# CSC 411 Lecture 08: Generative Models for Classification

Ethan Fetaya, James Lucas and Emad Andrews

University of Toronto

# Today

- Classification – Bayes classifier
- Estimate input probability densities from data
- Naive Bayes

# Classification

- Given inputs $\mathbf{x}$ and classes $y$ we can do classification in several ways. How?

(features)

$\mathbf{x}$

e.g:

- height
- weight
- color

(class label)

$y$

**elefant**

**bear**

# Discriminative Classifiers

- **Discriminative** classifiers try to either:
  - learn mappings directly from the space of inputs $\mathcal{X}$ to class labels $\{0, 1, 2, \ldots, K\}$

(features)

$\mathbf{x}$ $\longrightarrow$ $y$

(class label)

e.g:
- height
- weight
- color

elefant

bear

For example:

Decision trees

# Discriminative Classifiers

- **Discriminative** classifiers try to either:
  - or try to learn $p(y|\mathbf{x})$ directly

(features) $\mathbf{x}$ $\longrightarrow$ (class probability) $p(y|\mathbf{x})$

e.g:
- height
- weight
- color

elefant

bear

For example:
- Logistic Regression

# Generative Classifiers

How about this approach: build a model of "what data for a class looks like"

- **Generative** classifiers try to model $p(\mathbf{x}, y)$. If we know $p(y)$ we can easily compute $p(\mathbf{x}|y)$.
- Classification via Bayes rule (thus also called Bayes classifiers)

(prob. of features given label)

$$p(\mathbf{x}|y) \longleftarrow y$$

(class label)

e.g:

- height
- weight
- color

**elefant**

**bear**

**Tries to model:**

- How does data look like for a class?

**Classification (How?)**

# Generative vs Discriminative

Two approaches to classification:

- Discriminative classifiers estimate parameters of decision boundary/class separator directly from labeled examples. Tries to solve: How do I separate the classes?
  - learn $p(y|\mathbf{x})$ directly (logistic regression models)
  - learn mappings from inputs to classes (least-squares, decision trees)
- Generative approach: model the distribution of inputs characteristic of the class (Bayes classifier). Tries to solve: What does each class "look" like?
  - Build a model of $p(\mathbf{x}|y)$
  - Apply Bayes Rule

# Bayes Classifier

- Aim to classify text into spam/not-spam (yes C=1; no C=0)
- Use bag-of-words features, get binary vector $\mathbf{x}$ for each patient
- Given features $\mathbf{x} = [x_1, x_2, \cdots, x_d]^T$ we want to compute class probabilities using Bayes Rule:

$$p(C|\mathbf{x}) = \frac{p(\mathbf{x}|C)p(C)}{p(\mathbf{x})}$$

- More formally

$$\text{posterior} = \frac{\text{Class likelihood} \times \text{prior}}{\text{Evidence}}$$

- How can we compute $p(\mathbf{x})$ for the two class case? (Do we need to?)

$$p(\mathbf{x}) = p(\mathbf{x}|C=0)p(C=0) + p(\mathbf{x}|C=1)p(C=1)$$

- To compute $p(C|\mathbf{x})$ we need: $p(\mathbf{x}|C)$ and $p(C)$

# Classification: Simple Example

- Let's start with a simple (but slightly redundant) example.
- Imagine that we have some biased coins and we observe a single outcome from one of these coins.
- We have $P(x|C) = Ber(\theta_C) = \theta_C^x \cdot (1 - \theta_C)^{1-x}$
- Notice that we have different parameters for each c oin
- How can I fit the distribution to my data?
- Simple approach - maximum likelihood

# MLE for Bernoulli

- Assumption: data points are independent and identically distributed (i.i.d)

$$p(\mathcal{D}_C|C) = \prod_{n=1}^{N} p(x^{(n)}|C) = \prod_{n=1}^{N} \theta_C^{x^{(n)}} \cdot (1-\theta_C)^{1-x^{(n)}} = \theta_C^{N_C} \cdot (1-\theta_C)^{N-N_C}$$

- We define $N_C = \sum_{i=1}^{N} x^{(n)}$ the number of ones (heads) seen.
- $N$ and $N_C$ are called sufficient statistics - hold all the information we need to compute $P(\mathcal{D}_C|C)$
- We can minimize the negative log-likelihood (NLL)

$$\ell_{log-loss} = -\log(p(x^{(1)}, \cdots, x^{(N)}|C)) = -N_C \log(\theta_C) - (N - N_C)\log(1-\theta_C)$$

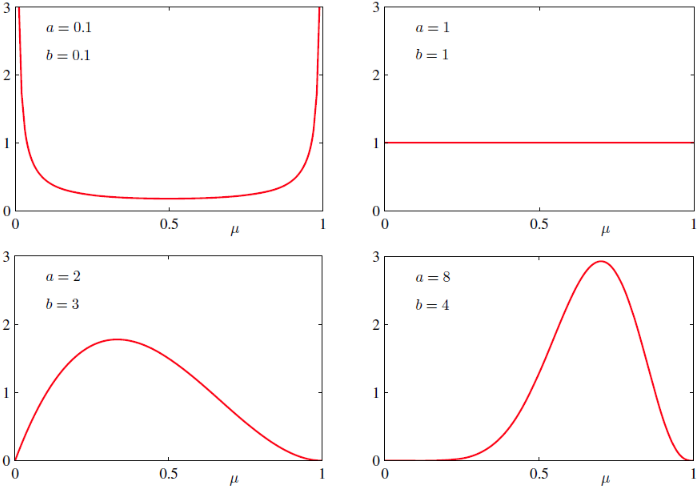
$$\frac{\partial \ell_{log-loss}}{\partial \theta_C} = -\frac{N_C}{\theta_C} + \frac{N-N_C}{1-\theta_C} = 0 \Rightarrow \theta_C = \frac{N_C}{N}$$

# Beta-Binomial

- MLE solution $\theta_C = \frac{N_C}{N}$. What if $N_C = 0$?
- Example: Some rare word unseen in a training corpus.
- In that case $P(x|C) = 0$ no matter what other information we have!
- Solution: A prior over $\theta$.
- Simple (conjugate) prior: Beta distribution
  - $Beta(\theta|a, b) \propto \theta^{a-1}(1-\theta)^{b-1}$

# Beta Distribution

- Examples of $Beta(\theta|a, b) \propto \theta^{a-1}(1-\theta)^{b-1}$:




[Image credit: Bishop]

# Beta-Binomial

- Likelihood $p(\mathcal{D}_C|\theta_C) = \theta_C^{N_C} \cdot (1-\theta_C)^{N-N_C}$
- Prior $P(\theta_C) = Beta(\theta_C|a,b) \propto \theta_C^{a-1}(1-\theta_C)^{b-1}$

$$p(\theta_C|\mathcal{D}_C) = \frac{p(\mathcal{D}_C|\theta_C)P(\theta_C)}{p(\mathcal{D}_C)} \propto \theta_C^{N_C} \cdot (1-\theta_C)^{N-N_C}\theta_C^{a-1}(1-\theta_C)^{b-1}$$

$$= \theta_C^{N_C+a-1} \cdot (1-\theta_C)^{N-N_C+b-1}$$

- We have $P(\theta_C|\mathcal{D}_C) = Beta(N_C+a, N-N_C+b)$
- MAP estimation $\theta_{C,map} = \frac{N_C+a-1}{N+a+b-2}$ (show!)

# Beta-Binomial *

- Can we do better then the using the MAP estimator? A more Bayesian approach.
- We have $P(\theta_C|\mathcal{D}_C) = Beta(N_c + a, N - N_C + b)$, what is $P(x = 1|\mathcal{D}_C)$?

$$P(x = 1|\mathcal{D}_C) = \int_0^1 P(x = 1|\theta_C)P(\theta_C|\mathcal{D}_C)$$

$$= \int_0^1 \theta_C P(\theta_C|\mathcal{D}_C) = \mathbb{E}[\theta_C|\mathcal{D}_C]$$

- Beta(a,b) has a closed form mean $\frac{a}{a+b}$ (a bit of work to show) so $\theta_C = P(x = 1|\mathcal{D}_C) = \frac{N_C+a}{N+a+b}$
- Equivalent to pseudo-counts, adding $a$ fictitious positive examples and $b$ negative ones.

# Moving beyond coins

- In the real world we tend to have a vector of observations $\mathbf{x} = [x_1, .., x_d]$.
- Modelling $p(\mathbf{x}, y)$ in this case is much more complex.

$$p(x_1, \cdots, x_d, y) = p(x_1 | x_2, \cdots, x_d, y) \cdots p(x_{d-1} | x_d, y) p(x_d, y)$$

- We need to make some assumptions!
- The Naive-Bayes Model is born from a particularly strong assumption.

# Naive-Bayes for Bernoulli variables

- Make the (naive) assumption - dimensions $\mathbf{x} = [x_1, .., x_d]$ are independent given the class $y$.

$$P(\mathbf{x}|y = C, \theta_C) = \prod_{j=1}^{d} p(x_j|y = C, \theta_{jC}) = \prod_{j=1}^{d} \theta_{jC}^{x_j}(1 - \theta_{jC})^{(1-x_j)} =$$

$$\exp\left(\sum_{j=1}^{d} x_j \log(\theta_{jC}/(1 - \theta_{jC})) + \sum_{j=1}^{d} \log(1 - \theta_{jC})\right) = \exp(\mathbf{w}_C^T\mathbf{x}+w_{0C})$$

- Define $w_{Cj} = \log(\theta_{jC}/(1 - \theta_{jC}))$, $w_{0C} = \sum_{j=1}^{d} \log(1 - \theta_{jC})$

# Naive-Bayes for Bernoulli variables

- How do we classify?

$$P(y = C|\mathbf{x}) \propto P(y = C)P(\mathbf{x}|y = C) = \exp(\mathbf{w}_C^T\mathbf{x} + b_C)$$

- $w_{Cj} = \log(\theta_{jC}/(1 - \theta_{jC}))$, $b_C = w_{0C} + \log(P(y = C))$
- Linear classifier! Model is similar to logistic regression, but different optimization.
  - No gradients - just need to count! Really fast to train.
  - Doesn't take into account correlation between features.

# Example: 20newsgroups

Table: Top word per topic

| Topic | Naive Bayes | Logistic regression |
|---|---|---|
| 'alt.atheism', | don | enlightening |
| 'comp.graphics', | thanks | needed |
| 'comp.os.ms-windows.misc', | windows | windows |
| 'comp.sys.ibm.pc.hardware', | thanks | disappointing |
| 'comp.sys.mac.hardware', | mac | mac |
| 'comp.windows.x', | window | xtvaappinitialize |
| 'misc.forsale', | sale | semd |
| 'rec.autos', | car | car |
| 'rec.motorcycles', | bike | bike |
| 'rec.sport.baseball', | year | 950k |
| 'rec.sport.hockey', | team | hockey |
| 'sci.crypt', | key | encryption |
| 'sci.electronics', | use | cci |
| 'sci.med', | don | melittin |
| 'sci.space', | space | launch |
| 'soc.religion.christian', | god | satan |
| 'talk.politics.guns', | people | gun |
| 'talk.politics.mideast', | people | kidding |
| 'talk.politics.misc', | people | paranoia |
| 'talk.religion.misc' | people | compuserve |

# Beyond Bernoulli

- We focused on binary features, $x_i$, but Naive bayes is more general.
- Discrete features - multinomial.
- Continuous features - Gaussian (or any other).
- No problem to mix (unlike logistic regression)!

# NB recap

- Learning parameters:
  - Estimate $P(y = C)$, e.g. $P(y = C) = \frac{\#\, class\ C}{\#\, data\ points}$
  - For each class $C$ and feature $x_i$ estimate the distribution $p(x_i|y = C)$
- At test time:
  - For each class compute $S_C = \log(P(Y = C)) + \sum_{i=1}^{d} \log(p(x_i|y = C))$
  - Classify according to $max_C S_C$
- Probabilities: $P(y = C|\mathbf{x}) = \frac{\exp(S_C)}{\sum_{i=1}^{L} \exp(S_i)}$

# NB recap

- Pros:
  - Really fast to train (single pass through data!).
  - Fast to test.
  - Less over-fitting, sometimes better then logistic on small data sets
  - Easy to add/remove classes
  - Can handle partial data.
- Cons:
  - When naive i.i.d assumption doesn't hold (almost always) - can perform much worse.