Visual Recommender Systems

Advisor

Julian Mcauley

Team Members

Alex Egg, Peyman Hesami, Deepthi Mysore Nagaraj, Julius Remigio

Abstract

State of the art recommender systems exploits implicit and explicit user feedbacks to improve the effectiveness and accuracy of their recommendations. This is done by discovering the underlying dimensions that contain the hidden information regarding the items and the users. One of the most popular recommender systems models users' individual preferences towards products by using a bayesian personalized ranking (BPR) from implicit feedback. In this paper we extend this model by incorporating novel features. We explore a novel mix of visual, and nonvisual features gained through product details. We propose a matrix factorization model incorporating such visual and nonvisual features into predictors of user's opinion towards items, which we apply to a selection of large real-world Amazon dataset. Nonvisual features are extracted from the product description and details while visual features are extracted from pre-trained convolutional neural networks of which we extract the most important visual dimensions that best learn user feedback [9]. These features are then used separately and also in combination with each other to improve personalized rankings. This leads to significantly more accurate personalized ranking and also it helps to alleviate the so called cold start issue. It also helps to analyze the visual and nonvisual dimensions of products that influences people's opinion towards them.

Introduction

Recommender systems are widely used in many companies. E-commerce companies (Netflix, Amazon), social network companies (Facebook, Twitter, LinkedIn), music streaming services (Spotify, Pandora) are all exploiting the benefits of recommender systems to boost their customer satisfaction and as a results their profits. The goal of recommender systems comprises of (but not limited to):

- Helping people discover new content (Netflix/Amazon)
- Helping people find the content that they were already looking for (Netflix)
- Personalizing user experiences in response to user feedback (Pandora)

The goal of this project is to implement a state of the art recommender system and improve its performance. There are two general approach to tackle this problem [1]:

1. **Content Filtering Methods:** In this method, a profile is constructed for each user or product to characterize its nature. For example for movies as products, attributes like its genre, year of release, actors,... can be used to create a movie profile. A user profile might include the demographic information (age, income, location,...) or user preferences towards different movie genres. The product (or user) profiles then will be used as features to create predictive models on the user preference towards different products.

2. **Collaborative Filtering Methods:**
   a. **Neighborhood methods:** These methods performs recommendation in terms of user/user and item/item similarity
   b. **Latent-Factor models:** These models perform recommendation by projecting users and items into some low-dimensional space

In this project we focus on latent-factor models as they have proved to have the best performance and they are utilized in many state of the art recommender systems.

The latent factor models rely solely on the past user's behavior. The user behavior is usually represented as a matrix (R) where each row represents a user and columns represents products:

$$R = \underbrace{\begin{pmatrix} 5 & 3 & \cdots & \cdot \\ 4 & 2 & & 1 \\ 3 & \cdot & & 3 \\ \cdot & 2 & & 4 \\ 1 & 5 & & \cdot \\ \vdots & & \ddots & \vdots \\ 1 & 2 & \cdots & \cdot \end{pmatrix}}_{\text{items}} \left.\vphantom{\begin{pmatrix} 5 \\ 4 \\ 3 \\ \cdot \\ 1 \\ \vdots \\ 1 \end{pmatrix}}\right\} \text{users}$$

These methods model ranking of user's preference (rating) to products by projecting the the user and product onto a low dimensional space and in this way derive the user's taste towards products (Fig.1)

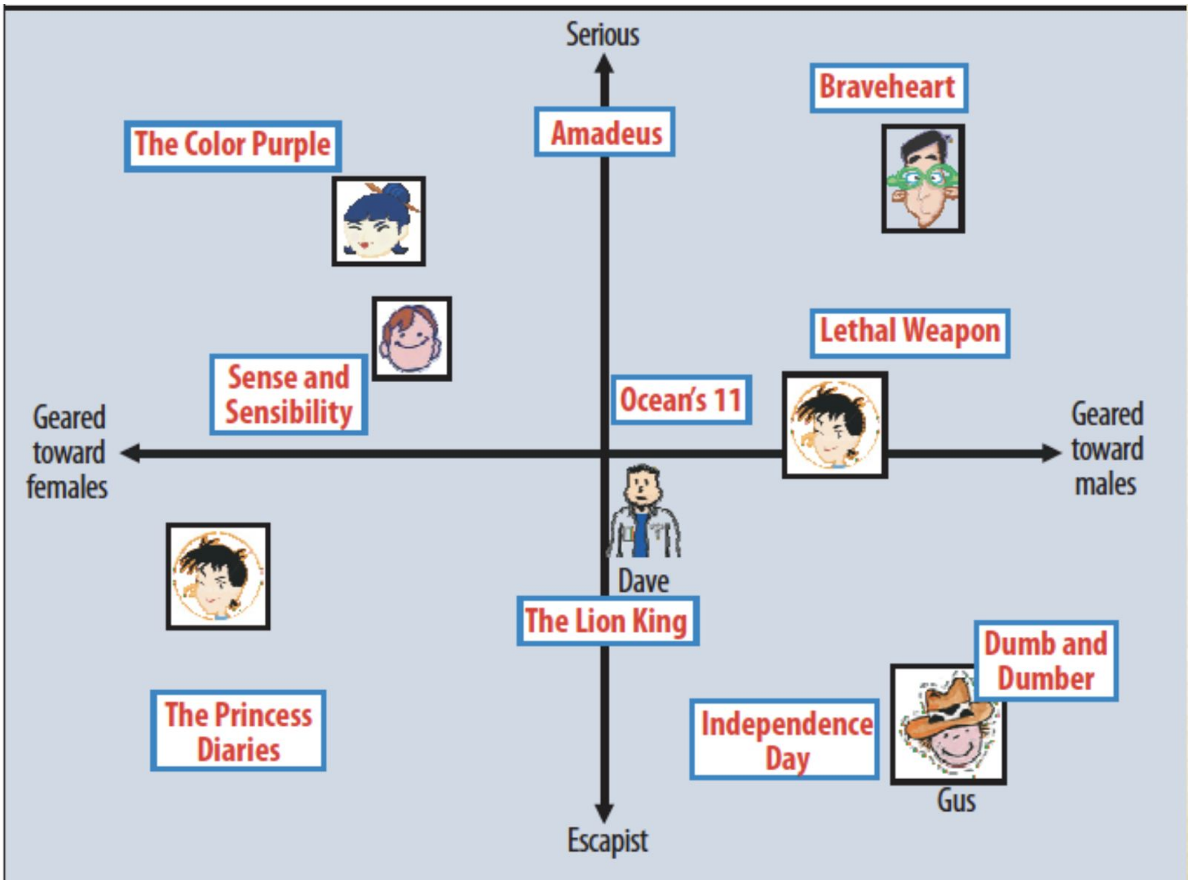


Fig 1. The projection of products and users onto two dimensional space

The historical data (matrix R) used to train these recommender systems comes in the form of either explicit feedback (such as star ratings) or implicit feedback (such as purchase history). The real-world datasets for such historical data are often very large and methods such as Matrix Factorizations (MF) have been introduced to uncover the latent dimensions of these data [1]. Although these methods have had great success in improving the performance of recommender systems, they suffer from *cold start* issue due to the sparsity of the real-work datasets.

A variety of sources of data have been used to build hybrid models to alleviate the cold start problems. From using the cast of movies to build context-aware recommender systems [10], using the review text [11], to a user's physical location [12], to the season or temperature [13].

Visual Personalized Ranking:

Incorporating the visual appearance of the items into the preference predictor has been explored and shown to give superior performance compared to the recommender systems that doesn't utilize these dimensions [9]. The intuition behind this is that one wouldn't buy an item (a t-shirt from Amazon for example) without seeing the item in question, and therefore it is possible to uncover the visual dimensions that are relevant to people's opinions which in turn can lead to improved performance at tasks like personalized ranking. The visual aspects of items are modeled by using representations of product images derived from a (pre-trained) deep network [11].

Non-Visual Personalized Ranking:

As mentioned before several source of non-visual data have been used to build context-aware recommender systems [10-13], however, most of these models are incorporating features that does not relate to the product directly (text review, season temperature, user's physical location). In this project we develop models that incorporate non-visual features of items (such as price, brand, product description) for the task of personalized ranking on implicit feedback datasets. By learning the non-visual dimensions people consider when selecting products we will be able to alleviate cold start issues, help explain recommendations in terms of item characteristics, and produce personalized rankings that are more consistent with users' preferences.

Hybrid Personalized Ranking:

The mixture of visual and non-visual features can also be used to produce personalized ranking by learning both visual and non-visual dimensions that people consider when selecting products. In this project we develop models that incorporate these two features of products for the task of personalized ranking on implicit feedback dataset which helps in alleviating the cold start issue and also improve the quality of the personalized rankings.

Related Work:

Matrix Factorization (MF) methods relate users and items by uncovering latent dimensions such that users have similar representations to items they rate highly, and are the basis of many state-of-the-art recommendation approaches [1]. When it comes to personalized ranking from implicit feedback, traditional MF approaches are challenged by the ambiguity of interpreting non-observed' feedback. In recent years, pointwise [12] and pairwise [4] methods have been successful at adapting MF to address such challenges.

In [4], Rendle et al. propose a generalized Bayesian Personalized Ranking (BPR) framework and experimentally show that BPR-MF (i.e., with MF as the underlying predictor) outperforms a variety of competitive baselines. More recently BPR-MF has been extended to accommodate both users' feedback and their social relations [13] as well as the visual factor of the items [9]. Our goal here is complementary as we aim to incorporate non-visual signals (and their combinations with visual signals) into BPR-MF.

As mentioned before, others have also developed content-based and hybrid models that make use of a variety of information sources, including text (and context), taxonomies, and user demographics [6, 14, 15]. However, to our knowledge none of these works has incorporated the features directly related to the product characteristics (such as price, brand, product description) or a mixture of both visual and non-visual features of the product as we do in this project.

Team (Roles and Responsibilities)

Alex Egg: Alex is currently a Data Science Manager at a Hedge Fund in San Diego. He holds an MS in Data Science from UCSD, a BS in Computer Engineering and Computer Science and a minor in Korean from San Diego State University. He posts his machine learning research at: http://eggie5.com. Alex was involved in modeling the recommender system and deploying it into production.

Peyman Hesami: Peyman holds a bachelor and masters in electrical engineering and currently a data scientist at Qualcomm working on wireless network analytics. His backgrounds are in machine learning, statistics, optimizations, and big data analytics. Peyman was involved in modeling the recommender system, tuning the model and incorporating new features to improve the accuracy of the model.

Deepthi Mysore Nagaraj: Deepthi holds a bachelor's degree in telecommunication engineering and currently works as a Data Scientist at Cymer, An ASML company working on error prediction in Laser systems using the data collected by the production systems. Deepthi was involved in data exploration, recommendation engine model enhancement, feature engineering and model evaluation.

Julius Remigio: Julius is currently a Data Architect at Scripps Health. He holds a MS in Data Science from UCSD and a BS in Business Administration from SDSU. His background is in Data Warehousing and Business Intelligence. He is also a Certified Business Intelligence Professional. Julius' main responsibilities included: treasurer, data acquisition, data and pipeline engineering, web scraping, and documentation.

Data Acquisition

Data Sources

The data used for this project was originally collected by Julian, et. al. as part of his original works referenced in the appendix. Any additions and modifications as required for this project are documented below.

Description

The dataset contains product reviews and the products metadata scraped from www.amazon.com. This dataset includes 142.8 million reviews spanning from May 1996 to July 2014 in a json format. This dataset includes reviews (ratings, text, helpfulness votes), product metadata (descriptions, category information, price, brand, and image features), and links (also viewed/also bought graphs).

Files

Files are broken down into several types: small, complete, per category. See Appendix for table of actual file names.

"Small" subsets

K-cores (i.e., dense subsets): These data have been reduced to extract the k-core, such that each of the remaining users and items have k reviews each.
Ratings only: These datasets include no metadata or reviews, but only (user, item, rating, timestamp) tuples.

Product Category	5-Core Reviews count	Ratings count
Books	8,898,041	22,507,155
Electronics	1,689,188	7,824,482
Movies and TV	1,697,533	4,607,047
CDs and Vinyl	1,097,592	3,749,004
Clothing, Shoes and Jewelry	278,677	5,748,920
Home and Kitchen	551,682	4,253,926
Kindle Store	982,619	3,205,467
Sports and Outdoors	296,337	3,268,695
Cell Phones and Accessories	194,439	3,447,249
Health and Personal Care	346,355	2,982,326
Toys and Games	167,597	2,252,771
Video Games	231,780	1,324,753
Tools and Home Improvement	134,476	1,926,047
Beauty	198,502	2,023,070
Apps for Android	752,937	2,638,172
Office Products	53,258	1,243,186
Pet Supplies	157,836	1,235,316
Automotive	20,473	1,373,768
Grocery and Gourmet Food	151,254	1,297,156
Patio, Lawn and Garden	13,272	993,490
Baby	160,792	915,446
Digital Music	64,706	836,006
Musical Instruments	10,261	500,176
Amazon Instant Video	37,126	583,933

Complete Review Data

- raw review data (20gb) - all 142.8 million reviews

The above file contains some duplicate reviews, mainly due to near-identical products whose reviews Amazon merges, e.g. VHS and DVD versions of the same movie. These duplicates have been removed in the files below:

- user review data (18gb) - duplicate items removed (83.68 million reviews), sorted by user
- product review data (18gb) - duplicate items removed, sorted by product
- ratings only (3.2gb) - same as above, in csv form without reviews or metadata
- 5-core (9.9gb) - subset of the data in which all users and items have at least 5 reviews (41.13 million reviews)

Finally, the following file removes duplicates more aggressively, removing duplicates even if they are written by different users. This accounts for users with multiple accounts or plagiarized reviews. Such duplicates account for less than 1 percent of reviews, though this dataset is probably preferable for sentiment analysis type tasks:

- aggressively deduplicated data (18gb) - no duplicates whatsoever (82.83 million reviews)

Format is one-review-per-line in (loose) json. See examples below for further help reading the data.

Sample review:


```
{
 "reviewerID": "A2SUAM1J3GNN3B",
 "asin": "0000013714",
 "reviewerName": "J. McDonald",
 "helpful": [
        2,
        3
 ],
 "reviewText": "I bought this for my husband who plays the piano. He is having a wonderful time playing these old hymns. The music is at times hard to read because we think the book was published for singing from more than playing from. Great purchase though!",
 "overall": 5,
 "summary": "Heavenly Highway Hymns",
 "unixReviewTime": 1252800000,
 "reviewTime": "09 13, 2009"
}
```


where

- reviewerID - ID of the reviewer, e.g. A2SUAM1J3GNN3B
- asin - ID of the product, e.g. 0000013714
- reviewerName - name of the reviewer

- helpful - helpfulness rating of the review, e.g. 2/3
- reviewText - text of the review
- overall - rating of the product
- summary - summary of the review
- unixReviewTime - time of the review (unix time)
- reviewTime - time of the review (raw)

Metadata

Metadata includes descriptions, price, sales-rank, brand info, and co-purchasing links:

- metadata (3.1gb) - metadata for 9.4 million products

Sample metadata:


```
{
 "asin": "0000031852",
 "title": "Girls Ballet Tutu Zebra Hot Pink",
 "price": 3.17,
 "imUrl": "http://ecx.images-amazon.com/images/I/51fAmVkTbyL._SY300_.jpg",
 "related": {
        "also_bought": [
        "B00JHONN1S",
        "B002BZX8Z6",
        "B00D2K1M3O",
        "0000031909",
        "B00613WDTQ",
        "B00D0WDS9A"
        ],
        "also_viewed": [
        "B002BZX8Z6",
        "B00JHONN1S",
        "B008F0SU0Y",
        "B00D23MC6W",
        "B00AFDOPDA",
        "B00E1YRI4C",
        "B002GZGI4E",
        "B003AVKOP2"
        ],
        "bought_together": [
        "B002BZX8Z6"
        ]
 },
 "salesRank": {
        "Toys & Games": 211836
 },
 "brand": "Coxlures",
 "categories": [
        [
        "Sports & Outdoors",
        "Other Sports",
        "Dance"
        ]
 ]
}
```

where

- asin - ID of the product, e.g. 0000031852
- title - name of the product
- price - price in US dollars (at time of crawl)
- imUrl - url of the product image
- related - related products (also bought, also viewed, bought together, buy after viewing)
- salesRank - sales rank information
- brand - brand name
- categories - list of categories the product belongs to

Visual Features

Visual features were extracted from each product image using a deep CNN (see citation below). Image features are stored in a binary format, which consists of 10 characters (the product ID), followed by 4096 floats (repeated for every product). See files below for further help reading the data.

- visual features (141 gb) - visual features for all products

The images themselves can be extracted from the imUrl field in the metadata files.

Per-Category Files

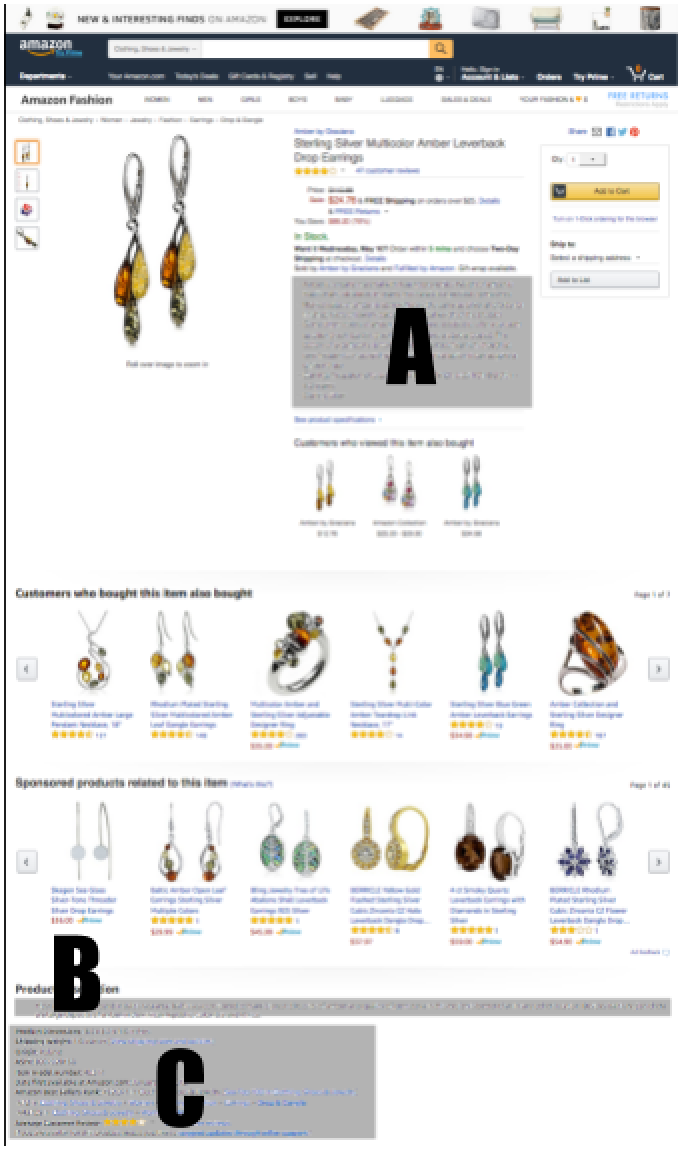
Below are files for individual product categories, which have already had duplicate item reviews removed. In addition, each per-category file also has an associated image features file. See Appendix for table of actual file names.

Category	Reviews Count	Metadata Count
Books	22,507,155	2,370,585
Electronics	7,824,482	498,196
Movies and TV	4,607,047	208,321
CDs aand Vinyl	3,749,004	492,799
Clothing, Shoes and Jewelry	5,748,920	1,503,384
Home and Kitchen	4,253,926	436,988
Kindle Store	3,205,467	434,702
Sports and Outdoors	3,268,695	532,197
Cell Phones and Accessories	3,447,249	346,793
Health and Personal Care	2,982,326	263,032
Toys and Games	2,252,771	336,072
Video Games	1,324,753	50,953
Tools and Home Improvement	1,926,047	269,120
Beauty	2,023,070	259,204

Apps for Android	2,638,173	61,551
Office	1,243,186	134,838
Pet Supplies	1,235,316	110,707
Automotive	1,373,768	331,090
Grocery and Gourmet Food	1,297,156	171,760
Patio, Lawn and Garden	993,490	109,094
Baby	915,446	71,317
Digital Music	836,006	279,899
Musical Instruments	500,176	84,901
Amazon Instant Video	583,933	30,648

Data Collection

The original data sources focused on attributes related to user purchase history and reviews. Textual product features did not exist. These include features such as product description, bulleted or highlighted features, and product specifications. These textual features were collected using the web scraping method outlined under *Data Preparation*.



Amazon Product Page Text Features:
A) Bulleted Features, B) Product Description, C) Product Information

Data Preparation

Data Quality Issues

The features price and brand from the metadata were very sparsely populated. Missing values were addressed through web scraping.

Web Scraping

New and missing data points were acquired by scraping the available data directly from each Amazon product web page. The legacy code used to acquire the data initially no longer worked because of changes in Amazon's webpage layouts over time and Amazon's anti-scraping policies.

Amazon's anti-scraping policies consist of aggressive IP bans as well as the implementation of CAPTCHAs -- *C(ompletely) A(utomated) P(ublic) T(uring) (Test to Tell) C(omputers and)H(umans) A(part)*. CAPTCHAs are tests used to tell if a user is human or a computer program. The result is that it is impossible to scrape Amazon large amounts of Amazon data undetected. Nevertheless, we were successful in scraping the data undetected using several techniques.

Challenges

They key to any web scraping project is to be undetectable. Web scraping behavior is easily distinguishable from normal human web browsing activities. Typical distinguishing factors include stark differences in page requests rates and user behaviors (while on a given page). A typical user may spend several minutes reviewing a product and clicking on related pages -- researching it's features and reading reviews, while a bot will typically just waits for the page to finish loading and moves on to the next request. This type of behavior detection is accomplished through the use of identifying information inherent in every web page request. This information typically consists of the user's IP address, user-agent string, and/or cookies.

Techniques

Several techniques were employed to overcome the aforementioned challenges:

- Public Proxies
- Custom User-Agent Strings
- Cookie Handling

Public Proxies

Proxies provide an effective way of anonymizing internet traffic. They send and receive page requests on behalf of the user. The net effect is that the server has no knowledge of the originating user's IP address. A pool of proxies was created by identifying 50 public proxies from around the world. Requests were randomly routed through this pool effectively masking the single origin of all of our requests while allowing us to increase our page requests rate by distributing the requests over many proxies simultaneously. Also, the effects of any IP-bans are mitigated because of both the size of the pool, as well as proxy reputation. This is because IP address assignments are public knowledge. As a result ephemeral IPs may face more scrutiny and be treated more aggressively by anti-scraping policies.

User-Agent Strings

User-agent strings are passed to the web server as part of every request. They typically identify the user's operating system of the user, web browser, and eversion of each. This combination also allows web servers to distinguish between whether the user is using a desktop computer mobile device. Because both the operating system market and browser market are largely dominated by a few companies, generating a random string is not enough as it would be easy to identify anomalous or invalid user-agents. In the course of testing, it was evident that randomly generating valid-user agent strings was not enough -- more current OS and browser versions were more likely to pass detection than variants that were 2-3 versions older. To counter this heuristic, up-to-date web traffic history was sourced from w3-schoools.com to create random user-agent string generator whose output statically matched current traffic distributions. This ensured that despite any spikes in traffic from any single proxy, the distribution of user-agent strings would still appear normal.

Cookie Handling

Cookies are text files that allow web servers to track individual users and their behavior. They often encode individual user preferences, can track a user's browsing history and are widely used throughout the web for tracking users and analyzing their behavior. This can pose a significant challenge when trying to scrape data because not accepting cookies can be red-flag, while accepting a server's cookies can easily reveal non-human behaviors. To overcome this problem, our spider employed the following characteristics:

- Each product request simulated a brand new user session
    - Clear existing cookies
    - Accept only new cookies
    - Ignore cookie update requests

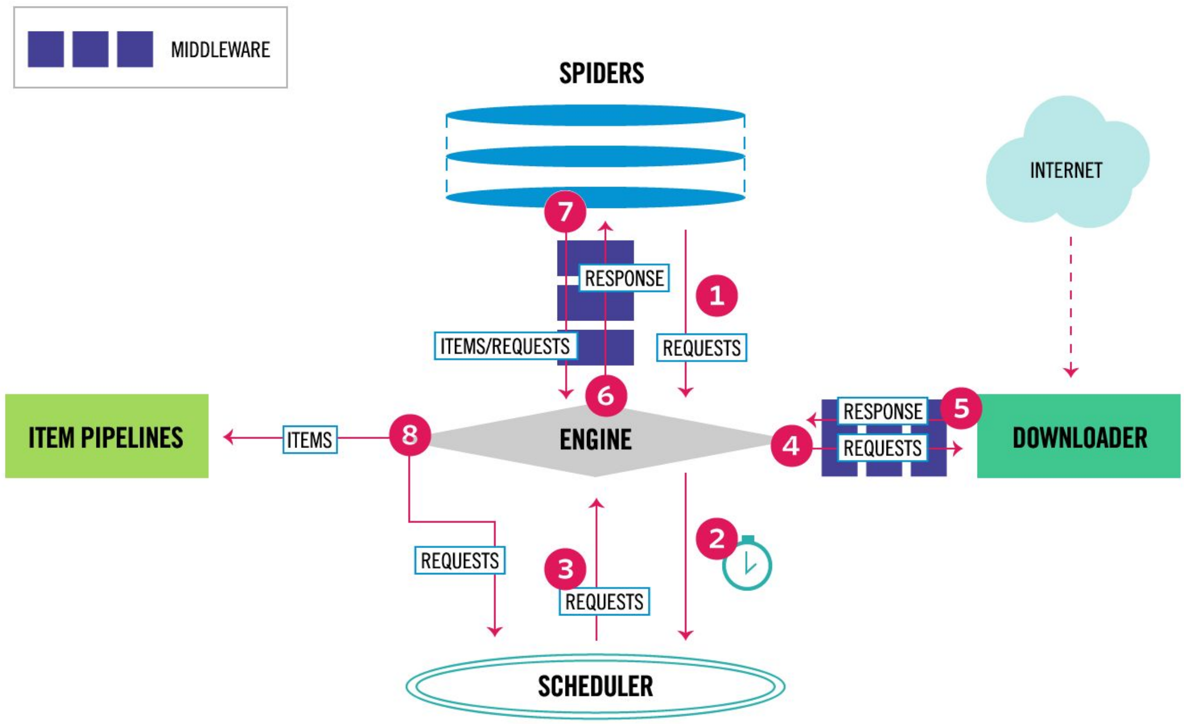
This allowed our spider to make thousands of page requests each appearing as a first time Amazon user.

Technical Implementation

The scraper is built using using the Scrapy framework. Scrapy was chosen because the engine is built on top the Twisted reactor, which allows for event dispatching and concurrency. The result is a scalable solution that allows for hundreds of thousands of requests.

A custom spider was developed to scrape Amazon product pages. Custom middlewares were also created to manipulate requests as described previously.

Architecture Overview



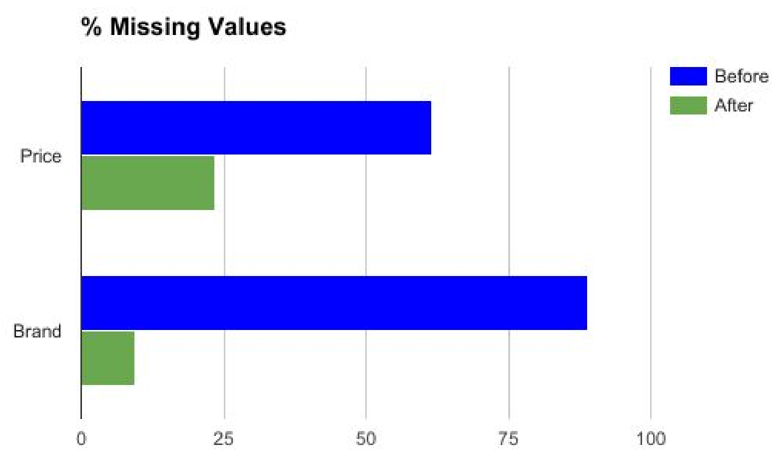
The data flow in Scrapy is controlled by the execution engine, and goes like this:

1. The Engine gets the initial Requests to crawl from the Spider.
2. The Engine schedules the Requests in the Scheduler and asks for the next Requests to crawl.
3. The Scheduler returns the next Requests to the Engine.
4. The Engine sends the Requests to the Downloader, passing through the Downloader Middlewares (see process_request()).
5. Once the page finishes downloading the Downloader generates a Response (with that page) and sends it to the Engine, passing through the Downloader Middlewares (seeprocess_response()).
6. The Engine receives the Response from the Downloader and sends it to the Spider for processing, passing through the Spider Middleware (see process_spider_input()).
7. The Spider processes the Response and returns scraped items and new Requests (to follow) to the Engine, passing through the Spider Middleware (see process_spider_output()).
8. The Engine sends processed items to Item Pipelines, then send processed Requests to the Scheduler and asks for possible next Requests to crawl.
9. The process repeats (from step 1) until there are no more requests from the Scheduler.

Web Scraping Results

The combination of these techniques allow our scraper to remain anonymous while simultaneously blending in with normal web traffic. This “anonymization gauntlet” allowed our scraper to achieve a CAPTCHA rate of approximately 0.006%. All of these “CAPTHCA’d” requests were eventually successful as each retried request is in essence a new anonymized request..

Each product page was saved and archived for analysis and post processing. All missing data elements were captured or verified to be unavailable on the corresponding saved page. Features for our model were extracted from these saved pages as part of our feature engineering effort.

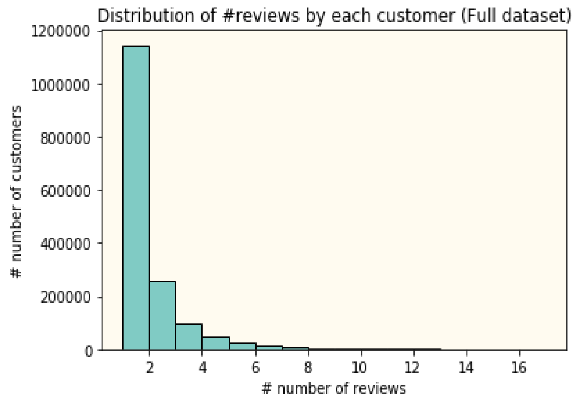


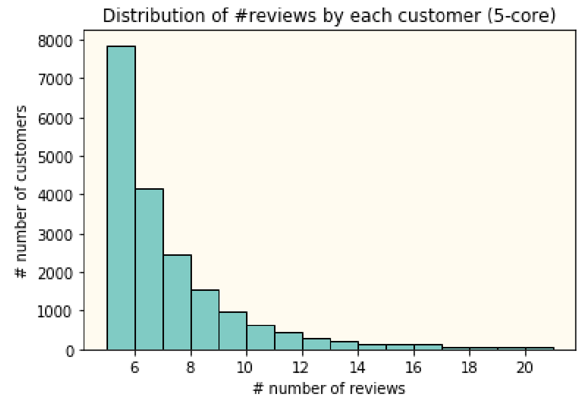
Data Transformations

Exploratory Analysis

We focused only on the Women’s category for this modeling exercise. We explored different aspects of the data to understand their distribution and identify any seasonality which could be beneficial for our modeling exercise.

First, we tried to understand how the ‘full data’ set is different from the ‘5-core’ data set. The histogram of number of reviews shows a long tailed distribution. ‘5-core’ data forms about 5% of the ‘full data’.

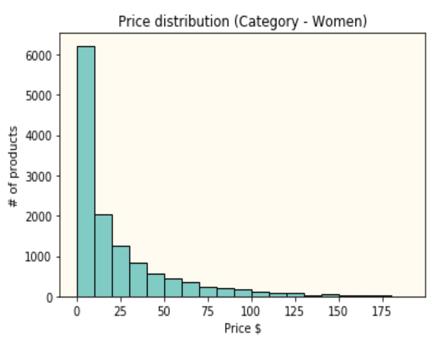


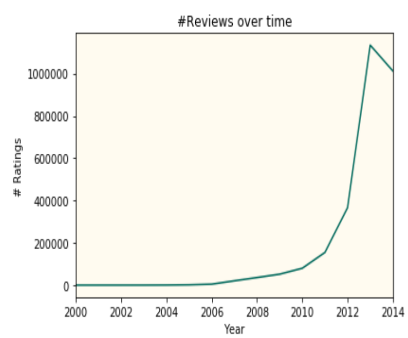


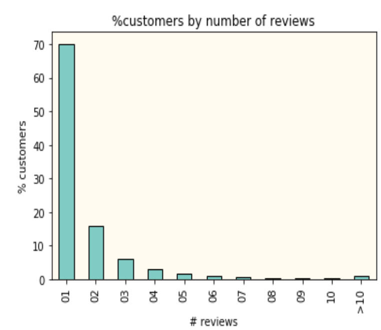
Then the graphs below (generated on the full data set) were used to obtain further insight into the data.

1. Most of the products are priced below \$100 - this came in handy in defining the number of categories during feature engineering
2. Most of the data available is after 2010. Having the latest data is beneficial because tastes and trends change over time
3. 70% of the reviews are from the customers who write just 1 review - This proves that reviews cannot be relied upon to build the model. Utilize implicit feedback where a purchase implies liking

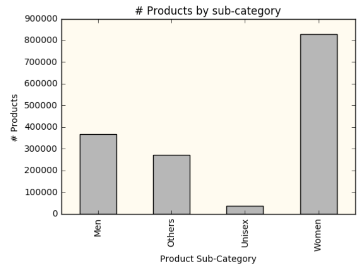
Note that similar trends were observed in 5-core data.

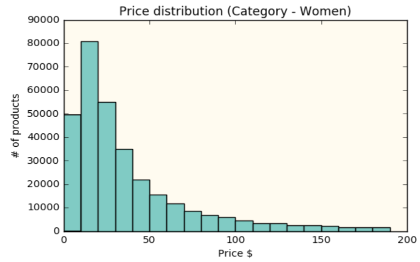


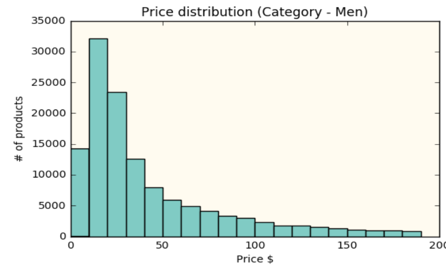




Further, we drilled down into 5-core data to understand more trends. We found that the product collection is skewed towards women's wear. Also, pricing distribution of men and women are similar







Feature Engineering

Three major categories of features were used in the model to augment the visual features. We call these new features as ‘Non Visual’ features. The three feature categories are:

a. Price features
b. Brand features
c. Product description

Price Features

Price of an item has a great influence on any purchase. We wanted to leverage this important feature to improve the performance of the BPR model. Based on the exploratory analysis we know that prices of products in women’s category varies from 1 to 100. So we created 10 categories of binary variables. For example:

Item	Price <10	Price 11 to 20	Price 21 to 30	Price 31 to 40	Price 41 to 50	Price 51 to 60	Price 61 to 70	Price 71 to 80	Price 81 to 90	Price >91
1	0	1	0	0	0	0	0	0	0	0
2	0	0	0	0	1	0	0	0	0	0
	0	0	0	0	0	0	0	0	1	0
n	0	0	1	0	0	0	0	0	0	0

Preparation

Price as a feature is extracted from one of several areas on the Amazon product page. The location and format of the price can vary based on page creation date and product category. It can appear in the form of Sale Price, Original Price or simply Price. These prices can also appear as ranges as Amazon is a marketplace of sellers selling the same product. When more than one price is available, i.e. a price range or a sale price and original price, the following rules are used to determine the price that is used in the model:

1. When multiple types of prices are available the first available price is used using the following order: Sale Price, Price, Original Price.
2. When a range or multiple ranges are available, the average of all available ranges is used.

Brand Features

Brand loyalty is a very well known phenomenon. Customers tend to shop their favourite brands often. Also, customers who like a particular brand tend to like similar brands. We wanted to capture this by including brand as a binary variable feature. For example:

Item	Crazy for Bargains	Serenity Crystals	Mango	DKNY	New Balance	Reebok	Nike	Bebe	Adidas	Swatch	…….
1	0	1	0	0	0	0	0	0	0	0	
2	0	0	0	1	0	0	0	0	0	0	
	0	0	0	0	0	0	1	0	0	0	
n	1	0	0	0	0	0	0	0	0	0	

There were about 1992 brands in the 5-core dataset. So, we have created a 1992 long vector for each product as a feature.

Product Description

To capture more characteristics of a product we leveraged the textual data. On exploring the 'Product Title' field and looking at term frequency of different n-grams it was found that 2-grams (not 1, not 3) had the useful information. Some of the top 2-grams and their frequency is given in the table below:

Bi-grams	Term frequency
sterling silver	963
plus size	331
long sleeve	245
running shoe	231
stainless steel	230
pendant necklace	227
stud earrings	211
cubic zirconia	197
925 sterling	150
sandal black	144

4525 bigrams were used as a feature vector of binary values as shown below.

Item	Sterling silver	Plus size	Long sleeve	Running shoe	Stainless steel	Pendant necklace	Stud earrings	Cubic zirconia	925 sterling	Sandal black	…….
1	0	1	0	0	0	0	0	0	0	0	
2	0	0	0	1	0	0	0	0	0	0	
	0	0	0	0	0	0	1	0	0	0	
n	1	0	0	0	0	0	0	0	0	0	

Modeling Recommender Systems

In this project, we build a personalized ranking model on top of a visual personalized ranking model (VBPR) to uncover non-visual (NVBPR) and hybrid of visual and non-visual (HBPR) dimensions simultaneously. We first formulate the task in question and introduce our Matrix Factorization based predictor function. Then we develop our training procedure using a Bayesian Personalized Ranking (BPR) framework.

Question Formulation:

Letting *U* and *I* denote the set of users and items respectively, each user u is associated with an item set I_u^+ about which u has expressed explicit positive feedback. In addition, a single vector of features (visual and non-visual features) is available for each item i $\in$ I. Our objective is to generate for each user u a personalized ranking of those items about which they haven't yet provided feedback (i.e. $I \setminus I_u^+$).

BPR model:

The preference predictor that we use is built on top of Matrix Factorization (MF), which is state-of-the-art for rating prediction as well as modeling implicit feedback, whose basic formulation assumes the following model to predict the preference of a user u toward an item i [1]:

$$x(u,i) = \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i$$

Where:

α is the popularity of item

β_u measures how much does this user tend to rate things above the mean

β_i measures how does this item tend to receive higher ratings than others

$\gamma_u \cdot \gamma_i$ is the projection of the rating matrix (R) into a low dimensional space (K) and is an indication of 'compatibility' between the user u and the item i

VBPR model:

Although theoretically latent factors are able to uncover any relevant dimensions, one major problem it suffers from is the existence of 'cold' (or 'cool') items in the system, about which there are too few associated observations to estimate their latent dimensions. Using explicit features can alleviate this problem by providing an auxiliary signal in such situations. In particular, He et al. in [9] propose to partition rating dimensions into visual factors and latent factors as:

$$x(u,i) = \alpha + \beta_u + \beta_i + \gamma_u \,.\, \gamma_i + \theta_u^T \,.\, \theta_i$$

where α, β, and γ are as before and θu and θ_i are newly introduced visual factors whose inner product models the visual interaction between u and i, i.e., the extent to which the user u is attracted to each of visual dimensions. As the dimensions of θu and θ_i can be very large, learning these many parameters for large datasets are not usually feasible. Instead, He et al. in [9] propose to learn an embedding kernel which linearly transforms such high-dimensional features into a much lower-dimensional (D) 'visual rating' space:

$$\theta_i = Ef_i$$

where E is an matrix embedding visual feature space (F-dimensional) into visual space (D-dimensional), and f_i is the original visual feature vector for item i. The numerical values of the projected dimensions can then be interpreted as the extent to which an item exhibits a particular visual rating facet. This embedding is efficient in the sense that all items share the same embedding matrix which significantly reduces the number of parameters to learn. As another dimension of product has been added to the model, it is necessary to also introduce a visual bias term β' whose inner product with f_i models users' overall opinion toward the visual appearance of a given item. In summary, the final prediction model is:

$$x(u,i) = \alpha + \beta_u + \beta_i + \gamma_u \,.\, \gamma i + \theta_u^T \,.\, (Ef_i) + \beta'^T f_i$$

The visual features for each item i in our dataset f_i is extracted using the Caffe reference model [11] which implements a CNN architecture that has 5 convolutional layers followed by 3 fully-connected layers, and has been pre-trained on 1.2 million ImageNet (ILSVRC2010) images. In our experiments, we take the output of the second fully-connected layer (i.e. FC7), to obtain an F = 4096 dimensional visual feature vector f_i .

NVBPR/HBPR Model:

The visual features mentioned in the previous section can help with better personalized recommendations. This is especially true when the visual aspects of the product can have a big impact on the user's opinion (such as clothing). However, for products where visual features might not have a profound impact on user's opinion (such as books), the visual features might not be advantageous. For such scenarios we propose to use non-visual features directly related to products (such as price, brand and product description). These features can be used independently to build a recommender based on non-visual features (NVBPR) or in conjunction with visual features to build a hybrid recommender system (HBPR). These features can be added to the model as separate terms and their parameters can be learnt independently. However, as the nature of the non-visual features are similar to the visual features, we will model them as in VBPR:

$$x(u,i) = \alpha + \beta_u + \beta_i + \gamma_u \,.\, \gamma i + \theta_u^T \,.\, (Ef_i) + \beta`^T f_i$$

Where f_i is the non-visual extracted features of the product or its combinations with the original visual feature vector for item i. The numerical values of the projected dimensions can then be interpreted as the extent to which an item exhibits a particular non-visual/visual rating facet.

The non-visual features consists of price, brand, and product description and as mentioned in the previous section is vectorized to form a binary vector f_i. In the case of hybrid BPR, this binary vector will be appended to the non-binary vector of image features.

Model Learning Using BPR

Bayesian Personalized Ranking (BPR) is a pairwise ranking optimization framework which adopts stochastic gradient ascent as the training procedure. A training set D_S consists of triples of the form (u, i, j), where u denotes the user together with an item i about which they expressed positive feedback, and a non-observed item j:

$$D_S = \{(u, i, j) | u \in U \wedge i \in I_u^+ \wedge j \in I \setminus I_u^+ \}$$

Following the notation in [4], Θ is the parameter vector and x_{uij} (Θ) denotes an arbitrary function of Θ that parametrizes the relationship between the components of the triple (u, i, j). The following optimization criterion is used for personalized ranking (BPR-OPT):

$$\sum_{X\,(u,i,j) \in DS} \ln\sigma(x_{uij}) - \lambda_\Theta ||\Theta||^2$$

where σ is the logistic (sigmoid) function and λ_Θ is a model specific regularization hyperparameter.

When using Matrix Factorization as the preference predictor (i.e., BPR-MF), x_{uij} is defined as

$$x_{uij} = x_{u,i} - x_{u,j}$$

where $x_{u,i}$ and $x_{u,j}$ are defined as before. BPR-MF can be learned efficiently using stochastic gradient ascent. First a triple (u, i, j) is sampled from D_S and then the learning algorithm updates parameters in the following fashion:

$$\Theta \leftarrow \Theta + \eta \cdot (\sigma(-x_{uij}) \ \frac{\partial xuij}{\partial \Theta} \ - \lambda_\Theta \Theta)$$

where η is the learning rate.

As we will see later, instead of learning each parameters b y deriving the closed form of their update, we will use TensorFlow and its auto-derivative to find learn the parameters of our model.

Findings

Preliminary Findings

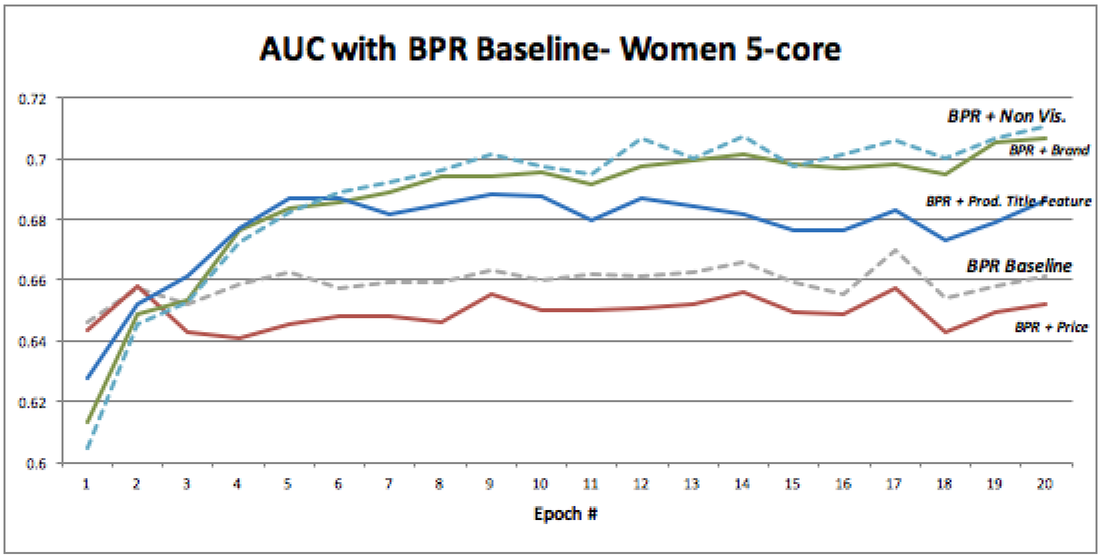
To quantify the performance of our proposed model, we performed experiments on two main categories of Amazon datasets, namely Women Clothing and Mobile. These datasets include a variety of settings where either visual appearance or non-visual product details are expected to play a role in consumers' decision-making process.

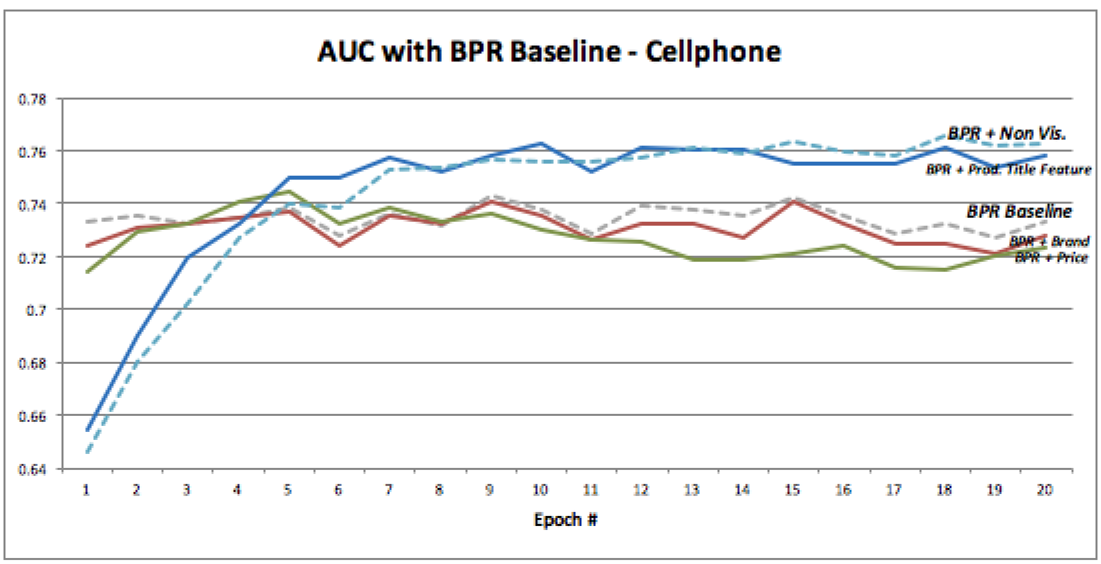
Baseline Model:

In order to have a framework to evaluate our recommender against, we first must establish a baseline. Although random (randomly recommend products to users) or most popular (recommend the most popular items to users) can be used as a baseline for performance, we instead chose BPR as our baseline for performance since the performance of random and most popular are often bad. We are using the Area Under the ROC Curve (AUC) standard measurement as variable to quantify accuracy. We use two main datasets to evaluate the performance of our models: Amazon Women datasets and Amazon Phone datasets.

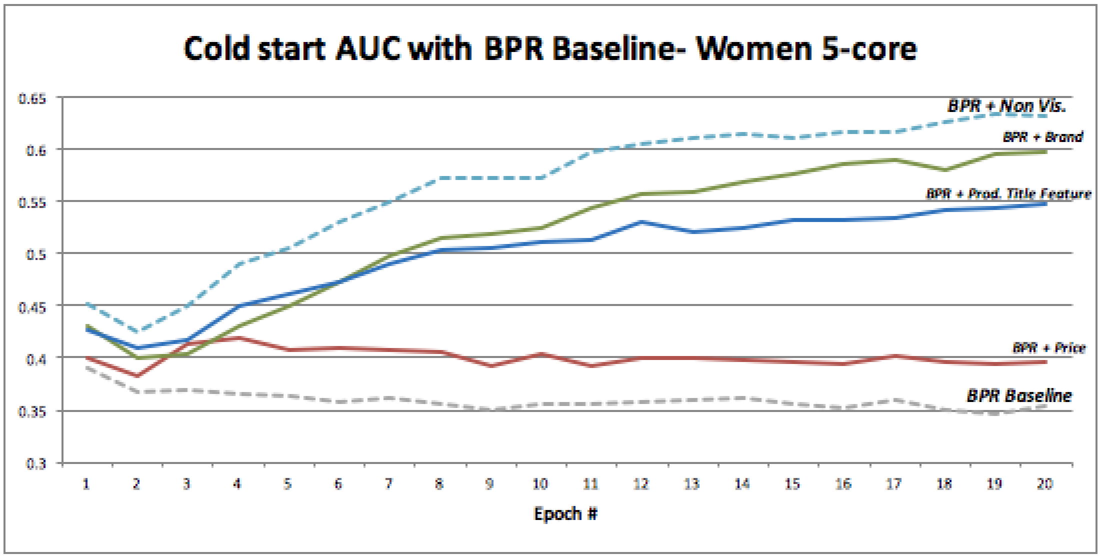
Advance BPR with non-Visual Features (NVBPR) Evaluation (on 5-core datasets):

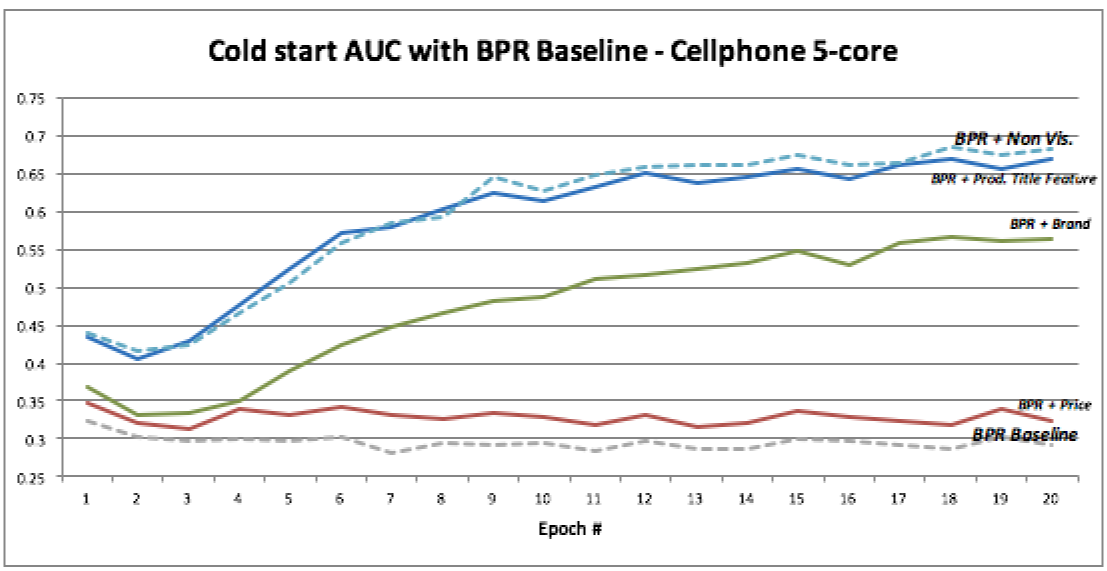
As mentioned in previous sections, we engineer three features from the main product characteristics (price, brand, product description) and add them separately to the model and evaluate the AUC of each model. Other than price, all other features (brand and product description) is showing significant and meaningful gain over the plain BPR. We also combine all these three non-visual feature which gives us the best performance in terms of AUC as illustrated below for Women and Mobile 5-core datasets:





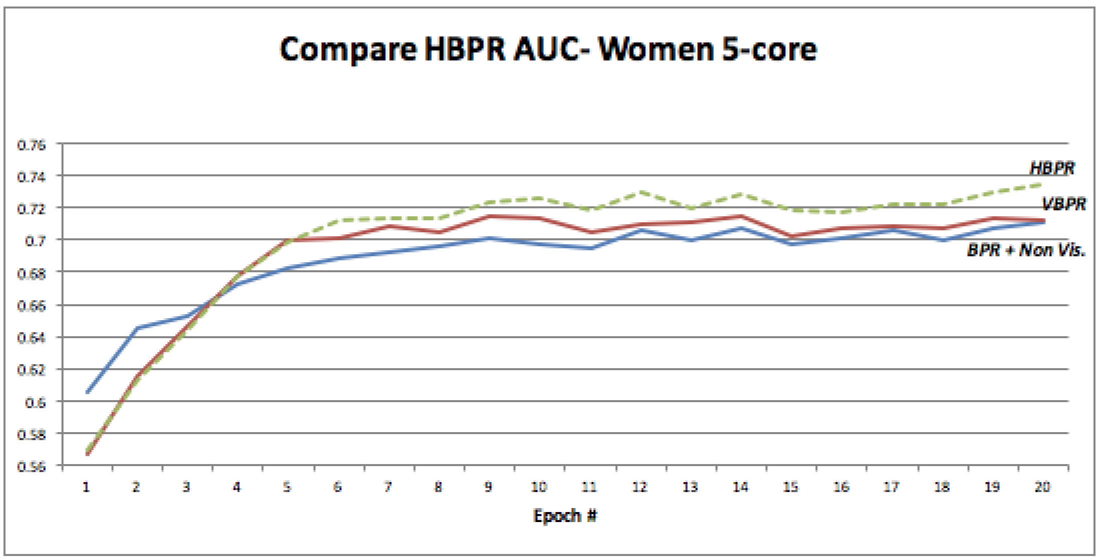
The cold start issue was also evaluated and in this case each of the three added features (price, brand, product description) are showing significant gain compared to the cold start performance of plain BPR. This is expected as for cold items, the context that our non-visual features add should help with recommending products to users based on the common characteristics of the products that they have purchased in the past. The cold start AUC performance is illustrated below:

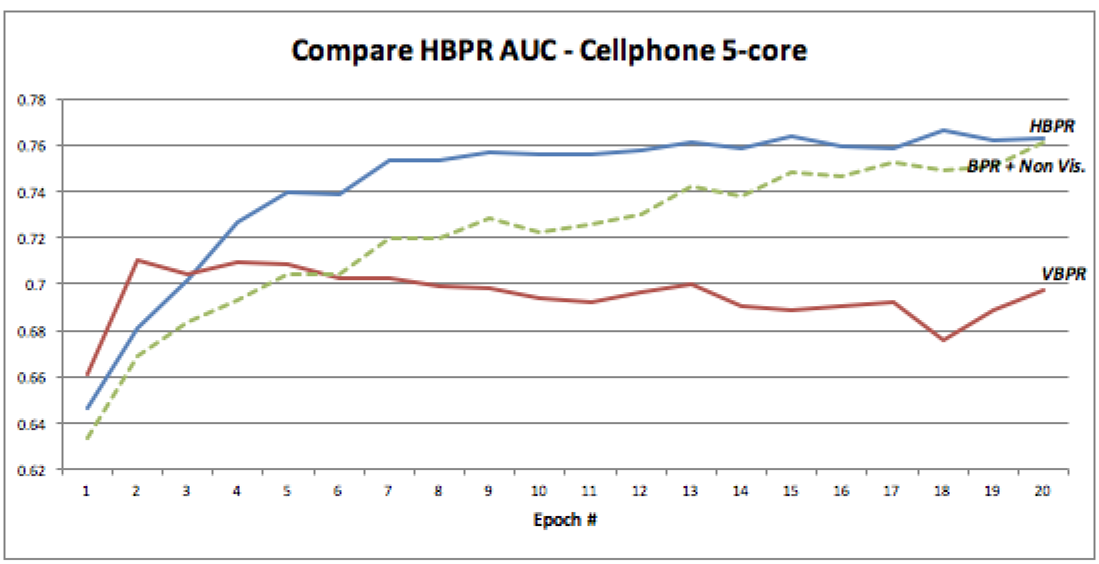




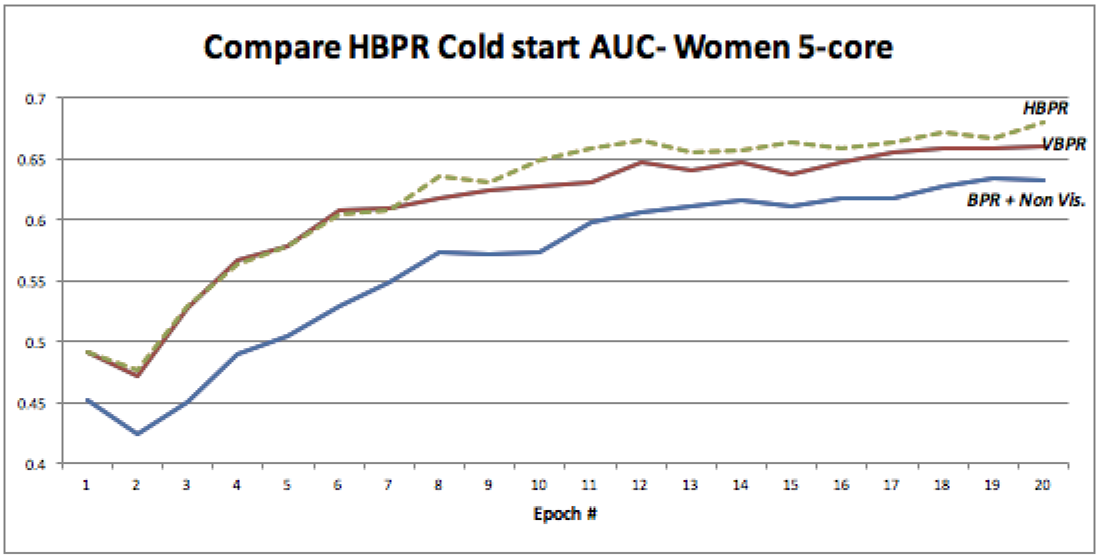
Advanced BPR modeling with hybrid of visual and nonvisual features (on 5-core datasets):

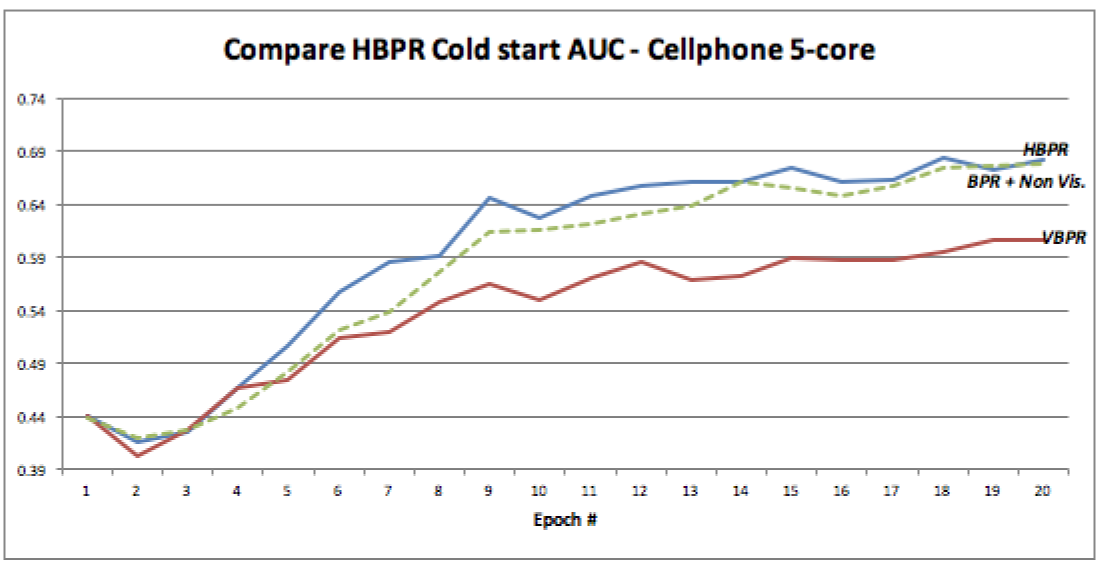
As the next logical step and in pursue of better gains, we combined all three engineered features from the main product characteristics (price, brand, product description) with the visual features of the products and add them jointly to build a hybrid model (HVBR) and evaluate the AUC of this model. We also ran the model with only visual features (VBPR) and used that as a baseline for our hybrid model. The hybrid model (visual+non-visual features) gives us the best performance in terms of AUC as illustrated below for Women and Mobile 5-core datasets:





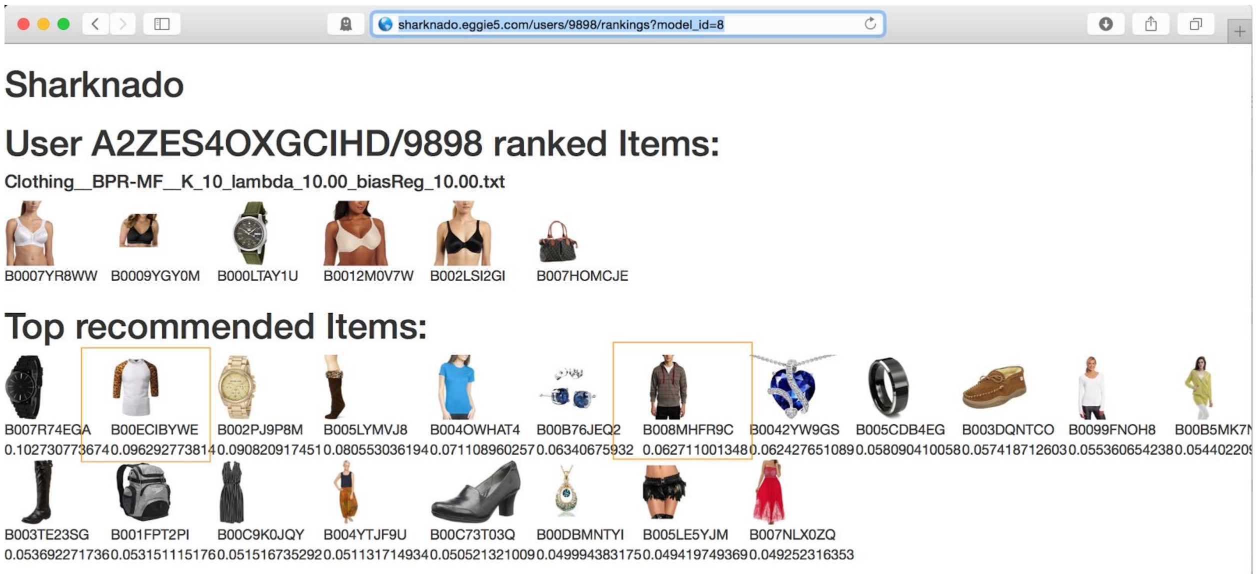
The cold start issue was also evaluated for both the VBPR and HBPR model and for the dataset where visual appearance has more impact on people's opinion (Women clothing) we still VBPR outperforming NVBPR, however, as expected, for Phone dataset where visual features might not be as significant (compared to product features, brand,...), the NVBPR outperforms VBPR. In both cases, however, we see that the hybrid of both visual and non-visual features (HBPR) is either comparable or outperform both VBPR and NVBPR. The cold start AUC performance is illustrated below:





The Visualization Web app:

In order to help visualize recommendation quality a tool was developed that shows the top 10 recommendations for a given user:

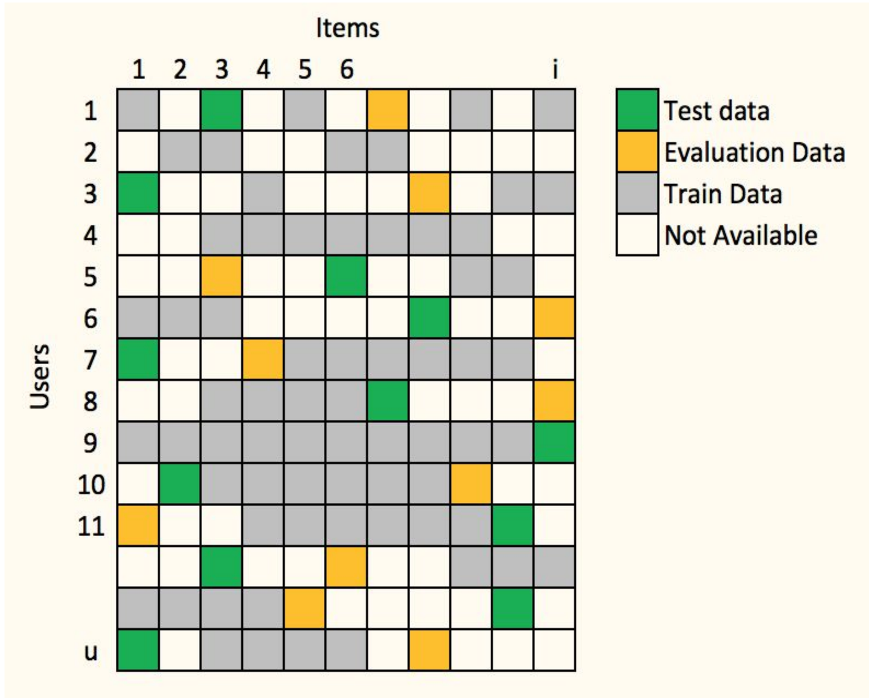


Screenshot of QA util highlighting a potentially inaccurate recommendation of a men's item to a probable woman.

Performance and Evaluation

Performance Evaluation Methodology

The split of the data into training/validation/test sets is done by selecting for each user u a random item to be used for validation V_u and another for testing T_u. All remaining data is used for training. This data splitting strategy has been illustrated in the figure below:



The predicted ranking is evaluated on T_u with the widely used metric AUC (Area Under the ROC curve):

$$AUC = \frac{1}{|U|} \sum_{u} \frac{1}{|E(u)|} \sum_{(i,j)\in E(u)} \delta(x_{u,i} > x_{u,j})$$

where the set of evaluation pairs for user u is defined as:

$$E(u) = \{(i, j) | (u, i) \in T_u \wedge (u, j) \notin (P_u \cup V_u \cup T_u)\}$$

and δ(b) is an indicator function that returns 1 iff b is true. In all cases we report the performance on the test set T for the hyperparameters that led to the best performance on the validation set V.

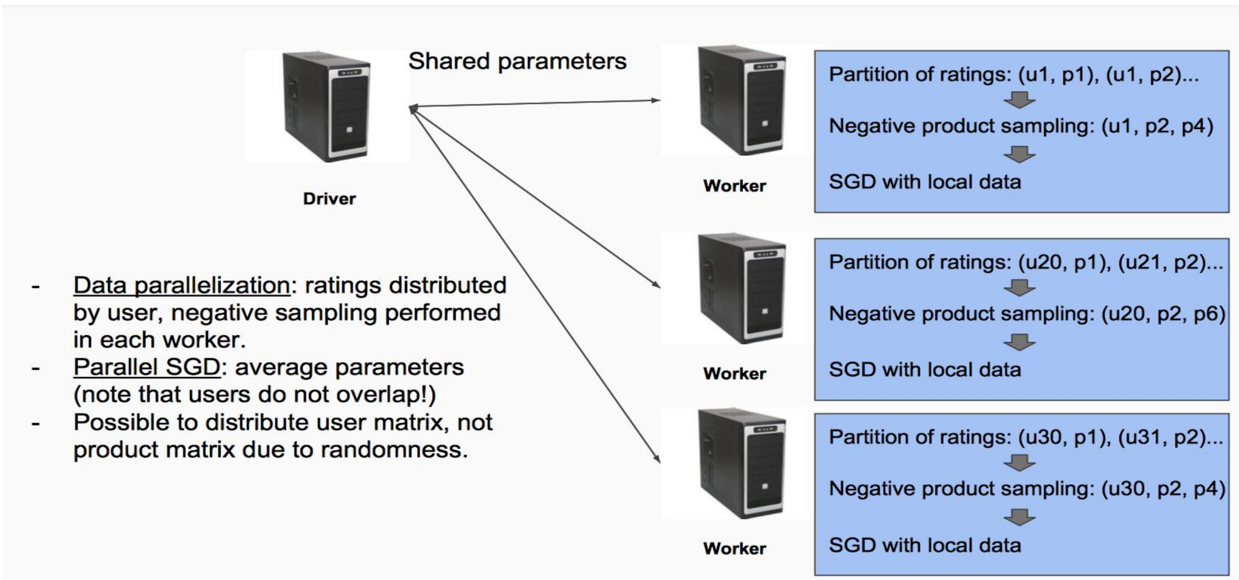
Scalability

Although the efficiency of the underlying BPR-MF makes our models similarly scalable, the serialized way of updating parameters in BPR makes it somewhat slow on very large datasets.

Our first implementation of the baseline BPR was in pure Python. This implementation, although simple, was taking around 15 minutes to run on a fast laptop machine (and databricks server) on the smaller 5-core dataset. This was alarming as our advanced models had more complexity and parameters to learn and the dataset was going to be larger and hence the cost (in terms of training time) was going to be very high if we were going to stick with the pure Python implementation and the model were not scalable. Therefore we started to explore more scalable approaches. We specifically explore Spark, Theano, and TensorFlow and based on the performance and complexity of implementation of each we ended up picking TensorFlow over the other options. In the next sections we will go over the details of these approaches.

Spark

As mentioned before BPR is not parallelizable in nature due to sequence to sequence nature of updating the variables. So implementing the native BPR in spark (PySpark specifically) was not feasible. Instead, we used the paradigm in [17], which includes two main steps of Data Parallelization and Parallel Stochastic Gradient Descent (SGD)explained in the figure below:

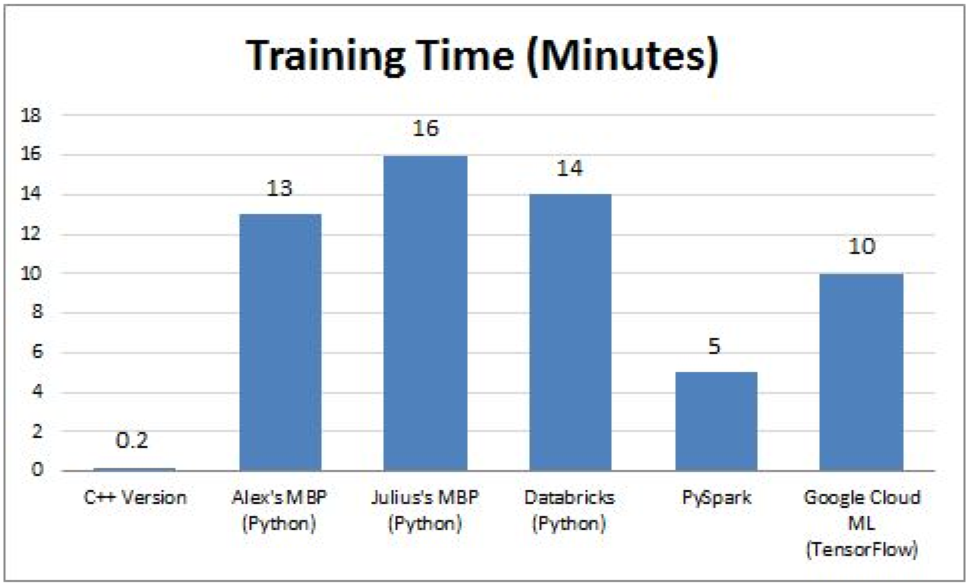


This architecture gave us a very good advantage (3x faster compared to native Python implementation) in running the BPR on 5-core Women dataset. However, the complexity of the PySpark implementation of this parallelized approach (as expected) was very high and extending the BPR model to more advanced BPR (VBPR, NVBPR, HBPR) would have added a huge deal of complications and for this reason we decided not to proceed with Spark.

Theano

TensorFlow Model

The plain python version of our model has a CPU bottleneck that has a large negative performance hit during the expensive AUC calculation. The AUC calculation is potentially highly parallelizable, however, this can’t be exploited in plain python. In order to possibly optimize the training time of our python model, we attempted to implement a version that runs on GPUs using the Tensorflow framework. This is still a work in progress as our Tensorflow model trains slower than the python version.



Training time across various implementations

Budget

The team was allotted $2000 in AWS credits for this project. $600 dollars went to a very expensive lesson: don't forget to spin down your spark instances. We learned early on that our model did not scale well horizontally and that Spark would not work for us. The decision to use Tensor Flow and Google Cloud freed us from budgetary constraints as $300 dollars of free credits were available for using Google Cloud.

Conclusions

In selecting a product, depending on the category of the product, either visual (image of product) or non-visual (price, brand, product description) factors influence many of the choices people make. In this project, we investigated the usefulness of non-visual features for personalized ranking tasks on implicit feedback datasets where visual features might not be advantageous. We proposed a scalable method that can incorporates both non-visual features extracted from the product details or visual features extracted from product images into Matrix Factorization, in order to uncover the hidden non-visual and visual dimension that most influence people's behavior. Our model is trained with Bayesian Personalized Ranking (BPR) using stochastic gradient ascent in Tensorflow. Experimental results on Amazon datasets demonstrate that we can significantly outperform state-of-the-art ranking techniques and significantly alleviate cold start issues by using non-visual features available and extracted from the product details. We also combined the non-visual features with visual features and showed that a hybrid combinations of these two features can beat the performance of a recommender system that only use one of these feature sets.

Future Work

This project can be continued in several different direction. Tuning the current model (tuning number of quantized levels for price, tuning the number of elements to use from product description,...) can add more gains to the performance of our recommender system. Another novel way to extend this project is to incorporate item grouping into our recommender systems [16]. Extending our model with temporal dynamics to account for the drifting of price/fashion tastes over time is another interesting future direction to extend this project.

References

1. Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. IEEE Computer, 42(8):30–37, 2009.
2. Overview of Recommender Systems (Lecture 4), available online at: http://cseweb.ucsd.edu/classes/fa15/cse255-a/
3. Overview of Recommender Systems, available online at: http://michael.hahsler.net/research/Recommender_SMU2011/slides/Recomm_2011.pdf
4. S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI 2009), 2009.
5. Schein, A.; Popescul, A.; Ungar, L.; and Pennock, D. 2002. Methods and metrics for cold-start recommendations. In SI- GIR.
6. Bao, Y.; Fang, H.; and Zhang, J. 2014. Topicmf: Simultaneously exploiting ratings and reviews for recommendation. In AAAI.
7. Qiao, Z.; Zhang, P.; Cao, Y.; Zhou, C.; Guo, L.; and Fang, B. 2014. Combining heterogenous social and geographical information for event recommendation. In AAAI.
8. Brown, P.; Bovey, J.; and Chen, X. 1997. Context-aware applications: from the laboratory to the marketplace. IEEE Wireless Communications.
9. R. He and J. McAuley. Vbpr: Visual bayesian personalized ranking from implicit feedback. CoRR, 2015.

10. Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In WWW, 2016.
11. Jia, Y. Caffe: An open source convolutional architecture for fast feature embedding: http://caffe.berkeleyvision.org/, 2013.
12. Hu, Y.; Koren, Y.; and Volinsky, C. 2008. Collaborative filtering for implicit feedback datasets. In ICDM. IEEE.
13. Zhao, T.; McAuley, J.; and King, I. 2014. Leveraging social connections to improve personalized ranking for collaborative filtering. In CIKM.
14. Lu, Z.; Dou, Z.; Lian, J.; Xie, X.; and Yang, Q. 2015. Content-based collaborative filtering for news topic recommendation. In AAAI.
15. Kanagal, B.; Ahmed, A.; Pandey, S.; Josifovski, V.; Yuan, J.; and Garcia-Pueyo, L. 2012. Supercharging recommender systems using taxonomies for learning user purchase behavior. VLDB Endowment.
16. Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. 2015b. Image-Based Recommendations on Styles and Substitutes. In Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '15). ACM, New York, NY, USA, 43–52.
17. Alfredo Lainez Rodrigo, Luke de Oliveira, Distributed Bayesian Personalized Ranking in Spark. Available at: https://stanford.edu/~rezab/classes/cme323/S16/projects_reports/rodrigo_oliveira.pdf
18. Simple matrix factorization in tensorflow: http://katbailey.github.io/post/from-both-sides-now-the-math-of-linear-regression/

Appendix

Reproducibility of Results

Our source code is on github: https://github.com/DSE-capstone-sharknado See the bpr project and the models-legacy project for python and c++ implmentations of matrix factorization respectively. See the respective project readmes for information on how to train the models. We found setting the regularization parameters to 1 achieved the best results.

Project Links

GitHub Repositories

GitHub is used to store and archive all code and notebooks

- Scraper - https://github.com/DSE-capstone-sharknado/scraper
- Tf-bpr - https://github.com/DSE-capstone-sharknado/tf-bpr
- Exploratory-data-analysis - https://github.com/DSE-capstone-sharknado/exploratory-data-analysis
- Bpr-spark - https://github.com/DSE-capstone-sharknado/bpr-spark
- Bpr - https://github.com/DSE-capstone-sharknado/bpr
- Webapp - https://github.com/DSE-capstone-sharknado/webapp
- models-legacy - https://github.com/DSE-capstone-sharknado/models-legacy
- Main - https://github.com/DSE-capstone-sharknado/main
- Alex - https://github.com/DSE-capstone-sharknado/alex
- Theano-bpr - https://github.com/DSE-capstone-sharknado/theano-bpr
- AdvancedBPR - https://github.com/DSE-capstone-sharknado/AdvancedBPR

Files and Data

All final and intermediate files as well as presentations and reports are stored in a shared teamdrive ala Google Drive. The name of the teamdrive is *Sharknado*. Google Team Drives do not allow for linking directly and appear within a user's Google Drive once access is granted. Access Requests can be granted by any Sharknado team member as well as Dr. Ilkay Altantas (altintas@sdsc.edu) and Kevin Coakley (kcoakley@eng.ucsd.edu)

Files

The following lists detailed file information including filename, type, and record count. Each file also has an embedded hyperlink directly to the original file. All files were originally sourced from http://jmcauley.ucsd.edu/data/amazon/links.html.

5-core Files

category	reviews	reviews_cnt	ratings	ratings_cnt
Amazon Instant Video	reviews_Amazon_Instant_Video_5.json.gz	37,126	ratings_Amazon_Instant_Video.csv	583,933
Apps for Android	reviews_Apps_for_Android_5.json.gz	752,937	ratings_Apps_for_Android.csv	2,638,172
Automotive	reviews_Automotive_5.json.gz	20,473	ratings_Automotive.csv	1,373,768
Baby	reviews_Baby_5.json.gz	160,792	ratings_Baby.csv	915,446
Beauty	reviews_Beauty_5.json.gz	198,502	ratings_Beauty.csv	2,023,070
Books	reviews_Books_5.json.gz	8,898,041	ratings_Books.csv	22,507,155
CDs and Vinyl	reviews_CDs_and_Vinyl_5.json.gz	1,097,592	ratings_CDs_and_Vinyl.csv	3,749,004
Cell Phones and Accessories	reviews_Cell_Phones_and_Accessories_5.json.gz	194,439	ratings_Cell_Phones_and_Accessories.csv	3,447,249
Clothing, Shoes and Jewelry	reviews_Clothing_Shoes_and_Jewelry_5.json.gz	278,677	ratings_Clothing_Shoes_and_Jewelry.csv	5,748,920
Digital Music	reviews_Digital_Music_5.json.gz	64,706	ratings_Digital_Music.csv	836,006
Electronics	reviews_Electronics_5.json.gz	1,689,188	ratings_Electronics.csv	7,824,482
Grocery and Gourmet Food	reviews_Grocery_and_Gourmet_Food_5.json.gz	151,254	ratings_Grocery_and_Gourmet_Food.csv	1,297,156
Health and Personal Care	reviews_Health_and_Personal_Care_5.json.gz	346,355	ratings_Health_and_Personal_Care.csv	2,982,326
Home and Kitchen	reviews_Home_and_Kitchen_5.json.gz	551,682	ratings_Home_and_Kitchen.csv	4,253,926
Kindle Store	reviews_Kindle_Store_5.json.gz	982,619	ratings_Kindle_Store.csv	3,205,467
Movies and TV	reviews_Movies_and_TV_5.json.gz	1,697,533	ratings_Movies_and_TV.csv	4,607,047
Musical Instruments	reviews_Musical_Instruments_5.json.gz	10,261	ratings_Musical_Instruments.csv	500,176
Office Products	reviews_Office_Products_5.json.gz	53,258	ratings_Office_Products.csv	1,243,186
Patio, Lawn and Garden	reviews_Patio_Lawn_and_Garden_5.json.gz	13,272	ratings_Patio_Lawn_and_Garden.csv	993,490
Pet Supplies	reviews_Pet_Supplies_5.json.gz	157,836	ratings_Pet_Supplies.csv	1,235,316
Sports and Outdoors	reviews_Sports_and_Outdoors_5.json.gz	296,337	ratings_Sports_and_Outdoors.csv	3,268,695
Tools and Home Improvement	reviews_Tools_and_Home_Improvement_5.json.gz	134,476	ratings_Tools_and_Home_Improvement.csv	1,926,047
Toys and Games	reviews_Toys_and_Games_5.json.gz	167,597	ratings_Toys_and_Games.csv	2,252,771
Video Games	reviews_Video_Games_5.json.gz	231,780	ratings_Video_Games.csv	1,324,753

Per Category Files

category	image features	metadata	reviews	metadata_cnt	reviews_cnt
Amazon Instant Video	image_features_Amazon_Instant_Video.b	meta_Amazon_Instant_Video.json.gz	reviews_Amazon_Instant_Video.json.gz	30,648	583,933
Apps for Android	image_features_Apps_for_Android.b	meta_Apps_for_Android.json.gz	reviews_Apps_for_Android.json.gz	61,551	2,638,173
Automotive	image_features_Automotive.b	meta_Automotive.json.gz	reviews_Automotive.json.gz	331,090	1,373,768
Baby	image_features_Baby.b	meta_Baby.json.gz	reviews_Baby.json.gz	71,317	915,446
Beauty	image_features_Beauty.b	meta_Beauty.json.gz	reviews_Beauty.json.gz	259,204	2,023,070

Books	image_features_Books.b	meta_Books.json.gz	reviews_Books.json.gz	2,370,585	22,507,155
CDs and Vinyl	image_features_CDs_and_Vinyl.b	meta_CDs_and_Vinyl.json.gz	reviews_CDs_and_Vinyl.json.gz	492,799	3,749,004
Cell Phones and Accessories	image_features_Cell_Phones_and_Accessories.b	meta_Cell_Phones_and_Accessories.json.gz	reviews_Cell_Phones_and_Accessories.json.gz	346,793	3,447,249
Clothing, Shoes and Jewelry	image_features_Clothing_Shoes_and_Jewelry.b	meta_Clothing_Shoes_and_Jewelry.json.gz	reviews_Clothing_Shoes_and_Jewelry.json.gz	1,503,384	5,748,920
Digital Music	image_features_Digital_Music.b	meta_Digital_Music.json.gz	reviews_Digital_Music.json.gz	279,899	836,006
Electronics	image_features_Electronics.b	meta_Electronics.json.gz	reviews_Electronics.json.gz	498,196	7,824,482
Grocery and Gourmet Food	image_features_Grocery_and_Gourmet_Food.b	meta_Grocery_and_Gourmet_Food.json.gz	reviews_Grocery_and_Gourmet_Food.json.gz	171,760	1,297,156
Health and Personal Care	image_features_Health_and_Personal_Care.b	meta_Health_and_Personal_Care.json.gz	reviews_Health_and_Personal_Care.json.gz	263,032	2,982,326
Home and Kitchen	image_features_Home_and_Kitchen.b	meta_Home_and_Kitchen.json.gz	reviews_Home_and_Kitchen.json.gz	436,988	4,253,926
Kindle Store	image_features_Kindle_Store.b	meta_Kindle_Store.json.gz	reviews_Kindle_Store.json.gz	434,702	3,205,467
Movies and TV	image_features_Movies_and_TV.b	meta_Movies_and_TV.json.gz	reviews_Movies_and_TV.json.gz	208,321	4,607,047
Musical Instruments	image_features_Musical_Instruments.b	meta_Musical_Instruments.json.gz	reviews_Musical_Instruments.json.gz	84,901	500,176
Office Products	image_features_Office_Products.b	meta_Office_Products.json.gz	reviews_Office_Products.json.gz	134,838	1,243,186
Patio, Lawn and Garden	image_features_Patio_Lawn_and_Garden.b	meta_Patio_Lawn_and_Garden.json.gz	reviews_Patio_Lawn_and_Garden.json.gz	109,094	993,490
Pet Supplies	image_features_Pet_Supplies.b	meta_Pet_Supplies.json.gz	reviews_Pet_Supplies.json.gz	110,707	1,235,316
Sports and Outdoors	image_features_Sports_and_Outdoors.b	meta_Sports_and_Outdoors.json.gz	reviews_Sports_and_Outdoors.json.gz	532,197	3,268,695
Tools and Home Improvement	image_features_Tools_and_Home_Improvement.b	meta_Tools_and_Home_Improvement.json.gz	reviews_Tools_and_Home_Improvement.json.gz	269,120	1,926,047
Toys and Games	image_features_Toys_and_Games.b	meta_Toys_and_Games.json.gz	reviews_Toys_and_Games.json.gz	336,072	2,252,771
Video Games	image_features_Video_Games.b	meta_Video_Games.json.gz	reviews_Video_Games.json.gz	50,953	1,324,753

Complete Review Data

These files are not separated and contain all categories.

raw review data (20gb) - all 142.8 million reviews
complete.json.gz

The above file contains some duplicate reviews, mainly due to near-identical products whose reviews Amazon merges, e.g. VHS and DVD versions of the same movie. These duplicates have been removed in the files below:

user review data (18gb) - duplicate items removed (83.68 million reviews), sorted by user
user_dedup.json.gz

product review data (18gb) - duplicate items removed, sorted by product
Item_dedup.json.gz

ratings only (3.2gb) - same as above, in csv form without reviews or metadata
Item_dedup.json.gz

5-core (9.9gb) - subset of the data in which all users and items have at least 5 reviews (41.13 million reviews)
kcore_5.json.gz

Finally, the following file removes duplicates more aggressively, removing duplicates even if they are written by different users. This accounts for users with multiple accounts or plagiarized reviews. Such duplicates account for less than 1 percent of reviews, though this dataset is probably preferable for sentiment analysis type tasks:

aggressively deduplicated data (18gb) - no duplicates whatsoever (82.83 million reviews)
Format is one-review-per-line in (loose) json. See examples below for further help reading the data.
aggressive_dedup.json.gz

Tool setup

We used a standard set of tools for development.

Tools

- Python 2.7
- TensorFlow 1.1
- Anaconda 4.3 - Analytics Python Distribution
- Jupyter Notebooks - for data analysis
- D3 v4 - for d3 based visualizations
- Github Teams - provided by MAS - code management and wiki
- Slack - team communication, and collaboration
- Google Drive - File Sharing
- Google Apps - For spreadsheets, documents and presentations

A project site was created for the team on GitHub: https://github.com/DSE-capstone-sharknado. All our code repositories are stored here. We utilize the wiki functionality on here to help document ideas and code.

DSE MAS Knowledge

The capstone is a great opportunity to showcase all the knowledge and skills we have acquired on our journey through the program.

MAS Skills Used:

- Python programming
- Jupyter Notebooks
- PySpark
- Statistical Methods:
  - Bayesian probabilities
  - Stochastic Gradient Descent
  - Matrix Multiplication
- HTML/CSS/XQuery/XPath
- Git
- JSON

Python code is used heavily throughout the project. It is used for building everything from our ML models to data cleaning and manipulation. Jupyter Notebooks are used to as our IDE for writing python. Our current model utilizes several statistical techniques that were covered in one of the statistics classes. The QA is built on standard technologies HTML/CSS, skills that were covered in DSE241. Some of the more basic skills such as using GIT and JSON were learned early on in the program.