Discriminative models

CPSC 440/550: Advanced Machine Learning

`cs.ubc.ca/~dsuth/440/23w2`

University of British Columbia, on unceded Musqueam land

2023-24 Winter Term 2 (Jan–Apr 2024)

Admin

admin

- Online logistics:
  - Recording will go in a different place (posted on Piazza)
  - Your voice will be recorded if you speak up, but not your video
  - Would really appreciate if at least a few people have video on

- If you need a form signed, post it (privately) on Piazza

- Tutorials are going; one online right after class (and another Friday)
- Office hours too (two Thursday, one Friday)

- Assignment 1 due Friday 5pm!
- Scheduling for quiz 1 announced later this week

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

"Full" 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 tabular distribution:

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

  …and the same for probabilities given $Y = 1$
- 2^d possible binary vectors, so need $2^d - 1$ parameters for each condition
- MLE is counting, $\theta_{x|y} = n_{x|y}/n_y$; will discuss this + priors later (categorical dist.)
- Different kind of "naïvety" than naïve Bayes: each bit-vector is totally separate

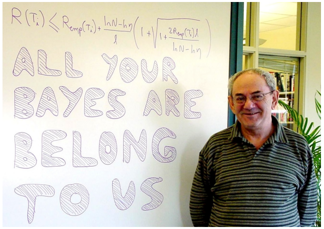
Outline

Discriminative classifiers

- Generative classifiers model $p(x, y)$, then use that to get $p(y \mid x)$

"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 tabular parameterization:

$$\Pr(\mathsf{spam} \mid \mathtt{aardvark} = 0, \ldots, \mathtt{lotto} = 0, \ldots, \mathtt{zyzzyva} = 0) = \theta_{0 \ldots 0 \ldots 0}$$

$$\vdots$$

$$\Pr(\mathsf{spam} \mid \mathtt{aardvark} = 1, \ldots, \mathtt{lotto} = 1, \ldots, \mathtt{zyzzyva} = 1) = \theta_{1 \ldots 1 \ldots 1}$$

- Can represent **any** conditional distribution on binary data
- Needs 2^d parameters (versus $2(2^d - 1)$ for "tabular 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

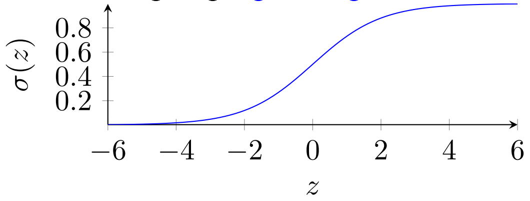
- Generally: would like to use a "parsimonious" parameterization
  - Full tabular distribution: can model anything, very many parameters
  - Making stronger assumptions: can't model everything, much less complex model

- 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!

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

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:
  - 2^d parameters
  - Can model any binary conditional parameter
  - Tends to overfit unless $2^d \ll n$
- Logistic regression:
  - d parameters (or $d + 1$ with 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

$$\hat{y}(x) = v^\mathsf{T} h_{L-1}(W_{L-1} h_{L-2}(W_{L-2} \cdots h_1(W_1 x) \cdots))$$

  or more often, add bias terms

$$\hat{y}(x) = \beta + v^\mathsf{T} h_{L-1}(b_{L-1} + W_{L-1} h_{L-2}(b_{L-2} + W_{L-2} \cdots h_1(b_1 + W_1 x) \cdots))$$

  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

Universal approximation

- For most activation functions, wide networks are universal approximators
  - Even if they only have one hidden layer
- Any continuous function on bounded domain can be approximated arbitrarily well

- But this is in a non-parametric regime:
  - The width of the hidden layer needs to grow with n
  - Any fixed-size network is not a universal approximator
- Other universal approximators:
  - k-nearest neighbours (if k grows with n)
  - Logistic regression with polynomial features – if degree grows with n
  - Linear models with Gaussian RBF features (with one basis per $x^{(i)}$)
  - Linear models with a Gaussian kernel
  - Fixed-width but growing-depth networks

Is training neural nets scary?

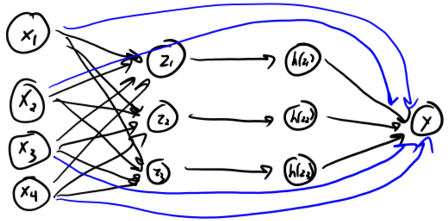
- The objective function is highly non-convex, even for one hidden layer
- Finding the global optimum is generally NP-hard
- Nearly always trained with variants of stochastic gradient descent (SGD)

$$W^{(k+1)} = W^{(k)} - \alpha_k \nabla_W \Big[-\log p^{(} y^{(i)} \mid x^{(i)}, W^{(k)}, v^{(k)} \Big] \; v^{(k+1)} = v^{(k)} - \alpha_k \nabla_v \Big[-\log p^{(} y^{(i}$$

  - Lots of variants: minibatches, different versions of momentum, Adam, ...
  - SGD not guaranteed to reach a global optimum for non-convex problems

Can ensure neural networks $\geq$ logistic regression

- Consider a neural network with one hidden layer and connections from input to output layer.
  - The extra connections are called "skip" connections.



$$\hat{y} = \underbrace{w^T x}_{\text{linear model}} + \underbrace{v^T h(Wx)}_{\text{neural network}}$$

- You could first set v=0, then optimize w using logistic regression.
  - This is a convex optimization problem that gives you the logistic regression model.
- You could then set W and v to small random values, and start SGD from the logistic regression model.
  - Even though this is non-convex, the neural network can only improve on logistic regression (improves "residual" error).
- And if you are worried about overfitting, you could stop SGD by checking performance on validation set.
  - This is called regularization by "early stopping".
- In practice, we typically optimize everything at once (which usually works better than the above).

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
- Linear models, especially logistic regression, simplify things
- "Fundamental trade-off" between fitting and overfitting
- Fully connected neural networks

- Next time: everything is regularization