Regression Metrics

- **MSE**: Mean Square Error $\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_t - y_t)^2$
- **MAE**: Mean Absolute Error $\frac{1}{n}\sum_{i=1}^{n}|\hat{y}_t - y_t|$
- **MAPE**: Mean Absolute Percentage Error $\frac{100\%}{n}\sum_{i=1}^{n}|\frac{\hat{y}_t - y_t}{y_t}|$

Classification Metrics

Accuracy

Limitation: Imbalanced data

		Actual	
		True	False
Predicted	True	TP	FP
	False	FN	TN

$$\text{Acc} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

, **Precision and Recall**

PRECISION		Actual	
		True	False
Predicted	True	TP	FP
	False	FN	TN

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}},$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

RECALL		Actual	
		True	False
Predicted	True	TP	FP
	False	FN	TN

- *Precision*: how many predicted positives actually positive?
- *Recall*: how many positives are correctly found?

F1 Score

$$\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- *F1 Score*: harmonic mean of precision and recall.
- *Harmonic Mean*: closer to the smaller value

Linear Regression

Univariate Normation Equation

$$\overline{x} = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}, \overline{y} = \frac{1}{m}\sum_{i=1}^{m} y^{(i)}, \overline{xy} = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}y^{(i)}, \overline{x^2} = \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)}\right)^2$$

$$\theta_0 \text{ (intercept)} = \frac{\overline{y}\cdot\overline{x^2} - \overline{x}\cdot\overline{xy}}{\overline{x^2} - (\overline{x})^2}, \theta_1 \text{ (slope)} = \frac{\overline{xy} - \overline{x}\cdot\overline{y}}{\overline{x^2} - (\overline{x})^2}$$

Multivariate Normation Equation

$$X = \begin{pmatrix} 1 & x_1^{(1)} & \dots & x_j^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & \dots & x_j^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & & & \ddots & \vdots \\ 1 & x_1^{(m)} & \dots & x_j^{(m)} & \dots & x_n^{(m)} \end{pmatrix}, \theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{pmatrix}, y = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{pmatrix}$$

$$(X^T X)\theta = X^T y$$

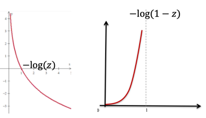
$$\theta = (X^T X)^{-1} X^T y$$

Logistic Regression

Basic Definition: $f_{\theta(x)} = \sigma(\theta^T x) = \frac{1}{1+\exp(-\theta^T x)} \in (0,1)$

Decision Bound: $\theta^T x = 0$, predicts $y = 1$ when $\theta^T x > 0$, vice versa

Logistic Loss: $L(y, f_\theta) = -(y \log f_\theta(x) + (1-y)\log(1 - f_\theta(x)))$

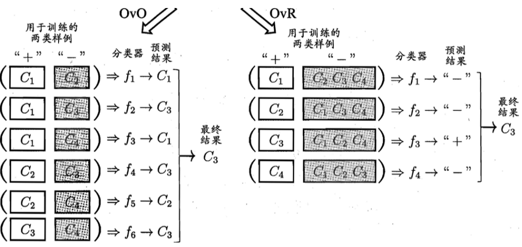


$$L(y, f_\theta) = \begin{cases} -\log f_\theta(x) & \text{if } y = 1 \\ -\log(1 - f_\theta(x)) & \text{if } y = 0 \end{cases}$$

Multi-class Classification

One-vs-One (OvA): Train C_n^2 classifers, use majority voting

One-vs-Rest (OvR): Train C classifers; each treat all other $C-1$ classes as negative. Choose class with largest probability as output.



Support Vector Machine (SVM)

Distance from point to a line $\mathbf{w}^T\mathbf{x} + b = 0$:

$$\text{Distance} = \frac{\mathbf{w}^T\mathbf{x} + b}{\|\mathbf{w}\|}$$

Let training set $\left\{(\mathbf{x}^{(i)}, y_i)\right\}_{i=1\dots n}, y_i \in \{-1, 1\}$ be separated by a hyperplane with margin γ. Then for each training sample $(\mathbf{x}^{(i)}, y_i)$:

$$y_i(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq \gamma/2 \Rightarrow \begin{cases} \mathbf{w}^T\mathbf{x}^{(i)} + b \leq -\gamma/2 & \text{if } y_i = -1 \\ \mathbf{w}^T\mathbf{x}^{(i)} + b \geq \gamma/2 & \text{if } y_i = 1 \end{cases}$$

If we rescale $\mathbf{w}$ and b by $\gamma/2$ in the above equality, we obtain distance between each **support vector** sample $\mathbf{x}^{(s)}$ with the hyperplane:

$$\frac{y_s(\mathbf{w}^T\mathbf{x}^{(s)} + b)}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}, \quad \text{normalized margin } \gamma' = \frac{2}{\|\mathbf{w}\|}$$

Optimization Problem

1. **Maximize Formulation**: Find $\mathbf{w}$ and b s.t. $\gamma' = \frac{2}{\|\mathbf{w}\|}$ is maximized, while for all samples $(\mathbf{x}^{(i)}, y_i)$ satisfies : $y_i(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1$
2. **Minimize Formulation**: Find $\mathbf{w}$ and b s.t. $\|\mathbf{w}\|^2 = \mathbf{w}^T\mathbf{w}$ is minimized, while for all samples $y_i(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1$

Soft Margin SVM

Allow some samples to be misclassified, and instead minimize:

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{subject to} \quad \begin{cases} \mathbf{w}^T\mathbf{x}^{(i)} + b \leq -1 + \xi_i & \text{if } y_i = -1 \\ \mathbf{w}^T\mathbf{x}^{(i)} + b \geq 1 - \xi_i & \text{if } y_i = 1 \\ \xi_i \geq 0 & \text{for all } i \end{cases}$$

ξ_i is the slack variable, and C is the regularization parameter.

- ξ_i is **not** hyperparameter. It is calculated for each training sample.
  - $\xi_i = 0.3$: point "penetrated" 30% way through margin boundary
  - $\xi_i = 1.0$: point sitting exactly on hyperplane (decision boundary)
  - $\xi_i = 1.5$: point crossed hyperplane, and 50% way on the other side
- C is hyperparameter. controls the trade-off

Neural Network

Perceptrons

Step Activation: $y = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$

$$w \leftarrow w - \alpha\frac{\partial L}{\partial w_i} \qquad \because \frac{\partial z}{\partial \mathbf{w}} = \mathbf{x},$$

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial y}\cdot\frac{\partial y}{\partial z}\cdot\frac{\partial z}{\partial \mathbf{w}} \qquad \therefore \frac{\partial L}{\partial \mathbf{w}} = \left(\frac{\partial L}{\partial y}\cdot\frac{\partial y}{\partial z}\right)\cdot\mathbf{x} = \delta\cdot\mathbf{x}$$

Where: δ is the error term, L (Loss): Loss function, α: Learning rate, $\frac{\partial L}{\partial y}$: sensitivity of the loss w.r.t. the activation output, $\frac{\partial y}{\partial z}$: derivative of the activation function

- $\frac{\partial z}{\partial \mathbf{w}}$: since $z = \mathbf{w}^T\mathbf{x} + b$, the derivative w.r.t. to w is input vector $\mathbf{x}$

Linear Activation: *(w/ MSE Loss)*

$$\hat{y} = z = \mathbf{w}^T\mathbf{x} + b; \;\; L(y, \hat{y}) = (y - \hat{y})^2$$

$$\delta = \frac{\partial L}{\partial \hat{y}}\cdot\frac{\partial \hat{y}}{\partial z} = \frac{\partial L}{\partial \hat{y}}\cdot 1 = 2(y - \hat{y})$$

Sigmoid Activation *(w/ Binary Cross Entropy Loss) :*

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$L(y, \hat{y}) = -(y\cdot\log\hat{y} + (1-y)\log(1-\hat{y}))$$

Derivatives:

$$\text{Loss}: \frac{\partial L}{\partial \hat{y}} = -\left(\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right) = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})}$$

$$\text{Activation}: \frac{\partial \hat{y}}{\partial z} = \hat{y}(1-\hat{y}) \textbf{ or } \sigma(z)(1-\sigma(z))$$

$$\text{Error Term}: \delta = \frac{\partial L}{\partial \hat{y}}\cdot\frac{\partial \hat{y}}{\partial z} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})}\cdot\hat{y}(1-\hat{y})$$

$$= \hat{y} - y \textbf{ or } \sigma(z) - y$$

Weight Update *(everything together)*:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha(\sigma(z) - y)\cdot\mathbf{x}$$

Weight Update *(everything together)*:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha(\hat{y}_j - y_j)\cdot\mathbf{x}$$

Regularized Gradient Descent

L2 Regularization

$$J(\theta) = \frac{1}{m}\left[\sum_{i=1}^{m} L(y_i, \hat{y}_i) + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

$$= \frac{1}{m}\sum_{i=1}^{m} L(y_i, \hat{y}_i) + \frac{\lambda}{m}\sum_{j=1}^{n}\theta_j^2$$

$$\therefore \frac{\partial J}{\partial \theta_j} = \frac{1}{m}\left[\sum_{i=1}^{m}\frac{\partial L}{\partial \theta_j} + 2\lambda\theta_j\right]$$

L1 Regularization

$$J = \frac{1}{m}\left[\sum_{i=1}^{m} L(y_i, \hat{y}_i) + \lambda\sum_{j=1}^{n}|\theta_j|\right]$$

$$= \frac{1}{m}\sum_{i=1}^{m} L(y_i, \hat{y}_i) + \frac{\lambda}{m}\sum_{j=1}^{n}|\theta_j|$$

$$\therefore \frac{\partial J}{\partial \theta_j} = \frac{1}{m}\left[\sum_{i=1}^{m}\frac{\partial L}{\partial \theta_j} + \lambda\,\text{sign}(\theta_j)\right]$$

where sign(x) is the sign function: $\text{sign}(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \end{cases}$

CNN (Convolution Neural Network)

Activation

ReLU (Rectified Linear Unit): $\text{ReLU}(x) = \max(0, x)$ Motivations:

- Outputs 0 for negative values, introducing sparse representation
- Does not face vanishing gradient as with sigmoid or tanh
- Fast: does not need e^x computation

Conv Layer

$$n_{\text{out}} = \left\lfloor\frac{n_{\text{in}} + 2p - k}{s}\right\rfloor + 1 = \left\lceil\frac{n_{\text{in}} + 2p - k + 1}{s}\right\rceil$$

where,

- n_{in}: number of input features, n_{out}: number of output features, k: convolution kernel size, p: padding size (on one side), s: convolution stride size

Training w/ Max-pooling

$y_{11} = \max(x_{11}, x_{12}, x_{21}, x_{22})$

Derivative

For values of input feature map $x_i \in \{x_1, x_2, \dots, x_n\}$, where x_i is amongst the inputs of $y_j \in \{y_1, y_2, \dots, y_m\}$

In simpler terms: x is input before max-pooling, y is output after max-pooling, $m < n$ for obvious reasons, x_i can be inputs of multiple y

Then we have,

$$\frac{\partial L}{\partial x_i} = \sum_{y_j \in \substack{\text{all max-pooling} \\ \text{windows convering } x_i}}\frac{\partial L}{\partial y_j}\cdot\frac{\partial y_j}{\partial x_i}$$

The core idea is that,

$$\frac{\partial L}{\partial x_i} = \begin{cases} \frac{\partial L}{\partial y}\cdot\frac{\partial y}{\partial x_i} & \text{if } x_i = y \\ 0 & \text{otherwise} \end{cases}$$

RNN (Recurrent Neural Network)

Vanilla RNN: $h_t = f_\theta(h_{t-1}, x_t)$

where h_t is new state, h_{t-1} is previous state, x_t is input

- **Pros**: Process variable-length sequence
- **Cons**: vanishing or exploding gradient.

Let's assume that the weight $\mathbf{w}$ is scalr, so is h_t. Loss at time T is L_T

$$\frac{\partial L_T}{\partial h_k} = \frac{\partial L_T}{\partial H_T}\frac{\partial H_T}{\partial h_{T-1}}\frac{\partial h_{T-1}}{\partial h_{T-2}}\cdots\frac{\partial h_{k+1}}{\partial h_k} = \frac{\partial L_T}{\partial H_T}(w)^{T-k}$$

- Then any weight value $w \neq 1$ will cause unstable gradient.

LSTM (Long Short-Term Memory):

- `new_state = forget(old_state) + select(new_state)`
- `output = select(new_state)`

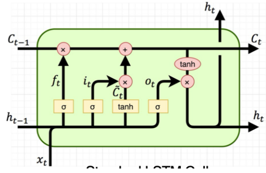
Input Processing:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \in (0,1)$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \in (0,1)$$

$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



Output:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \in (0,1)$$

$$h_t = o_t * \tanh(C_t)$$

- o_t select part of C_t to output based on h_{t-1} and x_t
- h_t is the actual output

Intuition:

- C is long-term memory.
- f_t forgets some old mem in C_{t-1}
- i_t select some new mem in $\tilde{C}_t$
- merge 'forgotten' and 'selected' mem to form new C_t

GRU (Gated Recurrent Unit):

$$z_t = \sigma(W_z[h_{t-1}, x_t]) \in (0,1)$$

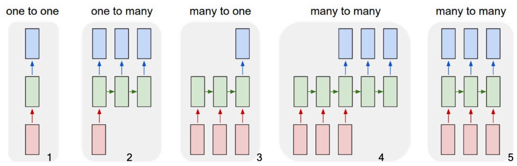
$$r_t = \sigma(W_r[h_{t-1}, x_t]) \in (0,1)$$

$$\tilde{h}_t = \tanh(W_h[r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- r_t forgets some old mem from h_{t-1}
- $\tilde{h}_t$ candidate state made from 'forgotten' old mem $r_t * h_{t-1}$ and x_t
- z_t creates coefficient for mixing candidate $\tilde{h}_t$ and old mem h_{t-1}

Sequence Learning w/ One RNN Layer



1. Standard NN - image classification
2. Sequence output - image captioning
3. Sequence input - Sentiment analysis
4. Sequence input output - machine translation
5. Sync seq input and output - video classification, label each frame

MapReduce

- *Problem*: Process massive data beyond single machine capacity.
- *Solution*: Parallel processing on commodity clusters.
- *MapReduce*: Programming model (like Von-Neumann)
- *User*: Defines `Map` & `Reduce` functions.
- *System*: Handles parallelism, data distribution, fault tolerance, communication

Workflow: `Input` → `Map` → `Shuffle & Sort` → `Reduce` → `Output`

1. **`Input`**: split input data into M splits,
   - each split processed by 1 machine
   - each split contains N records as key-value pairs
2. **`Map`** transforms input k-v to new set of k'-v' pairs
3. **`Shuffle & Sort`** the k'-v' pairs
4. All k'-v' with the same k' grouped together, sent to **same reduce**
5. **`Reduce`** processes all k'-v' into new k''-v'' pairs
6. **`Output`**: write resulting pairs to file

Example: Set difference $A - B$ (i.e., what A has but B doesn't)

```
Map(key, value) {
  # key: split A or B
  # value: elements
  for e in value:
    emit(e, key);
}
```

```
Reduce(key, values) {
  # key: element
  # values: list of names of splits
  if 'A' in values:
    if 'B' not in values:
      emit(key);
}
```

Dealing w/ Failures

- **`Map`** failures:
  - Completed & in-progress tasks on failed worker are reset to idle & rescheduled.
  - Completed `Map` outputs (on local disk of failed worker) are lost; **re-executed on another worker**
- **`Reduce`** failures:
  - In-progress tasks reset & restarted.
  - Completed `Reduce` output to global FS, so **NOT re-executed**

Recommender Systems

Content Based (CB)

Assumption: if past user liked a set of items with certain features, they will continue to like other items with similar features.

- **Pros**:
  - No need for data on other users
  - Able to recommend to unique user taste
  - Able to recommend new & unpopular items
  - Interpretability: provide explanation
- **Cons**:
  - Finding appropriate features is hard
  - Building user profile is slow (e.g., for new users)
  - Overspecialization (don't show diverse content to user)

Approach 1: CB based on Linear Regression

- *Input*: feature vector of item (e.g., movie's style feature vector)
- *Output*: user preference (e.g., scalar user rating value)
- *Model*: each user have their own model $\theta^{(j)}$
  - predicted rating $R_{j,i} = \theta_j^T x^{(i)}$
    - where $x^{(i)}$ is feature vector of item i

Collaborative Filtering (CF)

Assumption: user with similar history likely have similar preference

- **Pros**: Works on any kind of item (no feature selection needed)
- **Cons**:
  - Cold start, need enough user in system to find match
  - Sparsity: user / rating matrix very sparse
  - First rater: cannot recommend item has not been rated
  - Popularity bias: cannot recommend to someone with unique taste

Approach 1: CF based on Linear Regression

- ***Simutaneously*** learn user preference $\theta^{(j)}$ and item feature $x^{(i)}$
- Intuition: factorize $N \times M$ user-item rating matrix into $N \times F$ user-preference matrix and $F \times M$ feature-item matrix

$$J(x,\theta) = \underbrace{\left(\frac{1}{2m_{\text{ratings}}}\right) \sum_{(i,j)\text{ where } r(i,j)=1} \left(\left(\theta^{(j)}\right)^T x^{(i)} - y^{(i,j)} \right)^2}_{\text{Sum of Squared Errors for known ratings}}$$

$$- \underbrace{\left(\frac{\lambda}{2m_{\text{ratings}}}\right) \sum_{i=1}^{N_m} \sum_{k=1}^{n} \left(x_k^{(i)}\right)^2}_{\text{Regularization for Movie Features}}$$

$$- \underbrace{\left(\frac{\lambda}{2m_{\text{ratings}}}\right) \sum_{j=1}^{N_u} \sum_{k=1}^{n} \left(\theta_k^{(j)}\right)^2}_{\text{Regularization for User Preferences}}$$

- $r(i,j) = 1$ if user i has rated item j, 0 otherwise
- $\theta^{(j)} \in \mathbb{R}^F$ latent preference vector of user j **being optimized**
- $x^{(i)} \in \mathbb{R}^F$ latent feature vector of item i **being optimized**
- $y^{(i,j)} \in \mathbb{R}$ is the scalar rating of user i for item j
- m_{ratings} is the number of known ratings
- N_m is the number of movies, N_u is the number of users
- n is the number of features

Gradient Update for CF Objective:

$$\frac{\partial J(x^{(i)}, \theta^{(j)})}{\partial x_k^{(i)}} = \frac{1}{m} \left(\sum_{j:r(i,j)=1} \left(\left(\theta^{(j)}\right)^T x^{(i)} - y^{(i,j)} \right) \theta_k^{(j)} + \lambda x_k^{(i)} \right)$$

$$\frac{\partial J(x^{(i)}, \theta^{(j)})}{\partial \theta_k^{(j)}} = \frac{1}{m} \left(\sum_{i:r(i,j)=1} \left(\left(\theta^{(j)}\right)^T x^{(i)} - y^{(i,j)} \right) x_k^{(i)} + \lambda \theta_k^{(j)} \right)$$

$$x_k^{(i)} = x_k^{(i)} - \alpha \cdot \left(\frac{\partial J(x^{(i)}, \theta^{(j)})}{\partial x_k^{(i)}} \right), \theta_k^{(j)} = \theta_k^{(j)} - \alpha \cdot \left(\frac{\partial J(x^{(i)}, \theta^{(j)})}{\partial \theta_k^{(j)}} \right)$$

Approach 2: Finding 'Similar' Users

- *Jaccard Similarity*: $\text{sim}(x,y) = |x \cap y| \;/\; |x \cup y|$
  - counting number of **items rated by both users**
  - *Issue*: ignores the value of the rating
- *Cosine Similarity*:

$$\text{sim}(x,y) = \frac{\sum_{S_{xy}} r_{xi} \cdot r_{yi}}{\sqrt{\sum_{S_x} r_{xi}^2} \cdot \sqrt{\sum_{S_y} r_{yi}^2}}$$

  - *Issue*: implicitly zero-fills missing ratings.
- ***Pearson Correlation Coefficient***

  Intuition: the formula is equivalent to normalized cosine similarity. Each term mean-centered by subtracting average. Therefore, missing ratings (implicitly 0) is $r_{xs} - \overline{r_x} = 0$. Not bad!

  Therefore, EXAM TIP: make matrix mean-centered by subtracting each r_{xs} by $\overline{r_x}$ first. Then do it like normal cosine similarity.

$$\text{sim}(x,y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})(r_{ys} - \overline{r_y})}{\sqrt{\sum_{s \in S_x} (r_{xs} - \overline{r_x})^2} \sqrt{\sum_{s \in S_y} (r_{ys} - \overline{r_y})^2}}$$

- S_{xy} = set of items rated by both users x and y
- S_x, S_y = set of items rated by x, y respectively
- $\overline{r_x}, \overline{r_y}$ = avg. rating of x, y respectively
- r_{xs} = rating of x for item s

After computing a similarity matrix with the above
- Let r_x be the vector of user x's ratings

User-User CF: Predicted item i rating by user x from similar users:
- Let K_{si} be set of k users most similar to x who have rated item i

$$\hat{r}_{xi} = \frac{1}{k} \sum_{j \in K_{si}} r_{ji}, \quad \text{OR} \quad r_{xi} = \frac{\sum_{j \in K_{si}} \text{sim}(x,j) \cdot r_{ji}}{\sum_{j \in K_{si}} \text{sim}(x,j)}$$

Item-Item CF: Predicted item i rating by user x from similar items:
- Let K_{si} be set of k items most similar to i that user x has rated

$$\hat{r}_{xi} = \frac{1}{k} \sum_{j \in K_{si}} r_{xj}, \quad \text{OR} \quad r_{xi} = \frac{\sum_{j \in K_{si}} \text{sim}(i,j) \cdot r_{xj}}{\sum_{j \in K_{si}} \text{sim}(i,j)}$$

item-item generally works better, since items are simpler, while user have complex and nuanced tastes

Common Practice:
For weighted average methods, add baseline to correct for user biases

$$r_{xi} = b_{xi} + \frac{\sum_{j \in K_{si}} \text{sim}(i,j) \cdot (r_{xj} - b_{xj})}{\sum_{j \in K_{si}} \text{sim}(i,j)}$$

- $b_{xi} = \mu + b_x + b_i$: baseline rating for user x and item i
- b_x = (avg. rating of x) $- \mu$ rating deviation of user x
- b_i = (avg. rating of i) $- \mu$ rating deviation of item i
- μ = global mean movie rating

PageRank

Intuition: each score is like a **web surfer**
- **Locally**: each surfer randomly (evenly) go to one neighbor.
- **Globally**: after sufficient iterations, popular websites will accumulate more surfer than smaller ones (e.g., Google have more visitors than smaller websites at any given time)

Term Definitions:
- Let page $1 \le i \le N$ has d_i out-links.
- Then define ***Column Stochastic Matrix*** $M \in \mathbb{R}^{N \times N}$:
  - $M_{i,j}$ = probability that a surfer at page i will next go to page j

$$M_{i,j} = \begin{cases} \frac{1}{d_i} & \text{if exist path } i \to j \\ 0 & \text{otherwise} \end{cases}$$

- as well as ***Rank Vector*** $r \in \mathbb{R}^N$:
  - Each r_i is the current importance score of page i.
  - Scores sum to 1: $\sum_i r_i = 1$

Flow Equation: $r^{(t+1)} = M \cdot r^{(t)}$

$$\therefore r^{(t+1)} = Mr^{(t)} = M\left(Mr^{(t-1)}\right) = M^t r^{(0)}$$

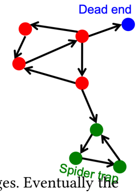
Intuition:
- assume $r^{(t)}$ converge and stop changing, then $Mr^{(t)} = 1 \cdot r^{(t)}$.
- Using the eigenvector definition $Mx = \lambda x$, $r^{(t)}$ is the first (or principal) eigenvector of M, with eigenvalue $\lambda = 1$

Proof
1. Assume M has n linear independent eigenvectors $x_1, x_2, \ldots x_n$ with eigenvalues $\lambda_1, \lambda_2, \ldots \lambda_n$, where $\lambda_1 > \lambda_2 > \ldots > \lambda_n$
2. Vectors $x_1, x_2, \ldots x_n$ form a basis, thus we can write
   - $r^{(0)} = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n$
3. Then:

$$\begin{aligned} Mr^{(0)} &= M(c_1 x_1 + c_2 x_2 + \ldots + c_n x_n) \\ &= c_1(Mx_2) + c_2(Mx_2) + \ldots + c_n(Mx_n) \\ &= c_1(\lambda_1 x_1) + c_2(\lambda_2 x_2) + \ldots + c_n(\lambda_n x_n) \\ M^k r^{(0)} &= c_1(\lambda_1^k x_1) + c_2(\lambda_2^k x_2) + \ldots + c_n(\lambda_n^k x_n) \\ &= \lambda_1^k \left[c_1 x_1 + c_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k x_2 + \ldots + c_n \left(\frac{\lambda_n}{\lambda_1}\right)^k x_n \right] \end{aligned}$$

1. Since $\lambda_1 > \lambda_2$, then $\frac{\lambda_2}{\lambda_1}, \frac{\lambda_3}{\lambda_1}, \ldots < 1$
2. Therefore, $\left(\frac{\lambda_i}{\lambda_1}\right)^k = 0$ as $k \to \infty$ for all $i \ge 2$
3. Thus $M^k r^{(0)} \to c_1(\lambda_1^k x_1)$, **the largest eigenvalue always 1**

Power Iteration Method



1. Suppose there are N web pages
2. Uniformly initialize $r^{(0)} = \left[\frac{1}{N}, \frac{1}{N}, \ldots, \frac{1}{N}\right]^T$
3. Iterate $r^{(t+1)} = M \cdot r^{(t)}$
4. Stop when $|r^{(t+1)} - r^{(t)}|_1 < \varepsilon$
   - $|x|_1$ is L1-norm. But other norms work too.

Problems:
- *Dead Ends*: scores leak out of network
- *Spider Traps*: scores stuck indefinitely in small set of pages. Eventually the *Spider Trap* will absorb all importance, draining other pages.

Solutions: Random Teleports

- ***What?***
  - With probability β, follow a link at random
  - With probability $(1-\beta)$, jump to some random page
  - In practice, $\beta = 0.8 \sim 0.9$ (make 5 steps on avg. before jump)

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1-\beta) \frac{1}{N}$$

  - Alternatively, reformulate as **Power Iteration Method**:

$$A = \beta M + (1-\beta) \left[\frac{1}{N}\right]_{N \times N}$$

  - Using the new matrix A, we get the familiar form: $r^{(t+1)} = Ar^{(t)}$

- ***Why?***
  - *Spider Traps*: Teleport out of spider traps in finite steps
  - *Dead Ends*: teleport to a random page when nowhere to go

- ***How?*** MapReduce program for PageRank:

```
Map(key, value) {
  # key: a current page id
  # value: page rank of the
current page
  for adj_page in Adj[key]:
    emit(adj_page, value /
count(Adj[key]))
}
```

```
Reduce(key, values) {
  # key: a current page id
  # values: a list of incoming
scores from other pages
  scores[key] = 1 - beta
  for score in values:
    scores[key] += beta * score
  emit(key, scores[key]);
}
```

Clustering

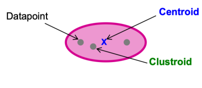
Distance Metrics (for Vectors)
- Euclidean Distance: $d(\mathbf{A}, \mathbf{B}) = \sqrt{\sum_{i=1}^n (A_i - B_i)^2}$
- Cosine Similarity:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

- Cosine Distance = $1 - \cos(\theta)$

Distance Metrics (for Sets)
- Jaccard Similarity: $J(S_A, S_B) = |S_A \cap S_B| \;/\; |S_A \cup S_B|$
- Jaccard Distance = $1 - J(S_A, S_B)$

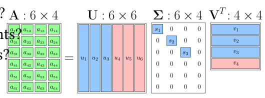
K-Means Clustering
1. Initialize: randomly create k cluster centroid points
2. Assign: assign each point to the nearest cluster centroid
3. Update each cluster centroid to the mean of all itis points
4. Repeat 2-3 until convergence



- **pro**: simple, user provide k
- **cons**: sphere, hard to guess k
- **complexity**: $O(n \times k \times I \times d)$
  - n: num of points, k num of clusters, I num iterations, d num of attributes

Concepts
- *Centroid*: is the avg. of all points in the cluster (artificial point)
- *Clustroid* is **existing point** closest to all other points

- smallest avg distance to other points?
- smallest sum of squares to other points?
- smallest max distance to other points?



Distance Metrics
- Continuous Variable: $L_1, L_2, \ldots, L_p$ norm
- Binary Variable: $|0-1| + |1-0| + |1-1| \ldots$
- Categorical: 1. One-hot encoding, 2. Simple matching: (total features $-$ matched features)/ total features

Dimensionality Reduction (DR)

Singular Vector Decomposition (SVD)

Easy Method (w/ Calculator):
1. Given matrix $A \in \mathbb{R}^{m \times n}$ to factorize
2. $U = AA^T \in \mathbb{R}^{m \times m}$. Find its eigenvectors $\{\vec{u}_1, \vec{u}_2, \ldots, \vec{u}_m\}$
3. $V = A^T A \in \mathbb{R}^{n \times n}$. Find its eigenvectors $\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n\}$
4. Determine U and V's **common eigenvalues** $\{\lambda_1, \lambda_2, \ldots, \lambda_r\}$

[SVD 分解] 性质：$A^{[m\times n]} = U^{[m\times m]}\Sigma^{[m\times n]}(V^{\mathrm{T}})^{[n\times n]}$，$U^{\mathrm{T}}U = V^{\mathrm{T}}V = I$

1. 计算$(AA^{\mathrm{T}})^{[m\times m]}$，找到它的特征向量$\vec{u}$、它的特征值$\lambda$并开方 $\boldsymbol{\sigma \coloneqq \sqrt{\lambda}}$

2. 给σ降序排序得到序列σ^*，然后得到Σ： **$\Sigma^{[m\times n]} = \mathrm{diag}(\sigma^*)$，其他地方补0**

3. 对所有$\vec{u}$按照对应的σ^*排序，得到U，即： $\boldsymbol{U^{[m\times m]}} = \left[\frac{\vec{u}_1^*}{|\vec{u}_1^*|} \quad \frac{\vec{u}_2^*}{|\vec{u}_2^*|} \quad \cdots \quad \frac{\vec{u}_r^*}{|\vec{u}_r^*|}\right]$

4. 根据$A = U\Sigma V^{\mathrm{T}}$，得到$\boldsymbol{V^{\mathrm{T}}} = \boldsymbol{\Sigma^+ U^{-1} A}$。$\Sigma^+$是左广义逆矩阵（$\Sigma^+\Sigma = I$）。例如，

$$\Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \Longrightarrow \Sigma^+ = \Sigma^{-1} = \frac{1}{\Sigma} = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{3} \end{bmatrix}, \qquad \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 3 \\ 0 & 0 \end{bmatrix} \Longrightarrow \Sigma^+ = \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \end{bmatrix} \text{ (pseudoinverse)}$$

去除$\sigma_k^* = 0$（甚至忽略较小的σ_k^*）对应的U、V的列，实现数据压缩，例如，

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} \cdot 2 \cdot \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}^{\mathrm{T}}$$

Best Low Rank Approx.
- Let $B = USV^T$, where $S \in \mathbb{R}^{k \times k}, k \le r$
- Objective: $\min_B \|A - B\|_F = \min_B \sqrt{\sum_{ij} (A_{ij} - B_{ij})^2}$
- Intuitively, select **top k best ranks** to approximate A

Eigenvector & Eigenvalue

$\boldsymbol{Mx = \lambda x}$, where $M \in \mathbb{R}^{n \times n}$ (square matrix), $x \in \mathbb{R}^n$, $\lambda \in \mathbb{R}$

$$Mx = \lambda x \Rightarrow Mx - \lambda x = 0$$

To factor our λ introduce **identity matrix** $I \in \mathbb{R}^{n \times n}$:

$$\begin{cases} Mx - \lambda I x = 0 \\ (M - \lambda I) x = 0 \end{cases} \Rightarrow \quad \text{solve: } \det(M - \lambda I) = 0$$

For 2×2 matrix: $\det\begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc$

Therefore, solve for characteristic equation: $(a - \lambda)(d - \lambda) - bc = 0$

Principal Component Analysis (PCA)

1. Compute covariance matrix Σ
2. Calculate eigenvalues of eigenvectors of Σ
   - Eigenvectors w/ largest eigenvalue λ_1 is 1st principal component
   - Eigenvectors with k^{th} largest eigenvalue λ_k is k^{th} PC
   - Proportion of variance captured by k^{th} PC = $\lambda_k / \left(\sum_i \lambda_i\right)$

NOTES:
- PCA can be seen as noise reduction.
- Fail when data consists of multiple clusters
- Direction of max variance may not be most informative

Non-linear DR
- ISOMAP: Isometric Feature Mapping
- t-SNE: t-Sochastic Neighbor Embedding
- Autoencoders: map $\mathbb{R}^n \to \mathbb{R}^k \to \mathbb{R}^n$, minimize reconstruction loss

Appendix

Derivatives

$$\frac{\partial x^n}{\partial x} = nx^{n-1}$$

$$\frac{\partial k * f(x)}{\partial x} = k \frac{\partial f(x)}{\partial x}$$

$$\frac{\partial f(x) + g(x)}{\partial x} = \frac{\partial f(x)}{\partial x} + \frac{\partial g(x)}{\partial x}$$

$$\frac{\partial f(x) - g(x)}{\partial x} = \frac{\partial f(x)}{\partial x} - \frac{\partial g(x)}{\partial x}$$

$$\frac{\partial f(x) g(x)}{\partial x} = f(x) \frac{\partial g(x)}{\partial x} + g(x) \frac{\partial f(x)}{\partial x}$$

$$\frac{\partial \frac{f(x)}{g(x)}}{\partial x} = \frac{g(x) \frac{\partial f(x)}{\partial x} - f(x) \frac{\partial g(x)}{\partial x}}{(g(x))^2}$$

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(u)}{\partial u} \cdot \frac{\partial g(x)}{\partial x}$$

where u = g(x)

Elementary Functions

Hyperbolic Functions

$$\frac{\partial \sinh(x)}{\partial x} = \cosh(x) \frac{\partial \cosh(x)}{\partial x} = \sinh(x$$

$$\frac{\partial \tanh(x)}{\partial x} = \text{sech}^2(x) \frac{\partial \coth(x)}{\partial x} = -\text{csch}^2$$

$$\frac{\partial\, \text{sech}(x)}{\partial x} = -\text{sech}(x) * \tanh(x)$$

$$\frac{\partial\, \text{csch}(x)}{\partial x} = -\text{csch}(x) * \coth(x)$$

Exponential Functions

$$\frac{\partial e^x}{\partial x} = e^x \qquad \frac{\partial a^x}{\partial x} = a^x \cdot \ln(a)$$

Logarithmic Functions

$$\frac{\partial \ln(x)}{\partial x} = \frac{1}{x} \qquad \frac{\partial \log_{a(x)}}{\partial x} = \frac{1}{x \cdot \ln(}$$

Derivatives of *Loss Function* $\left(\frac{\partial L}{\partial \hat{y}}\right)$
- No need to consider their batched version. For example, only need to consider $(y - \hat{y})^2$ for MSE. No need for $\frac{1}{n} \sum_{i=1}^n (y - \hat{y})^2$
- Because n samples $\to n$ different $\hat{y} \to n$ different gradients for each weight down the chain rule.
- For any L in the form of $\frac{1}{n} \sum f(\hat{y}_i, y_i)$, the derivative yield by each sample is $\frac{1}{n} f'(\hat{y}_i, y_i)$. Try
- So as the gradients accumulate for each weight w, they are automatically batch averaged $(1/n)$.

Softmax Derivative

$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^C} e^{z_j}$ subsititute, $S = \sum_{j=1}^C e^{z_j}$ then, $\sigma(z)_i = \frac{e^{z_i}}{S}$ $\because \frac{\partial}{\partial x}\left(\frac{u}{v}\right) = \frac{vu' - uv'}{v^2}$ Find u' and v' for $\sigma(z)_j$ where $u = e^{z_i}, v = S$:

$u' = \frac{\partial e^{z_i}}{\partial z_j} = \begin{cases} e^{z_i} & \text{if } i=j \\ 0 & \text{if } i \ne j \end{cases}, \quad v' = \frac{\partial S}{\partial z_j} = \frac{\partial}{\partial z_j}\left(\sum_{j=1}^C e^{z_j}\right) = e^{z_j}$

Differentiate (*note*: $\Delta_{ij} = \begin{cases} 1 \text{ if } i=j \\ 0 \text{ if } i \ne j \end{cases}$):

$\therefore \frac{\partial \sigma_i}{\partial z_j} = \frac{S \cdot [e^{z_i} \cdot \Delta_{ij}] - e^{z_i} \cdot e^{z_j}}{S^2} = e_{z_i} \cdot \frac{S \cdot \Delta_{ij} - e^{z_j}}{S^2} = \frac{e^{z_i}}{S} \cdot \frac{S \cdot \Delta_{ij} - e^{z_j}}{S} = \sigma(z)_i \cdot \frac{S \cdot \Delta_{ij} - e^{z_j}}{S} = \sigma(z)_i \cdot \left(\Delta_{ij} - \frac{e^{z_j}}{S}\right) = \sigma(z)_i \cdot \left(\Delta_{ij} - \sigma(z)_j\right)$

Tanh **Derivative**

$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ Let $u = e^x - e^{-x}$ and $v = e^x + e^{-x}$, find u' and v': $u' = e^x + e^{-x}$ $\quad v' = e^x - e^{-x}$

Differentiate:

$\frac{\partial}{\partial x} \tanh(x) = \frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2} = \frac{(e^x + e^{-x})^2 - (e^x - e^{-x})^2}{(e^x + e^{-x})^2}$

$= 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} = 1 - \tanh^2(x)$

OR: $\quad \frac{\partial}{\partial x} \tanh(x) = \frac{\partial}{\partial x}\left(\sinh \frac{x}{\cosh} x\right) = \frac{\cosh x \cosh x - \sinh x \sinh x}{\cosh^2 x} = \frac{\cosh^2 x - \sinh^2 x}{\cosh^2 x} = \frac{1}{\cosh^2 x} = \text{sech}^2 x = 1 - \tanh^2(x)$

Sigmoid Derivative

$\sigma(x) = \frac{1}{1 + e^{-x}}$, $\because \frac{\partial}{\partial u}\left(\frac{1}{u}\right) = \left(-\frac{1}{u^2}\right)$ Let $u = 1 + e^{-x}$:

$$\frac{\partial \sigma}{\partial x} = \frac{\partial \sigma}{\partial u} \cdot \frac{\partial u}{\partial x} = \frac{\partial}{\partial u}\left(\frac{1}{u}\right) \cdot \frac{\partial}{\partial x}(1 + e^{-x})$$

$$= -\frac{1}{u^2} \cdot -e^{-x} = \frac{e^{-x}}{u^2} = \frac{e^{-x}}{(1 + e^{-x})^2} = \frac{1}{1 + e^{-x}} \cdot \frac{e^{-x}}{1 + e^{-x}}$$

$$= \frac{1}{1 + e^{-x}} \cdot \frac{(1 + e^{-x}) - 1}{1 + e^{-x}} = \frac{1}{1 + e^{-x}} \cdot \left(\frac{1 + e^{-x}}{1 + e^{-x}} - \frac{1}{1 + e^{-x}}\right)$$

$$= \frac{1}{1 + e^{-x}} \cdot \left(1 - \frac{1}{1 + e^{-x}}\right) = \sigma(x) \cdot (1 - \sigma(x))$$