COMS 4771 Lecture 24

1. Collaborative filtering
2. Stochastic gradient descent

Collaborative filtering

Netflix problem: 480000 users, 18000 movies.

Recommender systems

Netflix problem: 480000 users, 18000 movies.

- Each user rates some subset of the movies with a score in $\{1, 2, \dots, 5\}$.

  On average, each user rates around 200 movies, though the variance is high (e.g., some user has rated >17000 movies).

Recommender systems

Netflix problem: 480000 users, 18000 movies.

- Each user rates some subset of the movies with a score in $\{1, 2, \ldots, 5\}$.

  On average, each user rates around 200 movies, though the variance is high (e.g., some user has rated >17000 movies).

- Goal is to predict how users would rate movies they haven't seen.

Recommender systems

Netflix problem: 480000 users, 18000 movies.

- Each user rates some subset of the movies with a score in $\{1, 2, \ldots, 5\}$.

  On average, each user rates around 200 movies, though the variance is high (e.g., some user has rated >17000 movies).

- Goal is to predict how users would rate movies they haven't seen.

- Common to reduce $\{1, 2, \ldots, 5\}$ to $\{-1, +1\}$ (e.g., $\{4, 5\} \mapsto +1$).

Recommender systems

Netflix problem: 480000 users, 18000 movies.

- Each user rates some subset of the movies with a score in $\{1, 2, \ldots, 5\}$.

  On average, each user rates around 200 movies, though the variance is high (e.g., some user has rated >17000 movies).

- Goal is to predict how users would rate movies they haven't seen.

- Common to reduce $\{1, 2, \ldots, 5\}$ to $\{-1, +1\}$ (e.g., $\{4, 5\} \mapsto +1$).

Common supervised learning approach:

- Get *features* for each user i and movie j (e.g., $\boldsymbol{x}_i \in \mathbb{R}^d$ and $\boldsymbol{y}_j \in \mathbb{R}^d$); goal is to predict rating as function of $(\boldsymbol{x}_i, \boldsymbol{y}_j)$.

Recommender systems

Netflix problem: 480000 users, 18000 movies.

- Each user rates some subset of the movies with a score in $\{1, 2, \ldots, 5\}$.

  On average, each user rates around 200 movies, though the variance is high (e.g., some user has rated >17000 movies).

- Goal is to predict how users would rate movies they haven't seen.

- Common to reduce $\{1, 2, \ldots, 5\}$ to $\{-1, +1\}$ (e.g., $\{4, 5\} \mapsto +1$).

Common supervised learning approach:

- Get *features* for each user i and movie j (e.g., $\boldsymbol{x}_i \in \mathbb{R}^d$ and $\boldsymbol{y}_j \in \mathbb{R}^d$); goal is to predict rating as function of $(\boldsymbol{x}_i, \boldsymbol{y}_j)$.

- **Linear function**: $\boldsymbol{x}_i \boldsymbol{y}_j^\top \mapsto \langle \boldsymbol{W}, \boldsymbol{x}_i \boldsymbol{y}_j^\top \rangle = \boldsymbol{x}_i^\top \boldsymbol{W} \boldsymbol{y}_j$ for $\boldsymbol{W} \in \mathbb{R}^{d \times d}$.
  Can use SVM, logistic regression, boosting, etc.

Recommender systems

Netflix problem: 480000 users, 18000 movies.

- Each user rates some subset of the movies with a score in $\{1, 2, \dots, 5\}$.

  On average, each user rates around 200 movies, though the variance is high (e.g., some user has rated >17000 movies).

- Goal is to predict how users would rate movies they haven't seen.

- Common to reduce $\{1, 2, \dots, 5\}$ to $\{-1, +1\}$ (e.g., $\{4, 5\} \mapsto +1$).

Common supervised learning approach:

- Get *features* for each user i and movie j (e.g., $\boldsymbol{x}_i \in \mathbb{R}^d$ and $\boldsymbol{y}_j \in \mathbb{R}^d$); goal is to predict rating as function of $(\boldsymbol{x}_i, \boldsymbol{y}_j)$.

- **Linear function**: $\boldsymbol{x}_i \boldsymbol{y}_j^\top \mapsto \boxed{\langle \boldsymbol{W}, \boldsymbol{x}_i \boldsymbol{y}_j^\top \rangle} = \boldsymbol{x}_i^\top \boldsymbol{W} \boldsymbol{y}_j$ for $\boldsymbol{W} \in \mathbb{R}^{d \times d}$.
  Can use SVM, logistic regression, boosting, etc.

What if you don't have any features?

Collaborative filtering:

- Users who rate the same movie similarly are likely to be similar.
- Movies that are rated similarly by the same user are likely to be similar.

Collaborative filtering

Collaborative filtering:

- Users who rate the same movie similarly are likely to be similar.
- Movies that are rated similarly by the same user are likely to be similar.

Upshot: there may be a wealth of information just in the ratings themselves ...

Collaborative filtering

what a genius idea???
get the information from ratings
themselves

Collaborative filtering:

- Users who rate the same movie similarly are likely to be similar.
- Movies that are rated similarly by the same user are likely to be similar.

Upshot: there may be a wealth of information just in the ratings themselves . . .

How can we effectively formalize the intuition stated above?

Simple latent factor model

Simple latent factor model:

Simple latent factor model:

- Each user i has some *hidden* feature vector $\boldsymbol{u}_i \in \mathbb{R}^k$.

Simple latent factor model

Simple latent factor model:

- Each user i has some *hidden* feature vector $\boldsymbol{u}_i \in \mathbb{R}^k$.
- Each movie j has some *hidden* feature vector $\boldsymbol{v}_j \in \mathbb{R}^k$.

Simple latent factor model:

- Each user i has some *hidden* feature vector $\boldsymbol{u}_i \in \mathbb{R}^k$.
- Each movie j has some *hidden* feature vector $\boldsymbol{v}_j \in \mathbb{R}^k$.
- Rating $a_{i,j}$ assigned by user i to movie j is, in expectation, $\langle \boldsymbol{u}_i, \boldsymbol{v}_j \rangle$.

$$\mathbb{E}\left\{\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}\right\} = \begin{bmatrix} — & \boldsymbol{u}_1^\top & — \\ — & \boldsymbol{u}_2^\top & — \\ & \vdots & \\ — & \boldsymbol{u}_m^\top & — \end{bmatrix} \begin{bmatrix} | & | & & | \\ \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_n \\ | & | & & | \end{bmatrix}$$

Simple latent factor model

Simple latent factor model:

- Each user i has some *hidden* feature vector $\boldsymbol{u}_i \in \mathbb{R}^k$.
- Each movie j has some *hidden* feature vector $\boldsymbol{v}_j \in \mathbb{R}^k$.
- Rating $a_{i,j}$ assigned by user i to movie j is, in expectation, $\langle \boldsymbol{u}_i, \boldsymbol{v}_j \rangle$.

$$\mathbb{E}\left\{ \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} \right\} = \begin{bmatrix} \text{---} & \boldsymbol{u}_1^\top & \text{---} \\ \text{---} & \boldsymbol{u}_2^\top & \text{---} \\ & \vdots & \\ \text{---} & \boldsymbol{u}_m^\top & \text{---} \end{bmatrix} \begin{bmatrix} | & | & & | \\ \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_n \\ | & | & & | \end{bmatrix}$$

great way to understand matrix!!!

We'll assume $k \leq \min\{m, n\}$ for reasons that will become clear later.

Complete ratings

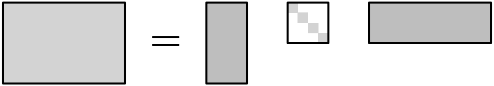
$$\mathbb{E}\left\{\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}\right\} = \begin{bmatrix} — & \boldsymbol{u}_1^\top & — \\ — & \boldsymbol{u}_2^\top & — \\ & \vdots & \\ — & \boldsymbol{u}_m^\top & — \end{bmatrix} \begin{bmatrix} | & | & & | \\ \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_n \\ | & | & & | \end{bmatrix}$$

Suppose every user rates every movie.

Complete ratings

$$\mathbb{E}\left\{\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}\right\} = \begin{bmatrix} — & \boldsymbol{u}_1^\top & — \\ — & \boldsymbol{u}_2^\top & — \\ & \vdots & \\ — & \boldsymbol{u}_m^\top & — \end{bmatrix} \begin{bmatrix} | & | & & | \\ \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_n \\ | & | & & | \end{bmatrix}$$

Suppose every user rates every movie. **Can use SVD**:



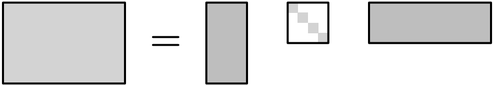
$\boldsymbol{A}$ $(m \times n)$ = $\boldsymbol{L}$ $(m \times r)$ $\boldsymbol{S}$ $(r \times r)$ $\boldsymbol{R}^\top$ $(r \times n)$

Use rows of $\boldsymbol{L}\boldsymbol{S}^{1/2}$ as the $\boldsymbol{u}_i$, and rows of $\boldsymbol{R}\boldsymbol{S}^{1/2}$ as the $\boldsymbol{v}_j$.

Complete ratings

$$\mathbb{E}\left\{\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}\right\} = \begin{bmatrix} \text{---} & \boldsymbol{u}_1^\top & \text{---} \\ \text{---} & \boldsymbol{u}_2^\top & \text{---} \\ & \vdots & \\ \text{---} & \boldsymbol{u}_m^\top & \text{---} \end{bmatrix} \begin{bmatrix} | & | & & | \\ \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_n \\ | & | & & | \end{bmatrix}$$

Suppose every user rates every movie. **Can use SVD**:



$$\underset{(m \times n)}{\boldsymbol{A}} \qquad \underset{(m \times r)}{\boldsymbol{L}} \quad \underset{(r \times r)}{\boldsymbol{S}} \quad \underset{(r \times n)}{\boldsymbol{R}^\top}$$

Use rows of $\boldsymbol{L}\boldsymbol{S}^{1/2}$ as the $\boldsymbol{u}_i$, and rows of $\boldsymbol{R}\boldsymbol{S}^{1/2}$ as the $\boldsymbol{v}_j$.

Unclear what use this is (e.g., if a new movie comes along, can we predict how users will rate it?).

Known user features

$$\mathbb{E}\left\{\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}\right\} = \begin{bmatrix} — & \boldsymbol{u}_1^\top & — \\ — & \boldsymbol{u}_2^\top & — \\ & \vdots & \\ — & \boldsymbol{u}_m^\top & — \end{bmatrix} \begin{bmatrix} | & | & & | \\ \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_n \\ | & | & & | \end{bmatrix}$$

Suppose the users' features were *not* hidden.

Known user features

$$\mathbb{E}\left\{\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}\right\} = \begin{bmatrix} — & \boldsymbol{u}_1^\top & — \\ — & \boldsymbol{u}_2^\top & — \\ & \vdots & \\ — & \boldsymbol{u}_m^\top & — \end{bmatrix} \begin{bmatrix} | & | & & | \\ \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_n \\ | & | & & | \end{bmatrix}$$

Suppose the users' features were *not* hidden.

Could try to *estimate* each movie vector $\boldsymbol{v}_j$ using linear regression:

$$\begin{bmatrix} a_{1,j} \\ a_{2,j} \\ \vdots \\ a_{m,j} \end{bmatrix} = \begin{bmatrix} — & \boldsymbol{u}_1^\top & — \\ — & \boldsymbol{u}_2^\top & — \\ & \vdots & \\ — & \boldsymbol{u}_m^\top & — \end{bmatrix} \begin{bmatrix} | \\ \boldsymbol{v}_j \\ | \end{bmatrix} + \begin{bmatrix} \varepsilon_{1,j} \\ \varepsilon_{2,j} \\ \vdots \\ \varepsilon_{m,j} \end{bmatrix}$$

Known user features

$$\mathbb{E}\left\{\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}\right\} = \begin{bmatrix} \text{---} & \boldsymbol{u}_1^\top & \text{---} \\ \text{---} & \boldsymbol{u}_2^\top & \text{---} \\ & \vdots & \\ \text{---} & \boldsymbol{u}_m^\top & \text{---} \end{bmatrix} \begin{bmatrix} | & | & & | \\ \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_n \\ | & | & & | \end{bmatrix}$$

Suppose the users' features were *not* hidden.

Could try to *estimate* each movie vector $\boldsymbol{v}_j$ using linear regression:

$$\begin{bmatrix} a_{1,j} \\ a_{2,j} \\ \vdots \\ a_{m,j} \end{bmatrix} = \begin{bmatrix} \text{---} & \boldsymbol{u}_1^\top & \text{---} \\ \text{---} & \boldsymbol{u}_2^\top & \text{---} \\ & \vdots & \\ \text{---} & \boldsymbol{u}_m^\top & \text{---} \end{bmatrix} \begin{bmatrix} | \\ \boldsymbol{v}_j \\ | \end{bmatrix} + \begin{bmatrix} \varepsilon_{1,j} \\ \varepsilon_{2,j} \\ \vdots \\ \varepsilon_{m,j} \end{bmatrix}$$

The catch: we don't observe all of the entries of $\boldsymbol{A}$.

Minimize error on training data

Let $\Omega \subseteq [m] \times [n]$ be the entries for which $a_{i,j}$ is observed.
Regard these $a_{i,j}$ as the training data.

Minimize error on training data

Let $\Omega \subseteq [m] \times [n]$ be the entries for which $a_{i,j}$ is observed.
Regard these $a_{i,j}$ as the training data.

Training obj.: find $\boldsymbol{U} = [\boldsymbol{u}_1 | \dots | \boldsymbol{u}_m] \in \mathbb{R}^{k \times m}$ and $\boldsymbol{V} = [\boldsymbol{v}_1 | \dots | \boldsymbol{v}_n] \in \mathbb{R}^{k \times n}$ with as small squared error as possible:

$$f(\boldsymbol{U}, \boldsymbol{V}) \ := \ \sum_{(i,j) \in \Omega} \big(a_{i,j} - \langle \boldsymbol{u}_i, \boldsymbol{v}_j \rangle\big)^2 .$$

Minimize error on training data

Let $\Omega \subseteq [m] \times [n]$ be the entries for which $a_{i,j}$ is observed.
Regard these $a_{i,j}$ as the training data.

Training obj.: find $\boldsymbol{U} = [\boldsymbol{u}_1 | \dots | \boldsymbol{u}_m] \in \mathbb{R}^{k \times m}$ and $\boldsymbol{V} = [\boldsymbol{v}_1 | \dots | \boldsymbol{v}_n] \in \mathbb{R}^{k \times n}$ with as small squared error as possible:

$$f(\boldsymbol{U}, \boldsymbol{V}) \ := \sum_{(i,j) \in \Omega} \big(a_{i,j} - \langle \boldsymbol{u}_i, \boldsymbol{v}_j \rangle \big)^2 .$$

Often called (low rank) matrix completion, because this is equivalent to:

$$\begin{aligned} \min_{\boldsymbol{X} \in \mathbb{R}^{m \times n}} \quad & \sum_{(i,j) \in \Omega} (a_{i,j} - X_{i,j})^2 \\ \text{s.t.} \quad & \operatorname{rank}(\boldsymbol{X}) \le k . \end{aligned}$$

Minimize error on training data

Let $\Omega \subseteq [m] \times [n]$ be the entries for which $a_{i,j}$ is observed.
Regard these $a_{i,j}$ as the training data.

Training obj.: find $\boldsymbol{U} = [\boldsymbol{u}_1| \dots |\boldsymbol{u}_m] \in \mathbb{R}^{k \times m}$ and $\boldsymbol{V} = [\boldsymbol{v}_1| \dots |\boldsymbol{v}_n] \in \mathbb{R}^{k \times n}$ with as small squared error as possible:

$$f(\boldsymbol{U}, \boldsymbol{V}) \; := \; \sum_{(i,j) \in \Omega} \big(a_{i,j} - \langle \boldsymbol{u}_i, \boldsymbol{v}_j \rangle\big)^2.$$

Often called (low rank) matrix completion, because this is equivalent to:

$$\begin{aligned} \min_{\boldsymbol{X} \in \mathbb{R}^{m \times n}} \quad & \sum_{(i,j) \in \Omega} (a_{i,j} - X_{i,j})^2 \\ \text{s.t.} \quad & \operatorname{rank}(\boldsymbol{X}) \leq k. \end{aligned}$$

Unfortunately, objective is not convex due to rank constraint.

Minimize error on training data

Let $\Omega \subseteq [m] \times [n]$ be the entries for which $a_{i,j}$ is observed.
Regard these $a_{i,j}$ as the training data.

Training obj.: find $\boldsymbol{U} = [\boldsymbol{u}_1 | \dots | \boldsymbol{u}_m] \in \mathbb{R}^{k \times m}$ and $\boldsymbol{V} = [\boldsymbol{v}_1 | \dots | \boldsymbol{v}_n] \in \mathbb{R}^{k \times n}$ with as small squared error as possible:

$$f(\boldsymbol{U}, \boldsymbol{V}) \;:=\; \sum_{(i,j) \in \Omega} \big(a_{i,j} - \langle \boldsymbol{u}_i, \boldsymbol{v}_j \rangle \big)^2 .$$

Often called (low rank) matrix completion, because this is equivalent to:

$$\begin{aligned} \min_{\boldsymbol{X} \in \mathbb{R}^{m \times n}} \quad & \sum_{(i,j) \in \Omega} (a_{i,j} - X_{i,j})^2 \\ \text{s.t.} \quad & \operatorname{rank}(\boldsymbol{X}) \leq k . \end{aligned}$$

Unfortunately, objective is not convex due to rank constraint.

Nevertheless, can derive an alternating minimization algorithm (similar to k-means and dictionary learning) to iteratively update $\boldsymbol{U}$ and $\boldsymbol{V}$.

Alternating minimization

Let $\Omega \subseteq [m] \times [n]$ be the entries for which $a_{i,j}$ is observed.

- Somehow initialize $\boldsymbol{u}_i \in \mathbb{R}^k$ for each $i \in [m]$ and $\boldsymbol{v}_j \in \mathbb{R}^k$ for each $j \in [n]$.
- Repeat until convergence:
  - For each user $i \in [m]$,

$$\boldsymbol{u}_i := \arg\min_{\boldsymbol{u}_i \in \mathbb{R}^k} f(\boldsymbol{U}, \boldsymbol{V})$$

  - For each movie $j \in [n]$,

$$\boldsymbol{v}_j := \arg\min_{\boldsymbol{v}_j \in \mathbb{R}^k} f(\boldsymbol{U}, \boldsymbol{V})$$

Alternating minimization

Let $\Omega \subseteq [m] \times [n]$ be the entries for which $a_{i,j}$ is observed.

- Somehow initialize $\boldsymbol{u}_i \in \mathbb{R}^k$ for each $i \in [m]$ and $\boldsymbol{v}_j \in \mathbb{R}^k$ for each $j \in [n]$.
- Repeat until convergence:
  - For each user $i \in [m]$,

$$\boldsymbol{u}_i := \left(\sum_{j \in [n] : (i,j) \in \Omega} \boldsymbol{v}_j \boldsymbol{v}_j^\top \right)^{-1} \sum_{j \in [n] : (i,j) \in \Omega} a_{i,j} \boldsymbol{v}_j$$

  - For each movie $j \in [n]$,

$$\boldsymbol{v}_j := \left(\sum_{i \in [m] : (i,j) \in \Omega} \boldsymbol{u}_i \boldsymbol{u}_i^\top \right)^{-1} \sum_{i \in [m] : (i,j) \in \Omega} a_{i,j} \boldsymbol{u}_i .$$

Alternating minimization

Let $\Omega \subseteq [m] \times [n]$ be the entries for which $a_{i,j}$ is observed.

- Somehow initialize $\boldsymbol{u}_i \in \mathbb{R}^k$ for each $i \in [m]$ and $\boldsymbol{v}_j \in \mathbb{R}^k$ for each $j \in [n]$.
- Repeat until convergence:
  - For each user $i \in [m]$,

$$\boldsymbol{u}_i := \left(\sum_{j \in [n] : (i,j) \in \Omega} \boldsymbol{v}_j \boldsymbol{v}_j^\top \right)^{-1} \sum_{j \in [n] : (i,j) \in \Omega} a_{i,j} \boldsymbol{v}_j$$

  - For each movie $j \in [n]$,

$$\boldsymbol{v}_j := \left(\sum_{i \in [m] : (i,j) \in \Omega} \boldsymbol{u}_i \boldsymbol{u}_i^\top \right)^{-1} \sum_{i \in [m] : (i,j) \in \Omega} a_{i,j} \boldsymbol{u}_i .$$

This assumes certain matrices involving $\boldsymbol{u}_i$ and $\boldsymbol{v}_j$ are invertible—but there's no guarantee of this!

Minimize regularized training error

Let $\Omega \subseteq [m] \times [n]$ be the entries for which $a_{i,j}$ is observed.

Regularized training objective:

$$f(\boldsymbol{U}, \boldsymbol{V}) \ := \ \sum_{(i,j)\in\Omega} \big(a_{i,j} - \langle \boldsymbol{u}_i, \boldsymbol{v}_j \rangle\big)^2 + \lambda \left(\sum_{i=1}^{m} \|\boldsymbol{u}_i\|_2^2 + \sum_{j=1}^{n} \|\boldsymbol{v}_j\|_2^2 \right).$$

Minimize regularized training error

Let $\Omega \subseteq [m] \times [n]$ be the entries for which $a_{i,j}$ is observed.

Regularized training objective:

$$f(\boldsymbol{U}, \boldsymbol{V}) \ := \ \sum_{(i,j)\in\Omega} \big(a_{i,j} - \langle \boldsymbol{u}_i, \boldsymbol{v}_j \rangle\big)^2 + \lambda \left(\sum_{i=1}^{m} \|\boldsymbol{u}_i\|_2^2 + \sum_{j=1}^{n} \|\boldsymbol{v}_j\|_2^2 \right).$$

New updates:

- For each user $i \in [m]$,

$$\boldsymbol{u}_i := \left(\sum_{j\in[n]:(i,j)\in\Omega} \boldsymbol{v}_j \boldsymbol{v}_j^\top + \lambda \boldsymbol{I} \right)^{-1} \sum_{j\in[n]:(i,j)\in\Omega} a_{i,j} \boldsymbol{v}_j$$

- (And similar for updating the $\boldsymbol{v}_j$.)

Minimize regularized training error

Let $\Omega \subseteq [m] \times [n]$ be the entries for which $a_{i,j}$ is observed.

Regularized training objective:

$$f(\boldsymbol{U}, \boldsymbol{V}) := \sum_{(i,j)\in\Omega} \left(a_{i,j} - \langle \boldsymbol{u}_i, \boldsymbol{v}_j \rangle\right)^2 + \lambda \left(\sum_{i=1}^{m} \|\boldsymbol{u}_i\|_2^2 + \sum_{j=1}^{n} \|\boldsymbol{v}_j\|_2^2 \right).$$

New updates:

- For each user $i \in [m]$,

invertiable

$$\boldsymbol{u}_i := \left(\sum_{j\in[n]:(i,j)\in\Omega} \boldsymbol{v}_j \boldsymbol{v}_j^\top \boxed{+ \lambda \boldsymbol{I}} \right)^{-1} \sum_{j\in[n]:(i,j)\in\Omega} a_{i,j} \boldsymbol{v}_j$$

- (And similar for updating the $\boldsymbol{v}_j$.)

Prediction: For a movie j that user i has not rated, predict rating to be

$$\hat{a}_{i,j} := \langle \boldsymbol{u}_i, \boldsymbol{v}_j \rangle.$$

Other things

- **Initialization**: can't initialize with all $\boldsymbol{u}_i = \boldsymbol{v}_j = \boldsymbol{0}$. (Try random.)
- How to pick k or λ?

Other things

- **Initialization**: can't initialize with all $\boldsymbol{u}_i = \boldsymbol{v}_j = \mathbf{0}$. (Try random.)
- How to pick k or λ?

  Answer is obvious by now, but there are some subtle issues.

Other things

- **Initialization**: can't initialize with all $\boldsymbol{u}_i = \boldsymbol{v}_j = \mathbf{0}$. (Try random.)
- How to pick k or λ?

  Answer is obvious by now, but there are some subtle issues.

  - If $\lambda = 0$, then for updates to work, every user must have seen at least k movies, and every movie must have been rated by at least k users.

Other things

- **Initialization**: can't initialize with all $\boldsymbol{u}_i = \boldsymbol{v}_j = \mathbf{0}$. (Try random.)
- How to pick k or λ?

  Answer is obvious by now, but there are some subtle issues.

  - If $\lambda = 0$, then for updates to work, every user must have seen at least k movies, and every movie must have been rated by at least k users.
  - If $\lambda > 0$, then technically updates work for any k.

Other things

- **Initialization**: can't initialize with all $\boldsymbol{u}_i = \boldsymbol{v}_j = \mathbf{0}$. (Try random.)
- How to pick k or λ?

  Answer is obvious by now, but there are some subtle issues.

  - If $\lambda = 0$, then for updates to work, every user must have seen at least k movies, and every movie must have been rated by at least k users.
  - If $\lambda > 0$, then technically updates work for any k.
  - Perhaps should have separate λ for each user/movie, and perhaps it should change as the $\boldsymbol{u}_i$ and $\boldsymbol{v}_j$ change!

Other things

- **Initialization**: can't initialize with all $\boldsymbol{u}_i = \boldsymbol{v}_j = \mathbf{0}$. (Try random.)

- How to pick k or λ?

  Answer is obvious by now, but there are some subtle issues.

  - If $\lambda = 0$, then for updates to work, every user must have seen at least k movies, and every movie must have been rated by at least k users.
  - If $\lambda > 0$, then technically updates work for any k.
  - Perhaps should have separate λ for each user/movie, and perhaps it should change as the $\boldsymbol{u}_i$ and $\boldsymbol{v}_j$ change!

- Incorporating biases

$$f(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{b}, \boldsymbol{c}, \mu) \;:=\; \sum_{(i,j)\in\Omega} \big(a_{i,j} - \langle \boldsymbol{u}_i, \boldsymbol{v}_j \rangle - b_i - c_j - \mu\big)^2 + \lambda\left(\sum_{i=1}^{m} \|\boldsymbol{u}_i\|_2^2 + \sum_{j=1}^{n} \|\boldsymbol{v}_j\|_2^2\right).$$

What about different loss functions?

Different training objective: logistic loss (each $a_{i,j} \in \{-1, +1\}$)

$$f(\boldsymbol{U}, \boldsymbol{V}) \;:=\; \sum_{(i,j)\in\Omega} \ell_{\log}(a_{i,j}\langle \boldsymbol{u}_i, \boldsymbol{v}_j\rangle).$$

No closed form minimizer of $\boldsymbol{U} \mapsto f(\boldsymbol{U}, \boldsymbol{V})$.

What about different loss functions?

Different training objective: logistic loss (each $a_{i,j} \in \{-1,+1\}$)

$$f(\boldsymbol{U},\boldsymbol{V}) \;:=\; \sum_{(i,j)\in\Omega} \ell_{\log}(a_{i,j}\langle \boldsymbol{u}_i, \boldsymbol{v}_j\rangle).$$

No closed form minimizer of $\boldsymbol{U} \mapsto f(\boldsymbol{U},\boldsymbol{V})$.

Alternating gradient updates

For $t = 1, 2, \dots$:

- For each user $i \in [m]$,

$$\boldsymbol{u}_i^{(t+1)} \;:=\; \boldsymbol{u}_i^{(t)} - \eta \nabla_{\boldsymbol{u}_i}\left\{ f(\boldsymbol{U}^{(t)}, \boldsymbol{V}^{(t)}) \right\}.$$

- For each movie $j \in [n]$,

$$\boldsymbol{v}_j^{(t+1)} \;:=\; \boldsymbol{v}_j^{(t)} - \eta \nabla_{\boldsymbol{v}_j}\left\{ f(\boldsymbol{U}^{(t+1)}, \boldsymbol{V}^{(t)}) \right\}.$$

What about different loss functions?

Different training objective: logistic loss (each $a_{i,j} \in \{-1,+1\}$)

$$f(\boldsymbol{U},\boldsymbol{V}) \;:=\; \sum_{(i,j)\in\Omega} \ell_{\log}(a_{i,j}\langle \boldsymbol{u}_i, \boldsymbol{v}_j\rangle).$$

No closed form minimizer of $\boldsymbol{U} \mapsto f(\boldsymbol{U},\boldsymbol{V})$.

Alternating gradient updates

For $t = 1, 2, \ldots$:

- For each user $i \in [m]$,

$$\boldsymbol{u}_i^{(t+1)} \;:=\; \boldsymbol{u}_i^{(t)} + \eta_t \sum_{j\in[n]:(i,j)\in\Omega} a_{i,j} \frac{1}{1+\exp(a_{i,j}\langle \boldsymbol{u}_i^{(t)}, \boldsymbol{v}_j^{(t)}\rangle)} \boldsymbol{v}_j^{(t)}.$$

- For each movie $j \in [n]$,

$$\boldsymbol{v}_j^{(t+1)} \;:=\; \boldsymbol{v}_j^{(t)} + \eta_t \sum_{i\in[m]:(i,j)\in\Omega} a_{i,j} \frac{1}{1+\exp(a_{i,j}\langle \boldsymbol{u}_i^{(t)}, \boldsymbol{v}_j^{(t)}\rangle)} \boldsymbol{u}_i^{(t)}.$$

What about different loss functions?

Different training objective: logistic loss (each $a_{i,j} \in \{-1,+1\}$)

$$f(\boldsymbol{U},\boldsymbol{V}) \;:=\; \sum_{(i,j)\in\Omega} \ell_{\log}(a_{i,j}\langle \boldsymbol{u}_i, \boldsymbol{v}_j\rangle).$$

No closed form minimizer of $\boldsymbol{U} \mapsto f(\boldsymbol{U},\boldsymbol{V})$.

Alternating gradient updates

For $t = 1, 2, \dots$:

- For each user $i \in [m]$,

$$\boldsymbol{u}_i^{(t+1)} \;:=\; \boldsymbol{u}_i^{(t)} + \eta_t \sum_{j\in[n]:(i,j)\in\Omega} a_{i,j} \frac{1}{1+\exp(a_{i,j}\langle \boldsymbol{u}_i^{(t)}, \boldsymbol{v}_j^{(t)}\rangle)} \boldsymbol{v}_j^{(t)}.$$

- For each movie $j \in [n]$,

$$\boldsymbol{v}_j^{(t+1)} \;:=\; \boldsymbol{v}_j^{(t)} + \eta_t \sum_{i\in[m]:(i,j)\in\Omega} a_{i,j} \frac{1}{1+\exp(a_{i,j}\langle \boldsymbol{u}_i^{(t)}, \boldsymbol{v}_j^{(t)}\rangle)} \boldsymbol{u}_i^{(t)}.$$

(Ambiguous whether to use $\boldsymbol{u}_i^{(t)}$ or $\boldsymbol{u}_i^{(t+1)}$ when updating $\boldsymbol{v}_j^{(t)} \to \boldsymbol{v}_j^{(t+1)}$.)

Stochastic gradient descent

Unconstrained convex optimization problem

$$\min_{\boldsymbol{w} \in \mathbb{R}^d} \quad f(\boldsymbol{x})$$

(f is the convex objective function).

Unconstrained convex optimization problem

$$\min_{\boldsymbol{w} \in \mathbb{R}^d} \quad f(\boldsymbol{x})$$

(f is the convex objective function).

Gradient descent for differentiable objectives

- Start with some initial $\boldsymbol{w}^{(1)} \in \mathbb{R}^d$.
- For $t = 1, 2, \ldots$ until some stopping condition is satisfied.
  - Compute gradient of f at $\boldsymbol{w}^{(t)}$:

    $$\boldsymbol{\lambda}^{(t)} := \nabla f(\boldsymbol{w}^{(t)}).$$

  - Update:

    $$\boldsymbol{w}^{(t+1)} := \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{\lambda}^{(t)}.$$

Unconstrained convex optimization

Unconstrained convex optimization problem

$$\min_{\boldsymbol{w} \in \mathbb{R}^d} \quad f(\boldsymbol{x})$$

(f is the convex objective function).

Gradient descent for differentiable objectives

- Start with some initial $\boldsymbol{w}^{(1)} \in \mathbb{R}^d$.
- For $t = 1, 2, \ldots$ until some stopping condition is satisfied.
  - Compute gradient of f at $\boldsymbol{w}^{(t)}$:

$$\boldsymbol{\lambda}^{(t)} := \nabla f(\boldsymbol{w}^{(t)}).$$

  - Update:

$$\boldsymbol{w}^{(t+1)} := \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{\lambda}^{(t)}.$$

If f is "nice", GD converges to global minimizer $\boldsymbol{w}_\star$, where $\nabla f(\boldsymbol{w}_\star) = \mathbf{0}$.

Summation form objectives

In machine learning, we frequently deal with objectives of the form

$$f(\boldsymbol{w}) \ = \ \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{w})$$

where $f_1, f_2, \ldots, f_n \colon \mathbb{R}^d \to \mathbb{R}$ are convex functions (e.g., loss on example i).

Summation form objectives

In machine learning, we frequently deal with objectives of the form

$$f(\boldsymbol{w}) \;=\; \frac{1}{n}\sum_{i=1}^{\boxed{n}} f_i(\boldsymbol{w})$$

where $f_1, f_2, \ldots, f_n \colon \mathbb{R}^d \to \mathbb{R}$ are convex functions (e.g., loss on example i).

Example: **Gradient descent algorithm for logistic regression**

- Start with some initial $\boldsymbol{w}^{(1)} \in \mathbb{R}^d$.
- For $t = 1, 2, \ldots$

$$\begin{aligned} \boldsymbol{w}^{(t+1)} &:= \boldsymbol{w}^{(t)} - \eta_t \nabla f(\boldsymbol{w}^{(t)}) \\ &= \boldsymbol{w}^{(t)} + \eta_t \frac{1}{|S|} \sum_{(\boldsymbol{x},y)\in S} \frac{1}{1 + e^{y\langle \boldsymbol{w}^{(t)}, \boldsymbol{x}\rangle}} y\boldsymbol{x}. \end{aligned}$$

training
loss

Summation form objectives

In fact, we often *really* care about

$$f(\boldsymbol{w}) \;=\; \mathbb{E}\big[f(\boldsymbol{w}; Z)\big]$$

where Z is a random example, and $f(\cdot; Z)$ represents the (convex) loss on this random example.

Summation form objectives

test loss

In fact, we often *really* care about

$$f(\boldsymbol{w}) \ = \ \mathbb{E}\big[f(\boldsymbol{w}; Z)\big]$$

where Z is a random example, and $f(\cdot; Z)$ represents the (convex) loss on this random example.

Let us continue with this for now . . .

Stochastic optimization paradigm

Suppose at the push of a red button, we get an independent draw from the distribution P over labeled examples that we care about.

Stochastic optimization paradigm

Suppose at the push of a red button, we get an independent draw from the distribution P over labeled examples that we care about.

To make things even simpler, before the t-th button push, we must specify our *current weight vector* $\boldsymbol{w}^{(t)}$, and all we get back is

$$\boldsymbol{\lambda}^{(t)} := \nabla f(\boldsymbol{w}^{(t)}; z_t)$$

where z_t is a independent random realization of Z.

Stochastic optimization paradigm

Suppose at the push of a red button, we get an independent draw from the distribution P over labeled examples that we care about.

To make things even simpler, before the t-th button push, we must specify our *current weight vector* $\boldsymbol{w}^{(t)}$, and all we get back is

$$\boldsymbol{\lambda}^{(t)} := \nabla f(\boldsymbol{w}^{(t)}; z_t)$$

where z_t is a independent random realization of Z.

(Gradient of the loss on **a simple random example** at current weight vector.)

Stochastic optimization paradigm

Suppose at the push of a red button, we get an independent draw from the distribution P over labeled examples that we care about.

To make things even simpler, before the t-th button push, we must specify our *current weight vector* $\boldsymbol{w}^{(t)}$, and all we get back is

not overall sample

$$\boldsymbol{\lambda}^{(t)} := \nabla f(\boldsymbol{w}^{\boxed{(t)}}; \boxed{z_t})$$

where z_t is a independent random realization of Z.

(Gradient of the loss on **a simple random example** at current weight vector.)

Can we still find a good weight vector?

Stochastic gradient descent (SGD)

randomly pick one and minimize its cost

- Start with some initial $\boldsymbol{w}^{(1)} \in \mathbb{R}^d$.
- For $t = 1, 2, \ldots$
  - Push the red button and get

$$\boldsymbol{\lambda}^{(t)} := \nabla f(\boldsymbol{w}^{(t)}; z_t).$$

just for one sample point each time

  - Update:

$$\boldsymbol{w}^{(t+1)} := \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{\lambda}^{(t)}.$$

Why does this work?

We care about

$$f(\boldsymbol{w}) \;=\; \mathbb{E}\big[f(\boldsymbol{w}; Z)\big].$$

Why does this work?

We care about

$$f(\boldsymbol{w}) \;=\; \mathbb{E}\big[f(\boldsymbol{w}; Z)\big].$$

We update $\boldsymbol{w}^{(t)}$ using $\boldsymbol{\lambda}^{(t)}$, where

$$\mathbb{E}\Big[\boldsymbol{\lambda}^{(t)}\Big] \;=\; \mathbb{E}\Big[\nabla f(\boldsymbol{w}^{(t)}; z_t)\Big] \;=\; \nabla\Big\{\mathbb{E}\Big[f(\boldsymbol{w}^{(t)}; z_t)\Big]\Big\} \;=\; \nabla f(\boldsymbol{w}^{(t)})$$

- Expectation is only w.r.t. z_t, an independent realization of Z.
- Gradient is with respect to $\boldsymbol{w}$ argument, and is evaluated at $\boldsymbol{w} = \boldsymbol{w}^{(t)}$.

Why does this work?

We care about

$$f(\boldsymbol{w}) \;=\; \mathbb{E}\big[f(\boldsymbol{w}; Z)\big].$$

average. not a random example

We update $\boldsymbol{w}^{(t)}$ using $\boldsymbol{\lambda}^{(t)}$, where

$$\mathbb{E}\Big[\boldsymbol{\lambda}^{(t)}\Big] \;=\; \mathbb{E}\Big[\nabla f(\boldsymbol{w}^{(t)}; z_t)\Big] \;=\; \nabla\Big\{\mathbb{E}\Big[f(\boldsymbol{w}^{(t)}; z_t)\Big]\Big\} \;=\; \nabla f(\boldsymbol{w}^{(t)})$$

- Expectation is only w.r.t. z_t, an independent realization of Z.
- Gradient is with respect to $\boldsymbol{w}$ argument, and is evaluated at $\boldsymbol{w} = \boldsymbol{w}^{(t)}$.

In expectation, we move in the correct direction w.r.t. the objective we actually care about!

online perceptron use this idea

Why does this work?

We care about

$$f(\boldsymbol{w}) \; = \; \mathbb{E}\big[f(\boldsymbol{w}; Z)\big].$$

We update $\boldsymbol{w}^{(t)}$ using $\boldsymbol{\lambda}^{(t)}$, where

$$\mathbb{E}\Big[\boldsymbol{\lambda}^{(t)}\Big] \; = \; \mathbb{E}\Big[\nabla f(\boldsymbol{w}^{(t)}; z_t)\Big] \; = \; \nabla\Big\{\mathbb{E}\Big[f(\boldsymbol{w}^{(t)}; z_t)\Big]\Big\} \; = \; \nabla f(\boldsymbol{w}^{(t)})$$

- Expectation is only w.r.t. z_t, an independent realization of Z.
- Gradient is with respect to $\boldsymbol{w}$ argument, and is evaluated at $\boldsymbol{w} = \boldsymbol{w}^{(t)}$.

In expectation, we move in the correct direction w.r.t. the objective we actually care about!

Convergence is slower than Gradient Descent due to stochasticity.

Why does this work?

We care about

$$f(\boldsymbol{w}) = \mathbb{E}\big[f(\boldsymbol{w}; Z)\big].$$

the expectation of \lambda
(rather than directly calculate)

We update $\boldsymbol{w}^{(t)}$ using $\boldsymbol{\lambda}^{(t)}$, where

$$\mathbb{E}\Big[\boldsymbol{\lambda}^{(t)}\Big] = \mathbb{E}\Big[\nabla f(\boldsymbol{w}^{(t)}; z_t)\Big] = \nabla\Big\{\mathbb{E}\Big[f(\boldsymbol{w}^{(t)}; z_t)\Big]\Big\} = \nabla f(\boldsymbol{w}^{(t)})$$

- Expectation is only w.r.t. z_t, an independent realization of Z.
- Gradient is with respect to $\boldsymbol{w}$ argument, and is evaluated at $\boldsymbol{w} = \boldsymbol{w}^{(t)}$.

In expectation, we move in the correct direction w.r.t. the objective we actually care about!

Convergence is slower than Gradient Descent due to stochasticity.
But it is w.r.t. the objective we actually care about!

Implementing SGD with finite training data

- Assumption: training data is i.i.d. sample from distribution we care about.

Implementing SGD with finite training data

- Assumption: training data is i.i.d. sample from distribution we care about.
- **Randomly permute order of the data.**

Implementing SGD with finite training data

- Assumption: training data is i.i.d. sample from distribution we care about.
- **Randomly permute order of the data.**
- **Pushing the red button**: take the next training example z_t (according to the random order) and use it to compute gradient of $f(\cdot; z_t)$ at $\boldsymbol{w}^{(t)}$.

Implementing SGD with finite training data

- Assumption: training data is i.i.d. sample from distribution we care about.
- **Randomly permute order of the data.**
- **Pushing the red button**: take the next training example z_t (according to the random order) and use it to compute gradient of $f(\cdot; z_t)$ at $\boldsymbol{w}^{(t)}$.
- After making a pass through the data, theory says to stop (afaik)—repeating examples cannot really be regarded as independent.

Implementing SGD with finite training data

- Assumption: training data is i.i.d. sample from distribution we care about.
- **Randomly permute order of the data.**
- **Pushing the red button**: take the next training example z_t (according to the random order) and use it to compute gradient of $f(\cdot; z_t)$ at $\boldsymbol{w}^{(t)}$.
- After making a pass through the data, theory says to stop (afaik)—repeating examples cannot really be regarded as independent.
- In practice, can help to cycle through the data a few times.

there is no risk of overfitting!!!
1. very resistant to overfitting.
2. voted-perceptron

SGD with non-convex objectives

Can also apply SGD to non-convex summation objectives, e.g.,

$$f(\boldsymbol{U}, \boldsymbol{V}) \;:=\; \mathbb{E}\big[\ell_{\log}(a_{I,J}\langle \boldsymbol{u}_I, \boldsymbol{v}_J\rangle)\big]$$

(for random user I and movie J).

SGD WITH NON-CONVEX OBJECTIVES

Can also apply SGD to non-convex summation objectives, e.g.,

$$f(\boldsymbol{U}, \boldsymbol{V}) \;:=\; \mathbb{E}\big[\ell_{\log}(a_{I,J}\langle \boldsymbol{u}_I, \boldsymbol{v}_J\rangle)\big]$$

(for random user I and movie J).

Given ratings $\{a_{i,j} : (i,j) \in \Omega\}$, first randomly permute their order in Ω.

SGD WITH NON-CONVEX OBJECTIVES

push the button

Can also apply SGD to non-convex summation objectives, e.g.,

$$f(\boldsymbol{U}, \boldsymbol{V}) \ := \ \mathbb{E}\left[\ell_{\log}(a_{I,J}\langle \boldsymbol{u}_I, \boldsymbol{v}_J \rangle)\right]$$

(for random user I and movie J).

Given ratings $\{a_{i,j} : (i,j) \in \Omega\}$, first randomly permute their order in Ω.

For $t = 1, 2, \dots$:

- Get next (i,j) and $a_{i,j}$.
- Updates:

$$\boldsymbol{u}_i^{(t+1)} \ := \ \boldsymbol{u}_i^{(t)} + \eta_t a_{i,j} \frac{1}{1 + \exp(a_{i,j}\langle \boldsymbol{u}_i^{(t)}, \boldsymbol{v}_j^{(t)} \rangle)} \boldsymbol{v}_j^{(t)}$$

$$\boldsymbol{v}_j^{(t+1)} \ := \ \boldsymbol{v}_j^{(t)} + \eta_t a_{i,j} \frac{1}{1 + \exp(a_{i,j}\langle \boldsymbol{u}_i^{(t)}, \boldsymbol{v}_j^{(t)} \rangle)} \boldsymbol{u}_i^{(t)}.$$

- (For all other (i',j'), no change: $\boldsymbol{u}_{i'}^{(t+1)} = \boldsymbol{u}_{i'}^{(t)}$ and $\boldsymbol{v}_{j'}^{(t+1)} = \boldsymbol{v}_{j'}^{(t)}$.)

Recap

- **Stochastic gradient descent** (not technically a descent method): for convex problems, converges (somewhat slowly) to the optimum of the problem we actually care about!
- Can also be applied to non-convex problems.
- **A particular non-convex problem**: matrix completion, which is often used for collaborative filtering / recommender systems.