6.036 Recitation Notes

Helen Zhou, Parker Zhao

Agenda:

- Recommender Systems
  - content-based vs. collaborative filtering
  - nearest neighbor
  - matrix factorization
  - alternating minimization example
- Kernels
  - Definition
  - Kernel Perceptron
  - Kernel Linear Regression
    - ↳ derivation
    - ↳ example
  - Kernel function properties

RECOMMENDER SYSTEMS

- Purpose: produce recommendation / predict rating that a user would give something (e.g. Netflix movies, Amazon products)

- for this recitation, we use the example of users & movies.

- 2 ways to approach rec. sys.: 1) content-based 2) collaborative filtering
  - ↳ can be complementary, but in this class we address them separately

- Content-based recommendations:
  - represent each movie in terms of features you extract (e.g. [happiness level, whether certain actors appear in it, etc.])
  - can learn θ for each user
  - we <u>already know how to do this!</u>
    - ↳ linear classification / regression
  - <u>limitations</u>: depends on feature quality, and more features ⇒ more parameters to estimate ⇒ need more data

- Collaborative Filtering:
  - "<u>borrow</u>" data from other users
  - recognize <u>aligned preferences</u> rather than explicit features
  - relate users based on their ratings
  - no explicit features beyond ratings
  - Goal: make predictions X_{ai} representative of Y_{ai} (a: user, i: movie) & reasonable for unknown values in Y_{ai}.

$$Y_{ai} = \underbrace{\begin{bmatrix} 5 & 5 & ? & ? \\ ? & 4 & 1 & 0 \\ 3 & ? & 0 & 2 \end{bmatrix}}_{m\text{ movies}} \Big\} n \text{ users}$$

ratings we have observed

want to predict reasonable values for this

Collaborative Filtering

- Nearest Neighbor Prediction:
  - define similarity metric between 2 users a & b:

$$sim(a,b) = corr(a,b) \leftarrow \text{(correlation)}$$

$$= \frac{\sum_{j \in CR(a,b)} (Y_{aj} - \tilde{Y}_a)(Y_{bj} - \tilde{Y}_b)}{\sqrt{\sum_{j \in CR(a,b)} (Y_{aj} - \tilde{Y}_a)^2} \sqrt{\sum_{j \in CR(a,b)} (Y_{bj} - \tilde{Y}_b)^2}}$$

(where $CR(a,b)$ is set of movies both a & b have rated, and $\tilde{Y}_a$ refers to the average rating user a gives across all movies)

↳ ranges from -1 to 1
dissimilar

↳ note that we look @ deviation from average rating because some users generally give higher / lower ratings

- K Nearest Neighbor Prediction:
  weighted average of K nearest neighbors' deviations from their mean, plus one's own average rating

$$X_{ai} = \tilde{Y}_a + \frac{\sum_{b \in KNN(a,i)} sim(a,b)(Y_{bi} - \tilde{Y}_b)}{\sum_{b \in KNN(a,i)} |sim(a,b)|}$$

- notes:
  - conceptually simple
  - few adjustable params (just K)
  - similarity notion is good
  - limitation: users w/ mixed interests

- Matrix Factorization: (low-rank)
  - goal: infer underlying rating matrix for Y
  - if we didn't have constraints on our prediction matrix X:

$$\min \sum_{a,i \in D} \frac{(Y_{ai} - X_{ai})^2}{2} + \frac{\lambda}{2} \sum_{ai} X_{ai}^2$$

$$\Rightarrow \hat{X}_{ai} = \begin{cases} \frac{1}{1+\lambda} Y_{ai}, & (a,i) \in D \\ 0, & (a,i) \notin D \end{cases}$$

we make predictions of 0 for any unobserved entries; this is not what we want to predict

  - rank: controls # of parameters in the matrix
    - ↳ linear alg: dim(CS) = dim(RS) = rank
  - $X = UV^T$ ← prediction

$$U = \begin{bmatrix} — u^{(1)} — \\ \vdots \\ — u^{(n)} — \end{bmatrix} \qquad V^T = \begin{bmatrix} | & & | \\ v^{(1)} & \cdots & v^{(m)} \\ | & & | \end{bmatrix}$$

$n \times k$ (# of users ↗, ↖ rank) $\qquad$ $k \times m$ (rank ↗, ↖ # of movies) $\qquad$ $k \leq \min(n,m)$

  - objective: want to minimize:

$$J(U,V) = \sum_{(a,i) \in D} \frac{(Y_{ai} - [UV^T]_{ai})^2}{2} + \frac{\lambda}{2} \sum_{a=1}^{n} \sum_{j=1}^{k} U_{aj}^2 + \frac{\lambda}{2} \sum_{i=1}^{m} \sum_{j=1}^{k} V_{ij}^2$$

$$= \underbrace{\sum_{(a,i) \in D} \frac{(Y_{ai} - u^{(a)} \cdot v^{(i)})^2}{2}}_{\text{squared error}} + \underbrace{\frac{\lambda}{2} \sum_{a=1}^{n} \|u^{(a)}\|^2 + \frac{\lambda}{2} \sum_{i=1}^{m} \|v^{(i)}\|^2}_{\text{regularization}}$$

↳ notice that for any user $u^{(a)}$, there aren't any shared terms w/ another user

$\Rightarrow$ can optimize separately for each user

Alternating minimization of least squares:

Algorithm:

1) initialize movie feature vectors randomly

$$v^{(1)}, \dots, v^{(m)}.$$

2) fix $v^{(1)}, \dots, v^{(m)}$,

Solve for each $u^{(a)}$, $a = 1, \dots, n$

$$\min \sum_{i:(a,i)\in D} (Y_{ai} - u^{(a)} \cdot v^{(i)})^2/2 + \frac{\lambda}{2}\|u^{(a)}\|^2$$

3) Fix $u^{(1)}, \dots, u^{(n)}$

Solve for each $v^{(i)}$, $i = 1, \dots, m$

$$\min \sum_{a:(a,i)\in D} (Y_{ai} - u^{(a)} \cdot v^{(i)})^2/2 + \frac{\lambda}{2}\|v^{(i)}\|^2$$

- Example: rank 2 matrix factorization (just going to do one iteration, fixing V's & solving for u's)

problem: (want to predict ?'s)

$$Y = \begin{bmatrix} 2 & ? & 0 \\ ? & 2 & 1 \end{bmatrix}, \quad \lambda = 1$$

Solution:

randomly initialize V:

$$V = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 2 & 1 \end{bmatrix}$$

want to solve for u's:

$$U = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix}$$

$$UV^T = \begin{bmatrix} u_{11} & u_{11} + 2u_{12} & 2u_{11} + u_{12} \\ u_{21} & u_{21} + 2u_{22} & 2u_{21} + u_{21} \end{bmatrix}$$

$$J(U,V) = \frac{(2-u_{11})^2}{2} + \frac{(0-(2u_{11}+u_{12}))^2}{2} + \frac{(2-(u_{21}+2u_{22}))^2}{2} + \frac{(1-(2u_{21}+u_{21}))^2}{2} + \frac{\lambda}{2}(u_{11}^2 + u_{12}^2 + u_{21}^2 + u_{22}^2) + \frac{\lambda}{2}\underbrace{(1^2+1^2+2^2+0^2+2^2+1^2)}_{\text{constant } c}$$

minimize J with respect to u's.

$$\frac{dJ}{du_{11}} = -(2-u_{11}) + 2(2u_{11}+u_{12}) + \lambda u_{11} = 0$$

$$\Rightarrow -2+(5+\lambda)u_{11} + 2u_{12} = 0$$

$$\Rightarrow -2+6u_{11}+2u_{12} = 0$$

$$\frac{dJ}{du_{12}} = (2u_{11}+u_{12}) + \lambda u_{12} = 0$$

$$\Rightarrow 2u_{11} + (1+\lambda)u_{12} = 0$$

$$\Rightarrow 2u_{11} + 2u_{12} = 0$$

↳ notice that we can solve for user 1 at this point (can solve for each user independently)

$$\Rightarrow \begin{cases} 6u_{11}+2u_{12} = 2 \\ 2u_{11}+2u_{12} = 0 \end{cases} \Rightarrow 6u_{11} - 2u_{11} = 2 \quad \left. \begin{array}{l} u_{11} = 1/2 \\ u_{12} = -1/2 \end{array} \right]$$

Doing the same for u_{21} & u_{22},

$u_{21} = 1/5$

$u_{22} = 7/10$

$$\Rightarrow U = \begin{bmatrix} 1/2 & -1/2 \\ 1/5 & 7/10 \end{bmatrix} \Rightarrow UV^T = \begin{bmatrix} 1/2 & -1/2 & 1/2 \\ 1/5 & 8/5 & 11/10 \end{bmatrix}$$

← known entries not very similar to Y; continue iterating...

Next, fix U & find V ...

and so on until convergence ...

· Notes:

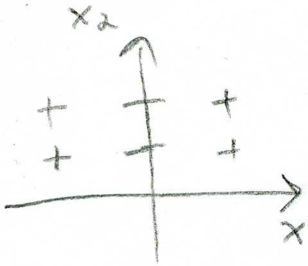
· initialization matters: if all 0's in V, won't be able to update parameters in U well.

· good to run a few times with random initialization

KERNELS

- linear classifiers for nonlinear predictions

$x \in \mathbb{R}^d \rightarrow \phi(x) \in \mathbb{R}^p, \quad p >> d$



transformation to make this linearly separable?

$$\underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} \rightarrow \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \end{bmatrix}}_{\phi(x)}$$

Kernel:

$K(x, x') = \phi(x) \cdot \phi(x')$

Kernel Perceptron

(0) Initialize: $\theta = 0$ (vector), $\theta_0 = 0$

(1) Cycle through training examples

$t = 1, \ldots, n$:

If $y^{(t)}(\theta \cdot \phi(x) + \theta_0) \leq 0$:

$\theta \leftarrow \theta + y^{(t)} \phi(x^{(t)})$

$\theta_0 \leftarrow \theta_0 + y^{(t)}$

what we would do if we explicitly calculated $\phi(x^{(t)})$

But, $\phi(x^{(t)})$ might be costly to calculate. Let's try to avoid having to explicitly calculate $\phi(x^{(t)})$.

Notice that θ is only updated when a mistake is made, & the update is by $y^{(t)} \phi(x^{(t)})$.

$$\Rightarrow \theta = \sum_{i=1}^{n} \alpha_i y^{(i)} \phi(x^{(i)})$$

α_i: # of mistakes on $(\phi(x^{(i)}), y^{(i)})$

(using same logic,)

$$\theta_0 = \sum_{i=1}^{n} \alpha_i y^{(i)}$$

Consider the discriminant:

$$\underbrace{\theta \cdot \phi(x)}_{\left(\sum_{i=1}^{n} \alpha_i y^{(i)} \underbrace{\phi(x^{(i)})\right) \cdot \phi(x)}_{K(x^{(i)},x)}} + \underbrace{\theta_0}_{\sum_{i=1}^{n} \alpha_i y^{(i)}}$$

$$= \sum_{i=1}^{n} \alpha_i y^{(i)} K(x^{(i)}, x) + \sum_{i=1} \alpha_i y^{(i)}$$

$$= \sum_{i=1}^{n} \alpha_i y^{(i)} (K(x^{(i)}, x) + 1)$$

$\Rightarrow$ new update rule:

$$\text{If } y^{(t)} \left(\sum_{i=1}^{n} \alpha_i y^{(i)} (K(x^{(i)}, x^{(t)}) + 1) \right) \leq 0 :$$

$$\alpha_t \leftarrow \alpha_t + 1$$

note that we never explicitly calculate $\phi(x)$!

· Note: usually only a few α_i are nonzero ; sparse

Kernelized Linear Regression

consider $\theta_0 = 0$,

$$J(\theta) = \frac{1}{n} \sum_{t=1}^{n} \frac{(y^{(t)} - \theta \cdot \phi(x^{(t)}))^2}{2} + \frac{\lambda}{2} \|\theta\|^2$$

$$\nabla_\theta J(\theta) = \frac{-1}{n} \sum_{t=1}^{n} \underbrace{(y^{(t)} - \theta \cdot \phi(x^{(t)}))}_{\text{Define } n\lambda\alpha_t = y^{(t)} - \theta \cdot \phi(x^{(t)})} \cdot \phi(x^{(t)}) + \lambda\theta$$

try to solve for closed form by setting gradient to 0 $\Rightarrow$ $0 = \frac{-1}{n} \sum_{t=1}^{n} n\lambda\alpha_t \phi(x^{(t)}) + \lambda\theta \Rightarrow \theta = \sum_{i=1}^{n} \alpha_i \phi(x^{(i)})$

plugging in to definition,

$$n\lambda\alpha = y^{(t)} - \theta\cdot\phi(x^{(t)})$$

$$= y^{(t)} - \left(\sum_{i=1}^{n} \alpha_i \phi(x^{(i)})\right)\cdot\phi(x^{(t)})$$

$$= y^{(t)} - \underbrace{\sum_{i=1}^{n} \alpha_i K(x^{(i)}, x^{(t)})}_{[K\vec{\alpha}]_t}$$

where $K_{i,j} = K(x^{(i)}, x^{(j)})$, aka K is the Gram matrix

$$\Rightarrow n\lambda\vec{\alpha} = \vec{y} - K\vec{\alpha}$$

$$\Rightarrow (n\lambda I + K)\vec{\alpha} = \vec{y}$$

$$\Rightarrow \vec{\alpha} = \underbrace{(n\lambda I + K)^{-1}}_{\text{always invertible for } \lambda > 0}\vec{y}$$

$$\Rightarrow \theta\cdot\phi(x) = \sum_{i=1}^{n} \hat{\alpha}_i \phi(x^{(i)})\cdot\phi(x) = \sum_{i=1}^{n} \hat{\alpha}_i K(x^{(i)}, x)$$

Example: learn predictor w/ kernel $K(x, x') = x\cdot x' + (x\cdot x')^2$, & data

i	$x^{(i)}$	$y^{(i)}$
1	(1, 3)	+1
2	(1, -1)	-1
3	(1, -2)	+1

(let $\lambda = 1$)

$$\Rightarrow \vec{\alpha} = \left(3\cdot\lambda I + \begin{bmatrix} 90 & 2 & 20 \\ 2 & 6 & 12 \\ 20 & 12 & 30 \end{bmatrix}\right)^{-1} \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 93 & 2 & 20 \\ 2 & 6 & 12 \\ 20 & 12 & 30 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.0154 \\ -0.3117 \\ 0.1530 \end{bmatrix}$$

$$\theta\cdot\phi(x) = \sum_{i=1}^{3} \alpha_i K(x^{(i)}, x)$$

Let's see how we do on our examples:

i	$x^{(i)}$	$\theta\cdot\phi(x^{(i)})$
1	(1, 3)	0.737
2	(1, -1)	-0.0649
3	(1, -2)	0.5410

$\Rightarrow$ correct predictions (signs match)

Kernel Function Properties

valid $\iff \exists \phi(x)$ s.t. $K(x,x') = \phi(x)\cdot\phi(x')$

1) $K(x,x') = 1$ is valid

2) $f: \mathbb{R}^d \to \mathbb{R}$,

   $\tilde{K}(x,x') = f(x)K(x,x')f(x')$ is valid

3) $K_1(x,x') + K_2(x,x')$ is valid

4) $K_1(x,x')\,K_2(x,x')$ is valid

Now let's build up the radial basis kernel: (look to the properties to see why each is valid)

$K(x,x') = x\cdot x'$ is a valid kernel

$\Rightarrow (x\cdot x') + (x\cdot x')^2$ is valid

$\Rightarrow 1 + (x\cdot x') + \frac{1}{2!}(x\cdot x')^2 + \dots = e^{x\cdot x'}$ is valid

$\Rightarrow$ radial basis kernel:

$$K(x,x') = e^{-\|x-x'\|^2/2} = e^{\frac{-\|x\|^2}{2} + x\cdot x' - \frac{\|x'\|^2}{2}}$$

$$= \underbrace{e^{-\|x\|^2/2}}_{f(x)}\; e^{x\cdot x'}\; \underbrace{e^{(-\|x'\|^2/2)}}_{f(x')}$$ is a valid kernel

↳ note that this corresponds to ∞-dimensional $\phi(x)$!

Radial basis kernel definition:

$$K(x,x') = e^{-\gamma\|x-x'\|^2}$$

γ: "bandwidth",

$\uparrow\gamma \Rightarrow \uparrow$ "squiggliness"