

STA 314: Statistical Methods for Machine Learning I

Lecture 9 - Matrix Factorization, Probabilistic Models

Chris J. Maddison

University of Toronto

Today

- Generalization of PCA: **matrix factorization**.
- Unifying the course: **probabilistic models**

- Dimensionality reduction aims to find a low-dimensional representation of the data.
- PCA projects the data onto a subspace which maximizes the projected variance, or equivalently, minimizes the reconstruction error.
- The optimal subspace is given by the top eigenvectors of the empirical covariance matrix.
- PCA gives a set of decorrelated features.

Some recommender systems in action

[←](#)
[→](#)
[https://www.amazon.ca/?ref=nav_signin&](#)

[☆](#)
[get 10](#)
[get 10](#)
[get 10](#)
[get 10](#)
[get 10](#)

[Apps](#)
[★ Bookmarks](#)
[🔌 Version Control wi...](#)
[📄 The latest Sci-Hu...](#)
[📖 Daylight Theory: S...](#)
[📄 A Guide to Creati...](#)
[🌱 How does physics...](#)
[🌱 Grilled Steak Taco...](#)
[📄 arXiv:0707.2071v2...](#)

Inspired by your browsing history [See more](#)

Your recently viewed items and featured recommendations

Inspired by your browsing history

Pixel 2 XL Case, Google Pixel 2 XL Case, Spigen Neo Hybrid - Flexible Inner TPU and Reinforced...
★★★★★ 134
CDN\$ 20.99 ✓prime

Pixel 2 XL Case, Google Pixel 2 XL Case, Spigen Thin Fit - Premium Matte Finish Coating for...
★★★★★ 143
CDN\$ 15.99 ✓prime

Google Pixel 2 XL Screen Protector (Not Glass)[2-Pack], IQ Shield LIQuidSkin Full Coverage Screen Protector for Google...
CDN\$ 27.16

Pixel 2 XL Case, Google Pixel 2 XL Case, Spigen Rugged Armor - Resilient Carbon Fiber Design...
★★★★★ 325
CDN\$ 15.99 ✓prime

VicTsing Mini DisplayPort (Thunderbolt Port Compatible) to HDMI/DVI/VGA Male to...
★★★★★ 306
CDN\$ 16.99 ✓prime

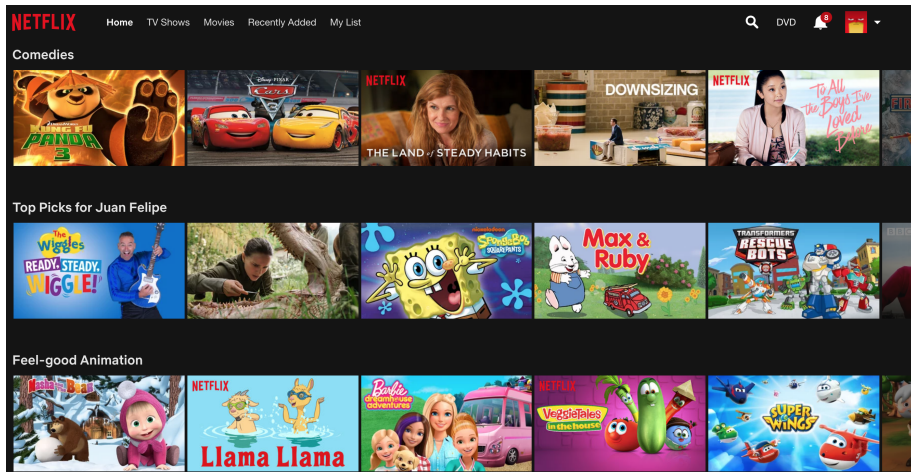
UGREEN Active Micro HDMI to HDMI VGA Video Converter Adapter with 3.5mm Audio Jack and...
★★★★★ 64
CDN\$ 25.49 ✓prime

AmazonBasics Nylon Braided USB A to Lightning Compatible Cable - Apple MFi...
★★★★★ 402
CDN\$ 12.99 ✓prime

Page 1 of 8












Ideally recommendations should combine global and seasonal interests, look at your history if available, should adapt with time, be coherent and diverse, etc.

Some recommender systems in action



The Netflix problem

Movie recommendation: Users watch movies and rate them out of 5★.

User	Movie	Rating
	Thor	★ ☆ ☆ ☆ ☆
	Chained	★ ★ ☆ ☆ ☆
	Frozen	★ ★ ★ ☆ ☆
	Chained	★ ★ ★ ★ ☆
	Bambi	★ ★ ★ ★ ★
	Titanic	★ ★ ★ ☆ ☆
	Goodfellas	★ ★ ★ ★ ★
	Dumbo	★ ★ ★ ★ ★
	Twilight	★ ★ ☆ ☆ ☆
	Frozen	★ ★ ★ ★ ★
	Tangled	★ ☆ ☆ ☆ ☆

Users only rate a few items, so would like to infer their preference for unrated items

PCA as a Matrix Factorization

- Recall PCA: project data onto a low-dimensional subspace defined by the top eigenvalues of the data covariance
- Today we consider a generalization, matrix factorizations
 - ▶ view PCA as a matrix factorization problem
 - ▶ extend to matrix completion, where the data matrix is only partially observed

PCA as Matrix Factorization

- Recall PCA: each input vector $\mathbf{x}^{(i)} \in \mathbb{R}^D$ is approximated as $\hat{\boldsymbol{\mu}} + \mathbf{U}\mathbf{z}^{(i)}$,

$$\mathbf{x}^{(i)} \approx \tilde{\mathbf{x}}^{(i)} = \hat{\boldsymbol{\mu}} + \mathbf{U}\mathbf{z}^{(i)}$$

where $\hat{\boldsymbol{\mu}} = \frac{1}{n} \sum_i \mathbf{x}^{(i)}$ is the data mean, $\mathbf{U} \in \mathbb{R}^{D \times K}$ is the orthogonal basis for the principal subspace, and $\mathbf{z}^{(i)} \in \mathbb{R}^K$ is the code vector, and $\tilde{\mathbf{x}}^{(i)} \in \mathbb{R}^D$ is $\mathbf{x}^{(i)}$'s reconstruction or approximation.

- Assume for simplicity that the data is centered: $\hat{\boldsymbol{\mu}} = 0$. Then, the approximation looks like

$$\mathbf{x}^{(i)} \approx \tilde{\mathbf{x}}^{(i)} = \mathbf{U}\mathbf{z}^{(i)}.$$

PCA as Matrix Factorization

- PCA(on centered data): input vector $\mathbf{x}^{(i)}$ is approximated as $\mathbf{U}\mathbf{z}^{(i)}$

$$\mathbf{x}^{(i)} \approx \mathbf{U}\mathbf{z}^{(i)}$$

- Write this in matrix form, we have $\mathbf{X} \approx \mathbf{Z}\mathbf{U}^\top$ where \mathbf{X} and \mathbf{Z} are matrices with one row per data point

$$\mathbf{X} = \begin{bmatrix} [\mathbf{x}^{(1)}]^\top \\ [\mathbf{x}^{(2)}]^\top \\ \vdots \\ [\mathbf{x}^{(N)}]^\top \end{bmatrix} \in \mathbb{R}^{N \times D} \quad \text{and} \quad \mathbf{Z} = \begin{bmatrix} [\mathbf{z}^{(1)}]^\top \\ [\mathbf{z}^{(2)}]^\top \\ \vdots \\ [\mathbf{z}^{(N)}]^\top \end{bmatrix} \in \mathbb{R}^{N \times K}$$

- Can write the squared reconstruction error as

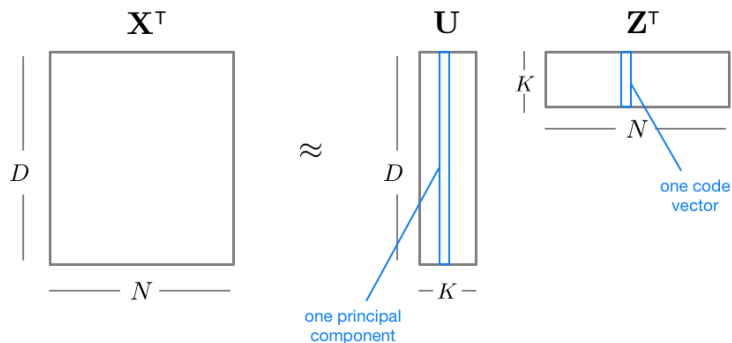
$$\sum_{i=1}^N \|\mathbf{x}^{(i)} - \mathbf{U}\mathbf{z}^{(i)}\|^2 = \|\mathbf{X} - \mathbf{Z}\mathbf{U}^\top\|_F^2,$$

- $\|\cdot\|_F$ denotes the **Frobenius norm**:

$$\|\mathbf{Y}\|_F^2 = \|\mathbf{Y}^\top\|_F^2 = \sum_{i,j} y_{ij}^2 = \sum_i \|\mathbf{y}^{(i)}\|^2.$$

PCA as Matrix Factorization

- So PCA is approximating $\mathbf{X} \approx \mathbf{Z}\mathbf{U}^\top$, or equivalently $\mathbf{X}^\top \approx \mathbf{U}\mathbf{Z}^\top$.














- Based on the sizes of the matrices, this is a rank- K approximation.
- Since \mathbf{U} was chosen to minimize reconstruction error, this is the *optimal* rank- K approximation, in terms of error $\|\mathbf{X}^\top - \mathbf{U}\mathbf{Z}^\top\|_F^2$.

Matrix Completion

- We just saw that PCA gives the optimal low-rank matrix factorization to a matrix \mathbf{X} .

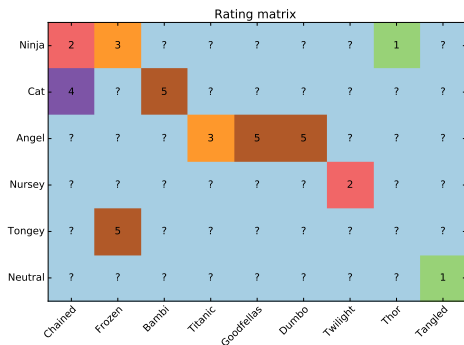
The Netflix problem

Recall: [movie recommendation](#).

User	Movie	Rating
	Thor	★ ☆ ☆ ☆ ☆
	Chained	★ ★ ☆ ☆ ☆
	Frozen	★ ★ ★ ☆ ☆
	Chained	★ ★ ★ ★ ☆
	Bambi	★ ★ ★ ★ ★
	Titanic	★ ★ ★ ☆ ☆
	Goodfellas	★ ★ ★ ★ ★
	Dumbo	★ ★ ★ ★ ★
	Twilight	★ ★ ☆ ☆ ☆
	Frozen	★ ★ ★ ★ ★
	Tangled	★ ☆ ☆ ☆ ☆

Matrix Completion

Matrix completion problem: Transform the table into a N users by M movies matrix \mathbf{R}



- **Data:** Users rate some movies.
 $\mathbf{R}_{\text{user}, \text{movie}}$. Very sparse
- **Task:** Predict missing entries, i.e. how a user would rate a movie they haven't previously rated
- **Evaluation Metric:** Squared error (used by Netflix Competition). Is this a reasonable metric?

Matrix Completion

- In our current setting, **latent factor models** attempt to explain the ratings by characterizing both movies and users on a number of factors K inferred from the ratings patterns.
- That is, we seek representations for movies and users as vectors in \mathbb{R}^K that can ultimately be translated to ratings.
- For simplicity, we can associate these factors (i.e. the dimensions of the vectors) with idealized concepts like
 - ▶ comedy
 - ▶ drama
 - ▶ action
 - ▶ But also uninterpretable dimensions

Can we use the sparse ratings matrix \mathbf{R} to find these latent factors automatically?

Matrix Completion

- Let the representation of user i in the K -dimensional space be \mathbf{u}_i and the representation of movie j be \mathbf{z}_j
 - ▶ Intuition: maybe the first entry of \mathbf{u}_i says how much the user likes horror films, and the first entry of \mathbf{z}_j says how much movie j is a horror film.
- Assume the rating user i gives to movie j is given by a dot product:
 $R_{ij} \approx \mathbf{u}_i^\top \mathbf{z}_j$
- In matrix form, if:

$$\mathbf{U} = \begin{bmatrix} - & \mathbf{u}_1^\top & - \\ & \vdots & \\ - & \mathbf{u}_N^\top & - \end{bmatrix} \text{ and } \mathbf{Z}^\top = \begin{bmatrix} | & & | \\ \mathbf{z}_1 & \dots & \mathbf{z}_M \\ | & & | \end{bmatrix}$$

then: $\mathbf{R} \approx \mathbf{U}\mathbf{Z}^\top$

- This is a matrix factorization problem!

Matrix Completion

- Recall PCA: To enforce $\mathbf{X}^\top \approx \mathbf{U}\mathbf{Z}^\top$, we minimized

$$\min_{\mathbf{U}, \mathbf{Z}} \|\mathbf{X}^\top - \mathbf{U}\mathbf{Z}^\top\|_F^2 = \sum_{i,j} (x_{ji} - \mathbf{u}_i^\top \mathbf{z}_j)^2$$

where \mathbf{u}_i and \mathbf{z}_j are the i -th rows of matrices \mathbf{U} and \mathbf{Z} , respectively.

- What's different about the Netflix problem?
 - ▶ Most entries are missing!
 - ▶ We only want to count the error for the observed entries.

Matrix Completion

- Let $O = \{(n, m) : \text{entry } (n, m) \text{ of matrix } \mathbf{R} \text{ is observed}\}$
- Using the squared error loss, matrix completion requires solving

$$\min_{\mathbf{U}, \mathbf{Z}} \frac{1}{2} \sum_{(i,j) \in O} (R_{ij} - \mathbf{u}_i^\top \mathbf{z}_j)^2$$

- The objective is non-convex in \mathbf{U} and \mathbf{Z} jointly, and in fact it's generally NP-hard to minimize the above cost function exactly.
- As a function of either \mathbf{U} or \mathbf{Z} individually, the problem is convex and easy to optimize. We can use coordinate descent, just like with K-means!

Alternating Least Squares (ALS): fix \mathbf{Z} and optimize \mathbf{U} , followed by fix \mathbf{U} and optimize \mathbf{Z} , and so on until convergence.

Alternating Least Squares

- Want to minimize the squared error cost with respect to the factor \mathbf{U} . (The case of \mathbf{Z} is exactly symmetric.)
- We can decompose the cost into a sum of independent terms:

$$\sum_{(i,j) \in O} \left(R_{ij} - \mathbf{u}_i^\top \mathbf{z}_j \right)^2 = \sum_i \underbrace{\sum_{j:(i,j) \in O} \left(R_{ij} - \mathbf{u}_i^\top \mathbf{z}_j \right)^2}_{\text{only depends on } \mathbf{u}_i}$$

This can be minimized independently for each \mathbf{u}_i .

- This is a linear regression problem in disguise. Its optimal solution is:

$$\mathbf{u}_i = \left(\sum_{j:(i,j) \in O} \mathbf{z}_j \mathbf{z}_j^\top \right)^{-1} \sum_{j:(i,j) \in O} R_{ij} \mathbf{z}_j$$

Alternating Least Squares

ALS for Matrix Completion problem

1. Initialize \mathbf{U} and \mathbf{Z} randomly
2. repeat until convergence
3. **for** $i = 1, \dots, N$ **do**
4. $\mathbf{u}_i = \left(\sum_{j:(i,j) \in O} \mathbf{z}_j \mathbf{z}_j^\top \right)^{-1} \sum_{j:(i,j) \in O} R_{ij} \mathbf{z}_j$
5. **for** $j = 1, \dots, M$ **do**
6. $\mathbf{z}_j = \left(\sum_{i:(i,j) \in O} \mathbf{u}_i \mathbf{u}_i^\top \right)^{-1} \sum_{i:(i,j) \in O} R_{ij} \mathbf{u}_i$

Next?

- So far we have motivated unsupervised learning by discovering latent structure in data: clusters or linear structure.
- Now we will start to put together a more formulaic (perhaps more principled) view of unsupervised learning as a probabilistic method.
- Actually this point of view will also unify supervised learning.

Next?

- So far in the course we have adopted a modular perspective, in which the model, loss function, optimizer, and regularizer are specified separately.
- Today we will begin putting together a **probabilistic interpretation** of the choice of model and loss, and introduce the concept of **maximum likelihood estimation**.
- Let's start with a simple biased coin example.
 - ▶ You flip a coin $N = 100$ times and get outcomes $\{x_1, \dots, x_N\}$ where $x_i \in \{0, 1\}$ and $x_i = 1$ is interpreted as heads H .
 - ▶ Suppose you had $N_H = 55$ heads and $N_T = 45$ tails.
 - ▶ What is the probability it will come up heads if we flip again? Let's design a model for this scenario, fit the model. We can use the fit model to predict the next outcome.

Model?

- The coin is possibly loaded. So, we can assume that one coin flip outcome x is a **Bernoulli random variable** for *some currently unknown parameter* $\theta \in [0, 1]$.

$$p(x = 1|\theta) = \theta \quad \text{and} \quad p(x = 0|\theta) = 1 - \theta$$

$$\text{or more succinctly } p(x|\theta) = \theta^x(1 - \theta)^{1-x}$$

- It's sensible to *assume* that $\{x_1, \dots, x_N\}$ are **independent and identically distributed (i.i.d.)** Bernoullis.
- Thus the joint probability of the outcome $\{x_1, \dots, x_N\}$ is

$$p(x_1, \dots, x_N|\theta) = \prod_{i=1}^N \theta^{x_i} (1 - \theta)^{1-x_i}$$

Loss?

- We call the probability mass (or density for continuous) of the observed data the **likelihood function** (as a function of the parameters θ):

$$L(\theta) = \prod_{i=1}^N \theta^{x_i} (1 - \theta)^{1-x_i}$$

- We usually work with log-likelihoods:

$$\ell(\theta) = \sum_{i=1}^N x_i \log \theta + (1 - x_i) \log(1 - \theta)$$

- How can we choose θ ? Good values of θ should assign high probability to the observed data. This motivates the **maximum likelihood criterion**, that we should pick the parameters that maximize the likelihood:

$$\hat{\theta}_{\text{ML}} = \arg \max_{\theta \in [0,1]} \ell(\theta)$$

Maximum Likelihood Estimation for the Coin Example

- Remember how we found the optimal solution to linear regression by setting derivatives to zero? We can do that again for the coin example.

$$\begin{aligned}\frac{d\ell}{d\theta} &= \frac{d}{d\theta} \left(\sum_{i=1}^N x_i \log \theta + (1 - x_i) \log(1 - \theta) \right) \\ &= \frac{d}{d\theta} (N_H \log \theta + N_T \log(1 - \theta)) \\ &= \frac{N_H}{\theta} - \frac{N_T}{1 - \theta}\end{aligned}$$

where $N_H = \sum_i x_i$ and $N_T = N - \sum_i x_i$.

- Setting this to zero gives the maximum likelihood estimate:

$$\hat{\theta}_{\text{ML}} = \frac{N_H}{N_H + N_T}.$$

Maximum Likelihood Estimation

- Notice, in the coin example we are actually minimizing cross-entropies!

$$\begin{aligned}\hat{\theta}_{\text{ML}} &= \arg \max_{\theta \in [0,1]} \ell(\theta) \\ &= \arg \min_{\theta \in [0,1]} -\ell(\theta) \\ &= \arg \min_{\theta \in [0,1]} \sum_{i=1}^N -x_i \log \theta - (1 - x_i) \log(1 - \theta)\end{aligned}$$

- This is an example of **maximum likelihood estimation**.
 - ▶ define a model that assigns a probability (or has a probability density at) to a dataset
 - ▶ maximize the likelihood (or minimize the neg. log-likelihood).
- Many examples we've considered fall in this framework! Let's consider classification again.

Generative vs Discriminative

Two approaches to classification:

- **Discriminative approach:** estimate parameters of decision boundary/class separator directly from labeled examples.
 - ▶ Model $p(t|\mathbf{x})$ directly (logistic regression models)
 - ▶ Learn mappings from inputs to classes (linear/logistic regression, decision trees etc)
 - ▶ Tries to solve: How do I separate the classes?
- **Generative approach:** model the distribution of inputs characteristic of the class (Bayes classifier).
 - ▶ Model $p(\mathbf{x}|t)$
 - ▶ Apply Bayes Rule to derive $p(t|\mathbf{x})$.
 - ▶ Tries to solve: What does each class "look" like?
- Key difference: is there a distributional assumption over inputs?

A Generative Model: Bayes Classifier

- Aim to classify text into spam/not-spam (yes $c=1$; no $c=0$)
- Example: “You are one of the very few who have been selected as a winners for the free \$1000 Gift Card.”
- Use bag-of-words features, get binary vector \mathbf{x} for each email
- Vocabulary:
 - ▶ “a”: 1
 - ▶ ...
 - ▶ “car”: 0
 - ▶ “card”: 1
 - ▶ ...
 - ▶ “win”: 0
 - ▶ “winner”: 1
 - ▶ “winter”: 0
 - ▶ ...
 - ▶ “you”: 1

Bayes Classifier

- Given features $\mathbf{x} = [x_1, x_2, \dots, x_D]^T$ we want to compute class probabilities using Bayes Rule:

$$\underbrace{p(c|\mathbf{x})}_{\text{Pr. class given words}} = \frac{p(\mathbf{x}, c)}{p(\mathbf{x})} = \frac{\overbrace{p(\mathbf{x}|c)}^{\text{Pr. words given class}} p(c)}{p(\mathbf{x})}$$

- 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 $p(c)$

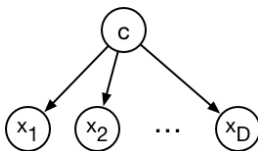
- Assume we have two classes: spam and non-spam. We have a dictionary of D words, and binary features $\mathbf{x} = [x_1, \dots, x_D]$ saying whether each word appears in the e-mail.
- If we define a joint distribution $p(c, x_1, \dots, x_D)$, this gives enough information to determine $p(c)$ and $p(\mathbf{x}|c)$.
- Problem: specifying a joint distribution over $D + 1$ binary variables requires $2^{D+1} - 1$ entries. This is computationally prohibitive and would require an absurd amount of data to fit.
- We'd like to impose **structure** on the distribution such that:
 - ▶ it can be **compactly** represented
 - ▶ **learning** and **inference** are both tractable

- Naïve assumption: **Naïve Bayes** assumes that the word features x_i are **conditionally independent** given the class c .
 - ▶ This means x_i and x_j are independent under the conditional distribution $p(\mathbf{x}|c)$.
 - ▶ Note: this doesn't mean they're independent.
 - ▶ Mathematically,

$$p(c, x_1, \dots, x_D) = p(c)p(x_1|c)\cdots p(x_D|c).$$

- Compact representation of the joint distribution
 - ▶ Prior probability of class: $p(c = 1) = \pi$ (e.g. spam email)
 - ▶ Conditional probability of word feature given class: $p(x_j = 1|c) = \theta_{jc}$ (e.g. word "price" appearing in spam)
 - ▶ $2D + 1$ parameters total (before $2^{D+1} - 1$)

- We can represent this model using an **directed graphical model**, or **Bayesian network**:



- This graph structure means the joint distribution factorizes as a product of conditional distributions for each variable given its parent(s).
- Intuitively, you can think of the edges as reflecting the order in which data is generated.

Naïve Bayes: Learning

- The parameters can be learned efficiently because the log-likelihood decomposes into independent terms for each feature.

$$\begin{aligned}\ell(\boldsymbol{\theta}) &= \sum_{i=1}^N \log p(c^{(i)}, \mathbf{x}^{(i)}) = \sum_{i=1}^N \log \left\{ p(\mathbf{x}^{(i)} | c^{(i)}) p(c^{(i)}) \right\} \\ &= \sum_{i=1}^N \log \left\{ p(c^{(i)}) \prod_{j=1}^D p(x_j^{(i)} | c^{(i)}) \right\} \\ &= \sum_{i=1}^N \left[\log p(c^{(i)}) + \sum_{j=1}^D \log p(x_j^{(i)} | c^{(i)}) \right] \\ &= \underbrace{\sum_{i=1}^N \log p(c^{(i)})}_{\text{Bernoulli log-likelihood of labels}} + \sum_{j=1}^D \underbrace{\sum_{i=1}^N \log p(x_j^{(i)} | c^{(i)})}_{\text{Bernoulli log-likelihood for feature } x_j}\end{aligned}$$

- Each of these log-likelihood terms depends on different sets of parameters, so they can be optimized independently.

Naïve Bayes: Learning

- We can handle these terms separately. For the prior we maximize:
 $\sum_{i=1}^N \log p(c^{(i)})$
- This is a minor variant of our coin flip example. Let $p(c^{(i)} = 1) = \pi$.
Note $p(c^{(i)}) = \pi^{c^{(i)}} (1 - \pi)^{1-c^{(i)}}$.
- Log-likelihood:

$$\sum_{i=1}^N \log p(c^{(i)}) = \sum_{i=1}^N c^{(i)} \log \pi + \sum_{i=1}^N (1 - c^{(i)}) \log(1 - \pi)$$

- Obtain MLEs by setting derivatives to zero:

$$\hat{\pi} = \frac{\sum_i \mathbb{1}[c^{(i)} = 1]}{N} = \frac{\# \text{ spams in dataset}}{\text{total \# samples}}$$

Naïve Bayes: Learning

- Each θ_{jc} 's can be treated separately: maximize $\sum_{i=1}^N \log p(x_j^{(i)} | c^{(i)})$
- This is (again) a minor variant of our coin flip example.

Let $\theta_{jc} = p(x_j^{(i)} = 1 | c)$. Note $p(x_j^{(i)} | c) = \theta_{jc}^{x_j^{(i)}} (1 - \theta_{jc})^{1-x_j^{(i)}}$.

- Log-likelihood:

$$\begin{aligned} \sum_{i=1}^N \log p(x_j^{(i)} | c^{(i)}) &= \sum_{i=1}^N c^{(i)} \{x_j^{(i)} \log \theta_{j1} + (1 - x_j^{(i)}) \log(1 - \theta_{j1})\} \\ &\quad + \sum_{i=1}^N (1 - c^{(i)}) \{x_j^{(i)} \log \theta_{j0} + (1 - x_j^{(i)}) \log(1 - \theta_{j0})\} \end{aligned}$$

- Obtain MLEs by setting derivatives to zero:

$$\hat{\theta}_{jc} = \frac{\sum_i \mathbb{1}[x_j^{(i)} = 1 \ \& \ c^{(i)} = c]}{\sum_i \mathbb{1}[c^{(i)} = c]} \quad \text{for } c = 1 \quad \frac{\text{\#word } j \text{ appears in spams}}{\text{\# spams in dataset}}$$

Naïve Bayes: Inference

- We predict the category by performing **inference** in the model.
- Apply **Bayes' Rule**:

$$p(c | \mathbf{x}) = \frac{p(c)p(\mathbf{x} | c)}{\sum_{c'} p(c')p(\mathbf{x} | c')} = \frac{p(c) \prod_{j=1}^D p(x_j | c)}{\sum_{c'} p(c') \prod_{j=1}^D p(x_j | c')}$$

- We need not compute the denominator if we're simply trying to determine the most likely c .
- Shorthand notation:

$$p(c | \mathbf{x}) \propto p(c) \prod_{j=1}^D p(x_j | c)$$

- For input \mathbf{x} , predict by comparing the values of $p(c) \prod_{j=1}^D p(x_j | c)$ for different c (e.g. choose the largest).

- Naïve Bayes is an amazingly cheap learning algorithm!
- **Training time:** estimate parameters using maximum likelihood
 - ▶ Compute co-occurrence counts of each feature with the labels.
 - ▶ Requires only one pass through the data!
- **Test time:** apply Bayes' Rule
 - ▶ Cheap because of the model structure. (For more general models, Bayesian inference can be very expensive and/or complicated.)
- We covered the Bernoulli case for simplicity. But our analysis easily extends to other probability distributions.
- Unfortunately, it's usually less accurate in practice compared to discriminative models due to its "naïve" independence assumption.