Datasets

We usually divide our dataset into three subsets:

- Training set:
  - Used to find the optimal parameter given this specific dataset and one/several model(s).
- Validation set:
  - Find the best *model* out of several candidate models (e.g. logistic regression vs SVM, or feature selection).
  - Find the most appropriate value for a *hyperparameter* (e.g. k for k-means, regularization coefficients).
- Test set:
  - Completely hold-out set that is used to give an unbiased estimate of how good your model captures the real underlying data distribution, after the developing a model.

Cross-validation

<u>K-fold cross-validation (LOOCV is special case when K = n):</u>

- Partition dataset into k folds
- Each time use one partition as validation set and rest K-1 folds as training sets.
- Final predictor: average/majority vote over the k hold-out estimates.

<u>Bias-variance tradeoff when choosing K:</u>

If K is large (close to n), then:

- *Bias* of error estimate is small, since each training set has close to n data points.
- *Variance* of error estimate is high, since each validation set has fewer data points, so the error might deviate a lot from the mean.
- Large computational time.

Common choice: K = 10

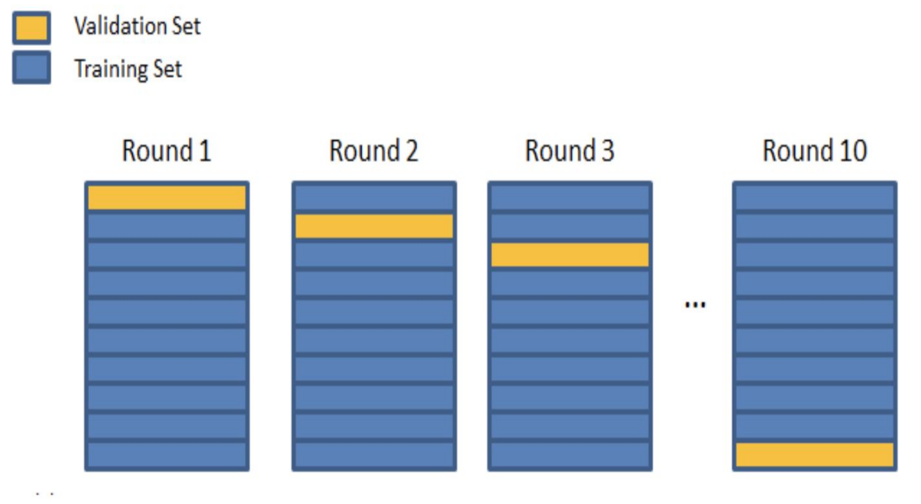


Figure credit: https://codesachin.wordpress.com/2015/08/30/cross-validation-and-the-bias-variance-tradeoff-for-dummies/

Regularization

Overfitting:

- Model has a much higher validation error than the empirical error, since the model tries too hard to capture noise in the training data that don't capture the true properties of data.

Note that:

1. MAP estimators requires us to find $argmax_\theta P(D|\theta)P(\theta)$
2. Then, the log-posterior is: $argmax_\theta log(P(D|\theta)) + logP(\theta)$
3. Both L1 and L2 regularization can be used in any regression techniques.

L1 Regularization / LASSO:

- Assume a Laplace prior for each parameter
  $\theta_i, p(\theta_i) \sim Laplace(0, b)$
- To account for this prior, we need to compute the log-posterior, which means we need to compute $log(P(\theta))$ as shown on the right.

$$\begin{aligned} logP(\theta) &= log\Pi_{\theta_i} \frac{1}{2b} e^{-\frac{|\theta_i - 0|}{b}} \\ &= \sum_i log(\frac{1}{2b} e^{-\frac{|\theta_i|}{b}}) \\ &= \sum_i log(\frac{1}{2b}) - \frac{|\theta_i|}{b} \\ &= \sum_i log(\frac{1}{2b}) - \sum_i \frac{|\theta_i|}{b} \\ &= constant - \frac{1}{b} \sum_i |\theta_i| \end{aligned}$$

Regularization (Cont)

<u>L1 Regularization / LASSO (Cont.):</u>

- When we use any gradient-based optimization technique, our update rule becomes:

$$\theta_i = \theta_i + \frac{\partial log - posterior}{\partial \theta_i}$$
$$= \theta_i + \frac{\partial log - likelihood}{\partial \theta_i} + \begin{cases} -\frac{1}{b}, & \text{if } \theta_i \geq 0 \\ \frac{1}{b}, & \text{otherwise} \end{cases}$$

- Note that, each time, LASSO changes each parameter by the same magnitude 1/b.
- This causes the originally "less important" features to first get to zero => Feature selection!

Regularization (Cont)

<u>L2 Regularization / RIDGE (Cont.):</u>

- Assume a Gaussian prior for each parameter: $\theta_i, P(\theta_i) \sim Gaussian(0, \sigma^2)$
- To account for this prior, we need to compute the log-posterior, which means we need to compute $log(P(\theta))$ as shown below.

$$
\begin{aligned}
logP(\theta) &= log(\Pi_{\theta_i} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\theta_i - 0)^2}{2\sigma^2}}) \\
&= \sum_i log(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\theta_i^2}{2\sigma^2}}) \\
&= \sum_i (log(\frac{1}{\sqrt{2\pi\sigma^2}}) - \frac{(\theta_i)^2}{2\sigma^2}) \\
&= \sum_i log(\frac{1}{\sqrt{2\pi\sigma^2}}) - \sum_i \frac{(\theta_i)^2}{2\sigma^2} \\
&= Constant - \frac{1}{2\sigma^2} \sum_i (\theta_i)^2
\end{aligned}
$$

- When we use any gradient-based optimization technique, our update rule becomes:

$$
\begin{aligned}
\theta_i &= \theta_i + \frac{\partial log - posterior}{\theta_i} \\
&= \theta_i + \frac{\partial log - likelihood}{\theta_i} - \frac{1}{\sigma^2}\theta_i
\end{aligned}
$$

- Note that, each time, RIDGE changes each parameter by an amount that is proportional to its own magnitude, i.e. $|\theta_i|$.
- Different from LASSO, all the parameters come closer to zero altogether at the same pace!

Naive Bayes

- Naive Bayes is a generative algorithm: $P(y|\mathbf{x}) \propto P(\mathbf{x}|y)P(y)$
- Naive Bayes Assumption: all the attributes *given label* are conditionally independent. I.e. $P(\mathbf{x}|y) = \Pi_j P(\mathbf{x}_j|y)$.
- Note: parameters are trained separately for each class.
- The classification rule therefore is: $\hat{y} = argmax_c \Pi_m P(\mathbf{x}_m|y=c)P(y=c)$
- Common text document encoding: bag of words
    a. The feature vector for a document is represented as $\mathbf{x} = [\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_m)]$, where $\phi(\mathbf{x}_i)$ = 1 if the ith word appears in this document and zero otherwise.
- Discrete Input Values:
    a. We model the mth feature as: $\mathbf{x}_m|y=c \sim Bern(\theta_{cm})$
    b. Add-k smoothing: equivalent to a Beta(k+1, k+1) prior on each parameter θ_{cm}
- Continuous Input Values:
    a. We model each feature vector as: $\mathbf{x}|y=c \sim N(\mu_c, \Sigma_c)$, where each Σ_c is a *diagonal* matrix by NB assumption. (One Gaussian model per class.)
- If the labels are binary, we model them using Bernoulli. Otherwise, we model them using Multinomial distribution.

SVM

- Support vectors:
    - For linearly separable case, these are the data on the boundary; for non-linearly separable case, these also include those misclassified data.
- Maximum Margin Classifier:
    - Margin is defined by the two closest positive and negative training examples.
    - $M = \frac{2}{\sqrt{w^T w}}$
- Optimization Problem in Primal Form:
    - Linearly separable: $min \frac{w^T w}{2}$, where
        - $\mathbf{w}^T \mathbf{x} + b \geq 1$ for all positive data, $\mathbf{w}^T \mathbf{x} + b \leq -1$ for all negative data.
    - Non-linearly separable: $min \frac{w^T w}{2} + \sum_{i=1}^{n} C\xi_i$, where
        - $\mathbf{w}^T \mathbf{x} + b \geq 1 - \xi_i$ for all positive data; $\mathbf{w}^T \mathbf{x} + b \leq -1 + \xi_i$ for all negative data.
        - For all i, $\xi_i \geq 0$ (nonzero when the ith data is misclassified).
- Kernel Methods:
    - Motivation:
        - higher dimension allows non-linearly separable data to be linearly separable
        - Reduce computational efficiency than working directly in the feature space.
    - Rewrite algorithms so that we only work with dot products of feature vectors: $\mathbf{x}^T \mathbf{z}$

SVM (Cont.)

- Dual SVM for linearly separable case:
  - $\min_{w,b} \max_{\alpha} \frac{\mathbf{w}^T\mathbf{x}}{2} - \sum_i \alpha_i[(\mathbf{w}^T\mathbf{x}+b)y-1]$, where $\alpha_i \geq 0, \forall i$
- Why are alpha values ≥ 0?
  - For the support vectors: $(\mathbf{w}^T\mathbf{x}+b)y-1=0$. To maximize, alpha can be anything.
  - For the correctly classified non-support vectors: $(\mathbf{w}^T\mathbf{x}+b)y-1>0$. To maximize, alpha theoretically should be ∞ . However, we cannot minimize our objective wrt $\mathbf{w}, b$ to if it is ∞ . Therefore, we need alpha to be zero for the non-support vectors.
  - For the misclassified data: $(\mathbf{w}^T\mathbf{x}+b)y-1<0$. To maximize, alpha will be ∞ , which makes our objective ∞. Thereby, we know some of our constraints are not satisfied.
- After Optimizing wrt $\mathbf{w}, b$:
  - $\max \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$ s.t. $\alpha_i \geq 0 \; \forall i, \sum_i \alpha_i y^{(i)} = 0$
- Why dual?
  - We have reduced the number of parameters from feature space ($\mathbf{w}, b$) to sample space (alpha; the number of support vectors).
  - Kernel trick: the prediction function is now transformed to only include the dot product between two data.

PCA

- Motivation:
  - To visualize and discover hidden patterns.
  - Preprocessing for supervised task (data compression, noise reduction).
- Goal:
  - Minimize reconstruction error: $\sum_{i=1}^{N} \|\mathbf{x}^{(i)'} - \mathbf{x}^{(i)}\|_2^2$. (e.g. to be a better autoencoder).
  - Maximize variance: $\sum_{i=1}^{N} (\mathbf{v}^T \mathbf{x}^{(i)})^2$. (So that it is easier to identify patterns.)
- Data needs to be centered.
- Z = XV, where V is m by k (The columns of V are the top k eigenvectors of $X^T X$.)
- Importantly, we have shown that:
  - Any vector $\mathbf{v}$ that maximizes the variance satisfy the following: $\Sigma\mathbf{v} = \lambda\mathbf{v}$
  - λ_i equals the variance of the projections along the eigenvector $\mathbf{v}_i$
- Therefore, the m eigenvectors of $X^T X$ are orthonormal directions of max variance.
- Choose k such that we retain some fraction of the variance: $\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i}$ (e.g. 95%).

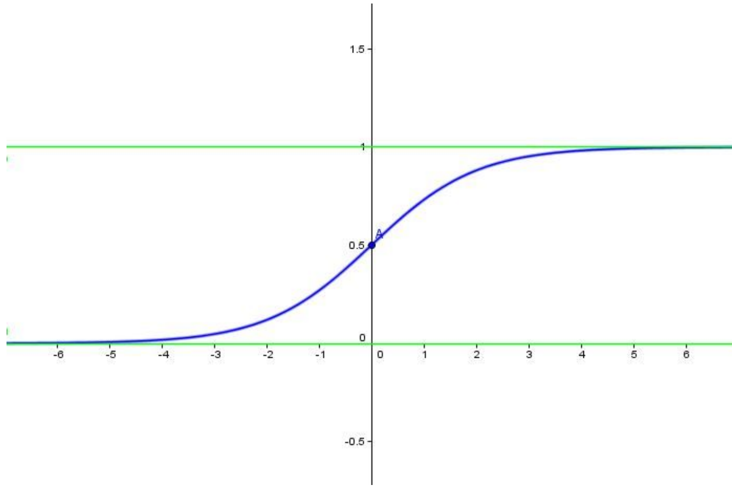
Linear Regression setup

- X: n*m matrix of inputs
- y: n*1 vector of outputs
- w: m*1 vector of weights
- Objective function: $J(w) = (½)(1/N)||y-Xw||_2^2$
  - fractions in front don't matter, constant w.r.t w
- Remember: this assumes a single point y has a Gaussian distribution centered at $x^T w$. Maximizing log likelihood of dataset assuming Gaussian is same as minimizing J(w)

Linear Regression solving

- Goal: make J as small as possible. Since J is concave, make $\nabla_w J(w) = 0$.
- Two ways to do this: closed form, or move towards minimum with gradient descent
- $\nabla_w J(w) = X^T(y - Xw)$ check: same shape as w?
- Closed form: $w = (X^T X)^{-1} X^T y$ (proof in slides)
- Gradient descent: $w \mathrel{-}= \alpha \nabla_w J(w)$ repeatedly until convergence
    - α = learning rate
    - Too small: takes too long to converge
    - Too large: can overshoot by jumping over minimum
- Remember: w is parameter, α is hyperparameter

Logistic Regression setup



- $y \sim \text{Bern}(u)$, $u = g(w^T x)$, $g(z) = (1 + \exp(-z))^{-1}$
- Constraints value between 0 and 1
  - Corresponds to probability
- Want to maximize log likelihood
- For single sample, what is prob of that sample?
- $g(w^T x)^y (1-g(w^T x))^{1-y}$
  - If y = 1, then the first term remains and we are left with the probability of y = 1
  - If y = 0, then the second term remains and we are left with the probability of y = 0
- Likelihood is simply the product of these terms for each sample
- Log likelihood is the log of this product - becomes sum of logs
- $l(w) = \Sigma_i \, y^{(i)} \log u^{(i)} + (1-y^{(i)}) \log(1-u^{(i)})$

Logistic Regression solving

- $\nabla_w l(w) = \Sigma_i (y^{(i)}-u^{(i)})x^{(i)T}$
- Notice how similar to linear regression gradient except for u
- No closed form solution! Have to use gradient descent etc

Decision Trees setup

- Classifier based on making yes-or-no decisions (imagine gigantic if/else)
- For us, talking about decision trees means talking about ID3
- ID3: greedy search for decision tree
- At each node, discriminate on the feature that helps us determine the label the best
- How do we quantify this?

Entropy and Information Gain

- Entropy: $H(Y) = -\Sigma_y P(Y=y) \lg P(Y=y)$
  - closer to 0 if “less random”, >> 0 if “more random”
  - E.g. If $P(Y=1)=1$, then $H(Y) = 0$; if $P(Y=1) = ½$, then $H(Y) = ½$
- Conditional Entropy: $H(Y|X) = \Sigma_x P(X=x) H(Y|X=x)$
  - Simply expected value of $H(Y)$ over the distribution of X
  - $H(Y|X=x)$ is simply $H(Y)$ just looking at the samples where $X=x$
- Information Gain: $H(Y) - H(Y|X)$
  - “How much more sure are we about Y now that we know X?”

Decision Trees - ID3

- At each node, pick the attribute that maximizes information gain
- If every attribute is the same (i.e. all one value), then information gain would be 0. Simply predict the mode of the Y's in this set.
    - Example dataset: {(X=A,Y=1),(X=A,Y=0),(X=A,Y=1)}. Predict 1 because there's no point in discriminating on X.
- Continuous variables: If there are n discrete values, then there are n+1 possible boundaries to split on
    - Have to check every possible split for best information gain
- ID3 doesn't make the 'best' tree! Simply a good approximation.

Neural Networks - Layers

- Each layer consists of two parts: matrix multiplication and elementwise function
- Matrix multiplication (“linear layers”)
  - Input X (n*m)   where n = number of samples, m
  - Weight W (m*k) and bias b (k*1)
  - Output Y = XW + b   where b is added to each row of XW
  - Therefore each row is $x^T w + b$
- Elementwise function (these are usually used as activation function)
  - Sigmoid, tanh, etc.
  - ReLU: f(x) = max(0, x)
  - Generally placed between linear layers. This is what give neural nets their power.

Neural Networks - Backpropagation

- Just like with logistic regression, no closed form easy solution for minimizing loss
- Use gradient descent - iteratively subtract the derivative of the loss w.r.t the parameters. Problem: how do we know the derivative of the loss w.r.t. a hidden layer? Remember chain rule.
- L = f(Y), Y = WX + b
- dL/dW = (dY/dW)(dL/dY)
- m*k   m*n   n*k
- dL/dY will be given to us (passed backwards through the network)
- dY/dW is X, intuitively from scalar calculus (won’t go through proof here)

Learning Theory

- R(h): risk of a hypothesis. Expected loss over X and Y. Unknown.
- R-hat(h): Risk over sample dataset (training error). Can be measured.
    - If equal to 0, we have fitted dataset perfectly.
- Realizable: 0 training error. Agnostic: nonzero training error.
- H: Hypothesis space. Can be finite (e.g. set of decision trees on categorical input) or infinite (e.g. set of all possible weights in linear regression)
- We want to find the minimum N such that N examples are sufficient for a bound on our true error

Learning Theory cont.

- Finite and realizable: $N >= 1/\epsilon(\log |H| + \log(1/\delta))$ examples are sufficient so that with prob $(1-\delta)$ all h in H with R-hat(h) = 0 (perfect training accuracy) have true error $R(h) <= \epsilon$.
- Finite and agnostic: $N >= 1/(2\epsilon^2)(\log |H| + \log(2/\delta))$ examples are sufficient so that with prob $(1-\delta)$ for all h in H we have $|R(h)-(R\text{-hat}(h))| <= \epsilon$.
  - Notice how this differs from the other case

MLE/MAP

- **MLE:** Find parameter θ that maximizes likelihood of <u>observed data</u>, $\text{argmax}_\theta\ p(D|\theta)$.
  - $L(D,\theta) = p(D|\theta) = \prod p(d_i|\theta)$

- **MAP:** Find parameter θ that maximizes likelihood of <u>posterior probability</u>, $\text{argmax}_\theta\ p(\theta|D)$.
  - Posterior ∝ Likelihood x Prior
  - $L(D,\theta) = p(\theta|D) \propto \prod p(d_i|\theta)\ p(\theta)$.

- Steps to finding parameter θ:
  1. Formulate likelihood function $L(D,\theta)$
  2. Take the log to get log-likelihood
  3. Take derivative of log-likelihood w.r.t θ, set it to 0, and solve for θ

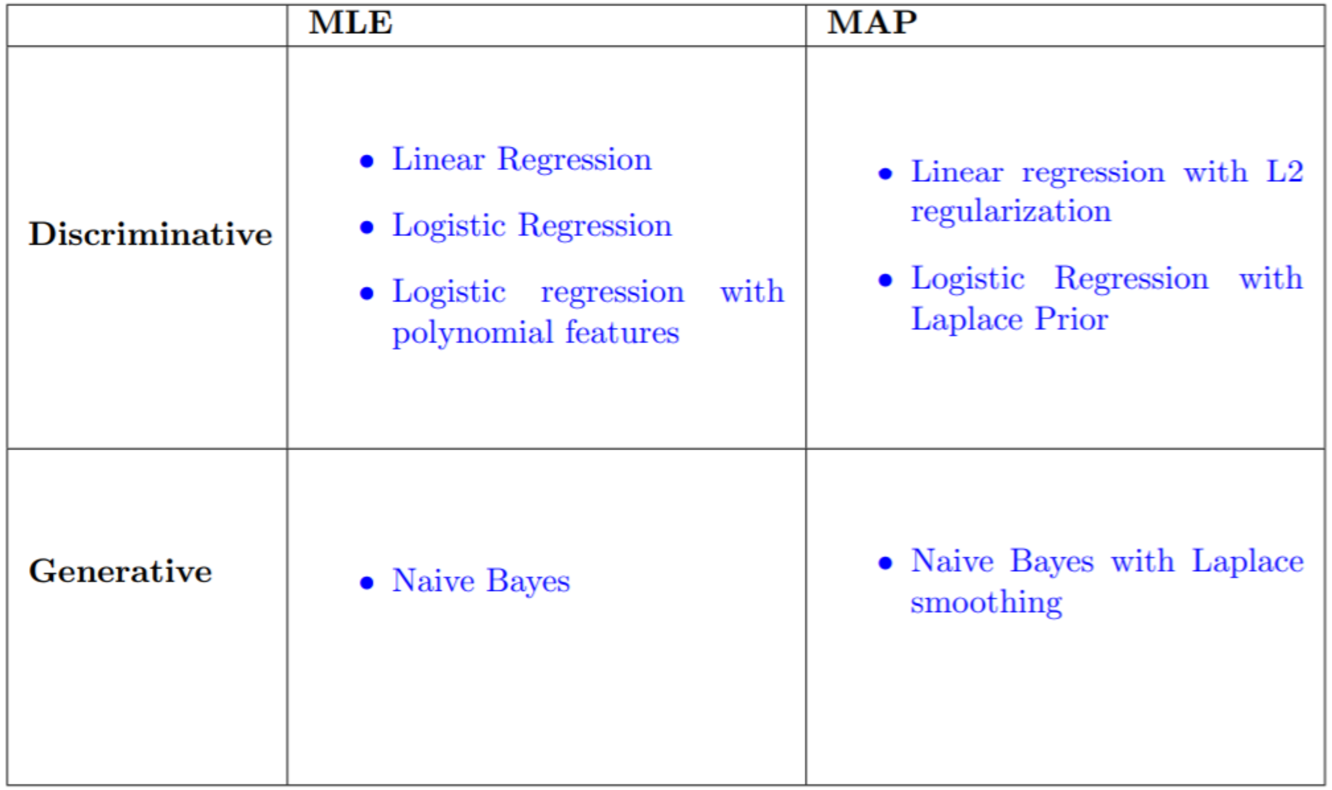
Resources: Lectures 4&5, Recitation 4, Homework 3&4

Generative Models

Generative vs Discriminative: Both eventually predicts P(Y|X)

- Generative:
  - Estimates P(Y) and P(X|Y) directly from data and gets joint distribution P(X,Y)
  - Predicts P(Y|X) with Bayes Theorem
- Discriminative:
  - Estimates P(Y|X) directly from data

Resources: Lecture 9, Recitation 5, Homework 5



	MLE	**MAP**
Discriminative	• Linear Regression • Logistic Regression • Logistic regression with polynomial features	• Linear regression with L2 regularization • Logistic Regression with Laplace Prior
Generative	• Naive Bayes	• Naive Bayes with Laplace smoothing

K-Nearest Neighbor

Given a new point *x*, predict its label *y* by

- finding its <u>closest</u> *k* neighbors [by metrics such as Euclidean distance]
- returning the most common class label among the *k* neighbors

K is a hyperparameter:

- Too small: overfitting | too dependent on the nearest single datapoint
- Too large: stable but too simple | considers irrelevant, far-off points

Resources: Lecture 13, Recitation 8, Homework 7

Non-Parametric Regression

- Non-parametric model: Number of parameters scale with number of training data
  - i.e. K-Nearest Neighbors Classifier, Decision Trees (sometimes), Kernel Regression

- Recall the steps to Kernel Regression:
  - **Step 1:** Compute $\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda\mathbb{I})^{-1}\boldsymbol{y}$ where: $\boldsymbol{K}_{ij} = \boldsymbol{k}(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ and **k** is your kernel.
  - **Step 2:** Given a new point **x**, predict $\hat{y} = \sum_{i=1}^{N} \alpha_i k(x, x^{(i)})$
  - α used as a 'normalizer' to allow weighted sum of kernel windows
  - λ used to ensure term is invertible

Resources: Lecture 16, Recitation 9, Homework 8

Clustering

Partition unlabeled data into groups of similar datapoints

Hierarchical algorithms:

- Bottom-up (single-linkage, complete-linkage, centroid, average-linkage)
- Top-down

Partition algorithms:

- K-means clustering (K-medoids)
- Mixture-Model based clustering

  Expectation Maximization (EM) with Gaussian Mixture Models (EM)

  $\lambda = \mu_1, \mu_2, \ldots, \mu_k, \Sigma_1, \ldots, \Sigma_k, \pi_1, \ldots, \pi_k$ where $\pi_j = P(z_j=1)$

  1. E-step: Calculate posterior probability (“expected” classes) $P(z_j = 1 | x_i , \lambda)$
  2. M-step: Apply MLE and update parameters π_j , μ_j , Σ_j

Resources: Lectures 22&23, Recitation 12, Homework 10

Recommender Systems

Matrix Factorization

Given a matrix $R \in \mathbb{R}^{N \times M}$ with label r_{ij} being rating of user i on item j our objective is to find to matrices $\mathrm{U} \in \mathbb{R}^{N \times K}$ and $\mathrm{V} \in \mathbb{R}^{M \times K}$.

K is a hyperparameter which we can choose. Higher K will give us more accurate predictions at the cost of complexity.

Our objective function is:

$$J(U,V) = \min_{uv} \left|\left|R - UV^T\right|\right|^2$$

However since some values of R are not defined we must instead optimize over a set $S = \{i, j\}$ for all r_{ij} that are known:

$$J(U,V) = \min_{uv} \sum_{i,j \in S} \left(r_{ij} - u^{(i)T} v^{(j)}\right)^2$$

To optimize this we use alternating minimization. This involves fixing v and performing gradient descent optimizing for u, then fixing u and optimizing for v.