \underbrace{v^{\mathsf{T}} W}_{1 \times m} x$
- Need to fit parameters W and v

Fitting neural networks

review

- $\hat{y}(x) = v^{\mathsf{T}} h(Wx)$: with fixed W, this is a linear model in the transformed features
- For binary classification, often use logistic likelihood

$$p(y \mid x, W, v) = \sigma\left(y\, \hat{y}(x)\right)$$

- Can then compute logistic negative log-likelihood
- Minimize it with some variant of gradient descent

- Deep networks do the same thing; a fully-connected L-layer network looks like

$$\hat{y}(x) = W_L h_{L-1}(W_{L-1} h_{L-2}(W_{L-2} \cdots h_1(W_1 x) \cdots))$$

or more often, add bias terms

$$\hat{y}(x) = b_L + W_L h_{L-1}(b_{L-1} + W_{L-1} h_{L-2}(b_{L-2} + \cdots h_1(b_1 + W_1 x) \cdots))$$

where each b is a vector with the same dimension as the activations at that layer
  - If W_j is $d_j \times d_{j-1}$, jth layer activations are length d_j, b_j is also length d_j
- Can still apply same logistic likelihood, optimize in same way

Summary

- Discriminative classifiers model $p(y \mid x)$ instead of $p(x, y)$
  - Most of modern ML uses discriminative classifiers
- Tabular parameterization models all possible conditionals
- Parameterized conditionals add some structure
  - Linear models, like logistic regression, or deep models
- "Fundamental trade-off" between fitting and overfitting

- Next time: everything is regularization