Recommender Systems

Sargur N. Srihari
srihari@cedar.buffalo.edu

This is part of lecture slides on Deep Learning:
http://www.cedar.buffalo.edu/~srihari/CSE676

Topics in Recommender Systems

- Types of Recommender Systems
- Collaborative Filtering
- Word Embeddings to Item Embeddings
- Bilinear Prediction
- Relationship to Reinforcement Learning
  - Contextual Bandits
  - Exploration vs. Exploitation

Acknowledgements: Goodfellow, Bengio, Courville, Deep Learning, MIT Press, 2016

Types of Recommender Apps

- A major family of applications of ML in the IT sector is the ability to make recommendations of items to potential users or customers
- Two major types of applications:
  - Online advertising
  - Item recommendations (for selling a product)
- Both rely on predicting association between user and an item

Predicting user-tem association

- Useful for either to predicting probability of some action
  - User buying product or some proxy for this action
- Or the expected gain (which may depend on the value of the product) if an ad is shown or a recommendation is made regarding that product to the user

Commercial importance

- The internet is currently financed by various forms of online advertising
- Major parts of the economy rely on online shopping
- Amazon, eBay use ML including deep learning for product recommendations
- Sometimes items are not products for sale
  - E.g., selecting posts to display on social network feeds, recommending movies to watch, recommending jokes, recommending advise

Collaborative Filtering

- Early work on recommender systems
  - Relied on minimal inputs for prediction
  - Rely on similarity between patterns of values of target variable for different users or different items
    - $\text{user } 1$ and $\text{user } 2$ both like items A, B and C
    - We may infer that user 1 and user 2 have similar tastes
    - If $\text{user } 1$ likes item D then this a strong cue that $\text{user } 2$ will also like D
    - Algorithms based on this principle come under the name of *collaborative filtering*

Collaborative Filtering Methods

- Both non-parametric and parametric
- Non-parametric methods
  - Nearest neighbor based on similarity between patterns of preferences
- Parametric methods
  - Rely on learning a distributed representation called an *embedding* for each user and each item
  - Bilinear prediction of the target variable described next

Word Embeddings to Item Embeddings

Sentence-word data

s0: "Man walked his dog"

s1: "Man took his dog to park"

s2: "Dog went to park"

Vocabulary:
{man, walked, his, dog, took, to, park, went}

Vector representation:
s0: [1,1,1,1,0,0,0,0]
s1: [1,0,1,1,1,1,1,0]
s1: [0,0,0,1,0,1,1,1]

Word2Vec produces word embeddings in a low-dimensional continuous space and carry semantic and syntactic information of words

User-item data

user0: Loc0, Loc1, Loc2, Loc3

user1: Loc0, Loc4, Loc2, Loc3, Loc5, Loc6

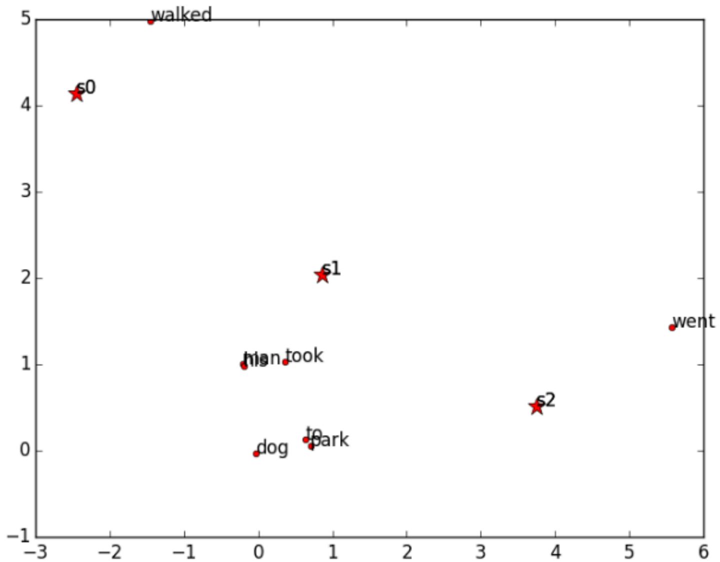
user2: Loc3, Loc7, Loc5, Loc6

Vocabulary:
{Loc0, Loc1, Loc2, Loc3, Loc4, Loc5, Loc6, Loc7}

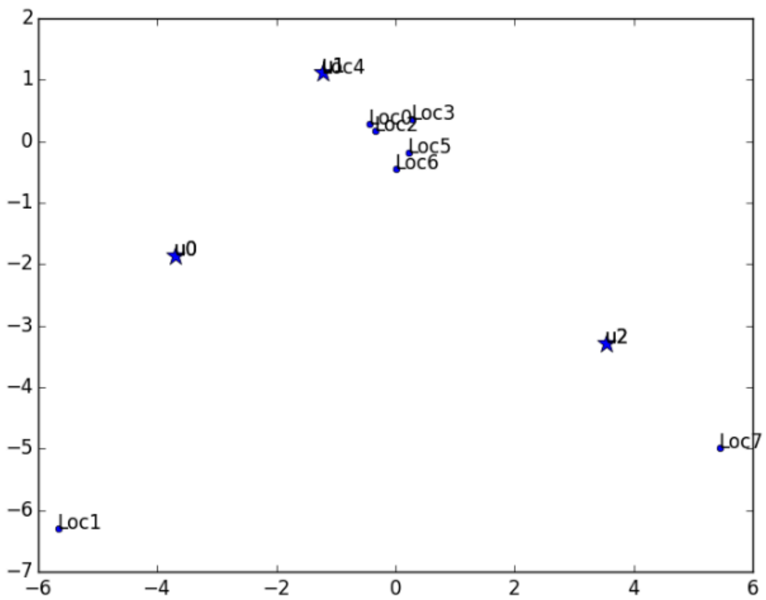
Vector representation:
user0: [1,1,1,1,0,0,0,0]
user1: [1,0,1,1,1,1,1,0]
user2: [0,0,0,1,0,1,1,1]

Continuous vector representations

Word2Vec



Item2Vec



Similarly we get user2Vec

Prediction of target variable

- Prediction of target variable (such as a rating)
  - Bilinear prediction is a simple parametric method
    - Highly successful
    - Found as a component in state-of-the-art systems
- Prediction is obtained by a dot product between the user embedding and the item embedding
  - Possibly corrected by by constants that depend only on either the user ID or item ID

Bilinear Prediction

- Definitions
  - Let R be the matrix containing our predictions
  - A a matrix with user embeddings in its rows
  - B a matrix with item embeddings in its columns
  - Let b and c be vectors that contain respectively
    - a kind of bias for each user
      - Representing how grumpy or positive that user is
    - For each item
      - Representing its general popularity
- The bilinear prediction is $\boxed{\hat{R}_{u,i} = b_u + c_i + \sum_j A_{u,j} B_{j,i}}$
- Typical goal: minimize squared error between predicted ratings $\boxed{\hat{R}_{u,i}}$ and actual ratings $R_{u,i}$

Use of embeddings

- User embeddings and item embeddings can then be conveniently visualized when they are first reduced to a low dimension (two or three)
- Or they can be used to compare users or items against each other, just like word embeddings

Obtaining embeddings

- One way to obtain these embeddings is by performing singular value decomposition (SVD) of the matrix R of actual targets (such as ratings)
- This corresponds to factorizing $\mathrm{R=UDV'}$ (or a normalized variant) into the product of two factors, the lower rank matrices $\mathrm{A=UD}$ and $\mathrm{B=V'}$

Missing Entry problem with SVD

- One problem with SVD is that it treats missing entries in an arbitrary way, as if they corresponded to a target value of 0
- Instead we would like to avoid paying any cost for the predictions made on missing entries
- Fortunately the sum-of-squared-errors on the observed ratings can also be easily minimized by gradient-based optimization

Netflix Competition

$$\hat{R}_{u,i} = b_u + c_i + \sum_j A_{u,j} B_{j,i}$$

- Both SVD and bilinear prediction performed very well
- Competition was to predict ratings for films, based on previous ratings by a large set of anonymous users
- Even though it did not win by itself, the simple bilinear prediction of SVD was a component of the ensemble models presented by most competitors including the winners

Limitation of Collaborative Filtering

- When a new item or a new user is introduced, its lack of rating history means that there is no way to evaluate its similarity with other items or users (respectively),
- Or the degree of association between that user and existing items
- This is the problem of *cold-start recommendations*

Solution to cold-start recommendation

- Introduce extra information about individual users or items
- Extra information could be user profile information or features of each item
- Systems that use such information are called *content-based recommender systems*
- The mapping from a rich set of user features or item features to an embedding can be learned through a deep learning architecture

Content-based recommender systems

- Deep learning architectures such as CNNs learn to extract from rich content, e.g., musical audio tracks, for music recommendation
  - CNN takes acoustic features as input and computes an embedding for the associated song
  - Dot product between this song embedding and the embedding for a user is then used to predict whether a user will listen to the song

Relationship to Reinforcement Learning

- A recommendation issue goes beyond supervised learning and into reinforcement learning
- Most recommendation problems are accurately described theoretically as *contextual bandits*
  - When recommendation system collects data, we get a biased and incomplete view of user preferences
    - We only see responses of users to items they were recommended and not to other items
    - In some cases no information on users for whom no recommendation has been made
      - E.g., with ad auctions, price proposed was below minimum, or does not win auction, so that ad is not shown

Need for additional information

- System gets no information on what outcome would result from recommending any other item
  - It would be like training a classifier by picking one class $\hat{y}$ for each training example $\boldsymbol{x}$ (typically class with highest probability) and then getting feedback whether this was correct or not
    - Each example conveys less information than in the supervised case where the true label y is directly observable, so more examples are necessary
      - We may keep on picking the wrong model output to show
    - The correct decision may have a low probability
      - Until learner learner picks the correct decision it does not learn about the correct decision

Reinforcement Learning and Bandits

- In reinforcement learning only the reward for the selected action is observed
- In general, reinforcement learning can involve a sequence of many actions and many rewards
- Bandits scenario is a special case of reinforcement learning, in which the learner takes only a single action and receives a single reward
  - Bandit problem is easier in that the learner knows which reward is associated with which action

Multi-armed Bandit Problem

One-armed
bandit

1. Which machines to play
2. How many times to play each machine
3. In which order to play

Multi-armed
bandit

- Each machine has
  - a probability distribution, $B=\{R_1,..,R_K\}$
    - Mean values $\mu_1,..\mu_K$ associated with rewards
- Objective:
  - Maximize reward through sequence of lever pulls
    - Minimize regret ρ , expected difference between optimal strategy and collected rewards r_t, after T rounds

$$\rho = T\mu^* - \sum_{t=1}^{T} \hat{r}_t$$

Contextual Bandits

- In general reinforcement learning, a high or low reward might have been caused by a recent action or by an action in the distant past
- Contextual bandits refers to where the action is taken in context of an input variable that can inform the decision
  - E.g., we at least want to know user identity, and we want to pick an item
- Mapping from context to action is called policy
  - Feedback loop between learner and data distribution (which depends on actions of learner)
    - a central research issue in reinforcement learning

Exploration vs Exploitation

- Reinforcement learning requires a tradeoff between exploration and exploitation
- Exploitation comes from taking actions that come from the current best version of the learned policy
  - Actions that we know will achieve a high reward
- Exploitation refers to taking actions specifically in order to obtain more training data

Exploration example

- We know: in context $\boldsymbol{x}$, action a has $\text{reward}=1$
  - We don't know whether it's the best possible reward
  - We may want to exploit our current policy and continue taking action a in order to be relatively sure of obtaining a reward of 1
- However we may also want to explore action a'
  - We do not know what will happen if we try action a'
  - We hope for $\text{reward}=2$ but we may get $\text{reward}=0$
  - Either way we gain some knowledge

Implementing Exploration-Exploitation

- Exploration can be implemented in many ways
  1. Random actions to cover all possible actions
  2. Model-based approaches
     - compute a choice of action based on its expected reward and the model's amount of uncertainty about that reward
- Factors determining exploration or exploitation
  - One prominent factor: Time scale
    - Agent has short time to accrue reward: exploitation
    - Agent has more time: begin with exploration so that future actions can be planned more effectively
      - As time progresses we move towards more exploitation

Supervised Learning has no tradeoff

- No tradeoff between exploration-exploitation
- Supervision signal always specifies which output is correct for each input
- There is no need to try out different outputs to determine if one is better than the model's current output
  - We always know that the label is the best output

Evaluating policies

- Another difficulty arising in the context of reinforcement learning
  - Besides exploration-exploitation tradeoff
- Difficulty of evaluating and comparing different policies
  - Reinforcement learning involves interaction between learner and environment
    - It is not straightforward to evaluate the learner's performance using a fixed set of test input values
    - The policy itself determines which inputs will be seen