Discriminative models and deep learning

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)

Last time

- Generative classifiers, e.g. Naive Bayes:
  - Model $p(x, y)$, typically with $p(y)$ and $p(x \mid y)$
  - Use that to model $p(y \mid x)$
  - Use that to make decisions
- Discriminative (probabilistic) classifiers, e.g. logistic regression:
  - Model $p(y \mid x)$ directly
    - $p(x)$ or $p(x \mid y)$ is often much harder to model correctly!
    - But if we don't model it, can't use it (e.g. outlier detection, sampling, ...)
  - Use that to make decisions
- Discriminative non-probabilistic classifiers, e.g. SVMs:
  - Learn a decision function directly
    - Don't need to try to model $p(y \mid x)$
    - But if we don't model it, can't use it (e.g. "decision theory")

Generative classifiers, usual framework

- Can generalize our previous notion of Naive Bayes to categorical data:

- $Y \sim \mathrm{Cat}(\boldsymbol{\theta}_y)$ e.g. $\begin{aligned} \Pr(Y = \texttt{important}) &= 0.1 \\ \Pr(Y = \texttt{promo}) &= 0.3 \\ \Pr(Y = \texttt{spam}) &= 0.4 \\ \Pr(Y = \texttt{other}) &= 0.2 \end{aligned}$

- $X_j \mid (Y = y) \sim \mathrm{Bern}(\theta_{j|y})$ e.g. $\Pr(\text{“ASAP”} \in \text{email} \mid Y = \texttt{important}) = 0.05$

- $p(\texttt{important} \mid x) = p(x \mid \texttt{important})p(\texttt{important}) / \sum_y p(x \mid y)p(y)$

- Can fit all the parameters $\Theta = \{\boldsymbol{\theta}_y, \theta_{1|1}, \dots\}$ with MLE: $\arg\max_\Theta p(\mathbf{X}, \mathbf{y} \mid \Theta)$
- Or put prior $p(\Theta)$, use MAP: $\arg\max_\Theta p(\Theta \mid \mathbf{X}, \mathbf{y}) = \arg\max_\Theta p(\mathbf{X}, \mathbf{y} \mid \Theta)p(\Theta)$
  - e.g. Dirichlet prior for $\boldsymbol{\theta}_y$, Beta for all the $\theta_{j|y}$

- Can use any other distributions for Y and $X \mid Y = y$ in the same way

Multi-class naïve Bayes on MNIST

- Binarized MNIST: label is categorical, but images are still product of Bernoullis
- Parameter of the Bernoulli for each class:

- One sample from each class:

Discriminative, probabilistic, binary classifiers

- Model $Y \mid (X = x) \sim \mathrm{Bern}(\theta_x)$
- Can do "discriminative" MLE/MAP/... for θ_x: $\arg\max_\Theta p(\mathbf{y} \mid \mathbf{X}, \Theta) p(\Theta)$

- One extreme ("galaxy brain"): each θ_x is a totally separate parameter
  - Can model absolutely anything, with enough data
  - You probably don't have enough data

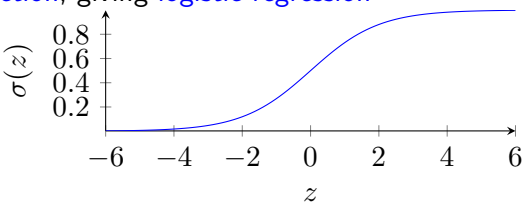
- Other extreme: each θ_x is the same
  - You probably have enough data to fit this well!
  - But it totally ignores x and makes the same decision for everything

- Almost always want an in-between: "similar x should have similar θ_x"
- ... but what does "similar" mean?
- Common choice: $\theta_x = \Pr(Y = 1 \mid X = x)$ given by some function $\hat{\theta}(x)$
- Can choose $\hat{\theta}(x)$ by MLE or MAP: $\arg\max_f p(\mathbf{y} \mid \mathbf{X}, \hat{\theta}) p(\hat{\theta})$

Logistic regression

- Linear models: $\theta_x = \Pr(Y = 1 \mid X = x) = \sigma(w \cdot x)$
- Defined by parameters $w \in \mathbb{R}^d$
- Common choice for σ: sigmoid function, giving logistic regression

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



Logistic (negative log-)likelihood

- Logistic regression uses

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

$$\arg\max_w p(\mathbf{y} \mid \mathbf{X}, w) = \arg\min_w -\log p(\mathbf{y} \mid \mathbf{X}, w)$$

$$= \arg\min_w \sum_{i=1}^{n} -\log p(y^{(i)} \mid x^{(i)}, w)$$

- 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

- Often we only talk about homogeneous linear models, $\sigma(w^\mathsf{T} x)$
- More generally inhomogeneous models, $\sigma(w^\mathsf{T} x + b)$, 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

Feature engineering

review

- If we're using a linear model, we want features that will make sense
- For example, how do we use categorical features x?
- Usually convert to set of binary features ("one-hot" / "one of k" encoding)

Age	City	Income
23	Van	26,000
25	Sur	67,000
19	Bur	16,500
43	Sur	183,000

$\rightarrow$

Age	Van	Bur	Sur	Income
23	1	0	0	26,000
25	0	0	1	67,000
19	0	1	0	16,500
43	0	0	1	183,000

- If you see a new category in test data: usually, just set *all* of them to zero

- Also often want to standardize features: subtract mean, divide by variance
- May or may not want to do this for one-hots

Recap: tabular versus logistic regression

- Tabular parameterization ("galaxy brain"):
  - Each θ_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 θ_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
- Totally naive parameterization of a categorical:
  - Each θ_x is equal to a single shared θ
  - One parameter
  - Can't model any non-constant effect
  - Underfits really awfully unless there's really just no signal

"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:
  - Tend to have small generalization gaps: don't overfit much
  - Tend to have larger training error (can't fit data very well)
- Complex models:
  - Tend to have small training error (fit data very well)
  - Tend to overfit more

Nonlinear feature transformations

review

- Linear models can have different complexities 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
- Fully-connected networks: one-hidden-layer, 1d output version is

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

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

- $f(x) = v^{\mathsf{T}} h(Wx)$: with fixed W, this is a linear model in the transformed features
- Can then plug this in to $\hat{\theta}(x) = \sigma(f(x))$ for binary classification
- 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

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

or more often, add bias terms

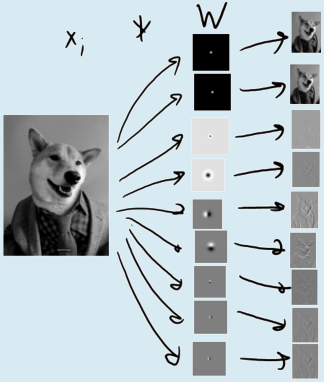
$$f(x) = h_L(b_L + W_L h_{L-1}(b_{L-1} + W_{L-1} h_{L-2}(b_{L-2} + \cdots h_1(b_1 + W_1 x) \cdots)))$$

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

Convolutional networks

review

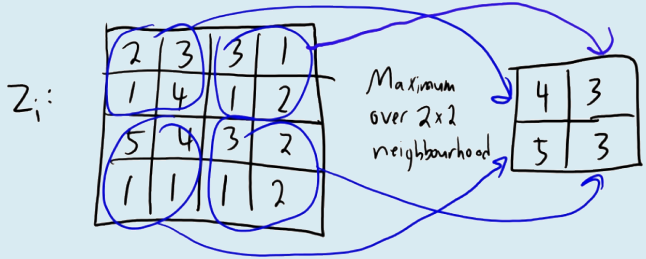
- Different architectures make different implicit assumptions about the structure of how θ_x changes with x
- Convolutional layers: restrict form of W to act like a bank of convolutions



Convolutional networks

review

- Different architectures make different implicit assumptions about the structure of how θ_x changes with x
- Convolutional layers: restrict form of W to act like a bank of convolutions
- Pooling layers: no-parameter ways to decrease hidden dim / enforce invariances



Max pooling: "there's an edge around here, I don't care exactly where"
Average pooling: "most of these patches look like they're part of an airplane"

Convolutional networks

review

- Different architectures make different implicit assumptions about the structure of how θ_x changes with x
- Convolutional layers: restrict form of W to act like a bank of convolutions
- Pooling layers: no-parameter ways to decrease hidden dim / enforce invariances
- Traditional architectures end by flattening and feeding into fully-connected layers

- Usual convolutions are 2-dimensional on images
- But they make sense whenever there's a notion of neighbourhood
  - 1d convolution on sequences (time series, sentences, ...)
  - Graph convolutional networks (will explore on A2)

Skip connections

review

- Standard fully-connected layer:

$$f_j(x) = h_j(b_j + W_j f_{j-1}(x))$$

- One form of skip connection:

$$f_j(x) = h_j(b_j + W_j f_{j-1}(x) + W_{j-2\to j} f_{j-2}(x))$$

- Residual connections (building blocks of ResNets) use a special form:

$$f_{2j}(x) = h_{2j}\left(b_{2j} + W_{2j-1\to 2j} h_{2j-1}(W_{2j-1} f_{2j-2}(x)) + f_{2j-2}(x)\right)$$

- DenseNets look at **everything** before:

$$f_j(x) = h_j\left(b_j + \sum_{\ell=0}^{j-1} W_{\ell\to j} f_\ell(x)\right)$$

Multi-class classification

- All of this gives different ways to parameterize $\hat{\theta}$ in $Y \mid (X = x) \sim \text{Bern}(\hat{\theta}(x))$
- Multiclass classification: Y takes one of k possible values
  - Is this image of a gorilla, or a drill, or a Burmese mountain dog, or...
- Swap $\text{Bern}(\hat{\theta}(x))$ for $\text{Cat}(\hat{\boldsymbol{\theta}}(x))$ and everything is the same!

- How to parameterize $\hat{\boldsymbol{\theta}}(x)$? Needs to be nonnegative and sum to one
- First, make the last layer of the network output k values instead of 1
- Softmax function first makes nonnegative by taking $\exp$, then normalizes:

$$\theta_c = [\text{softmax}(\mathbf{z})]_c = \frac{\exp(z_c)}{\sum_{c'=1}^{k} \exp(z_{c'})} \propto \exp(z_c)$$

  - Don't *have* to use softmax, other options exist, but this is the default

Beyond multi-class

- This framework now allows for other data types, too!
- A1 had an example of Poisson regression:

$$Y \mid (X = x) \sim \text{Poisson}(\lambda_x) \qquad \Pr(Y = y \mid X = x) = \frac{\lambda_x^y e^{-\lambda_x}}{y!} \mathbb{1}(y \in \mathbb{N}_{\geq 0})$$

  where we used $\lambda_x = \exp(w^\mathsf{T} x)$
- Could just as easily use a deep network instead of $w^\mathsf{T} x$

- Linear regression uses $Y \mid (X = x) \sim \mathcal{N}(w^\mathsf{T} x, \sigma^2)$ for some fixed σ^2
- Could just as easily use a deep network instead of $w^\mathsf{T} x$
- Could also parameterize σ^2 as a function of $w^\mathsf{T} x$

- Very powerful framework to mix-and-match pieces together with!

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: handling continuous x