

**Recommendation system**



**TEAM MEMBERS:**

M. Bhargav Chandra

P. Sumanth

P. Revanth

M. A. Jabbar

M. Sai Priyatham

**Recommendation System**

**Contents**

[**Abstract:** 2](#_Toc9682495)

[**Introduction:** 2](#_Toc9682496)

[**Objectives of Research:** 3](#_Toc9682497)

[**Problem Statement:** 3](#_Toc9682498)

[**Review of Literature:** 4](#_Toc9682499)

[**Data Collection:** 5](#_Toc9682500)

[Step 1: Find the URL that we need to scrape 5](#_Toc9682501)

[Step 2: Inspecting the Page 5](#_Toc9682502)

[Step 3: Finding the data we need 6](#_Toc9682503)

[Step 4: Write the code 6](#_Toc9682504)

[Step 5: Store the data in a required format 7](#_Toc9682505)

[**Methodology:** 7](#_Toc9682506)

[**Exploratory Data Analysis:** 8](#_Toc9682507)

[**Data Modelling:** 8](#_Toc9682508)

[Gibbs Sampling: 8](#_Toc9682509)

[Restricted Boltzmann Machines: 9](#_Toc9682510)

[Contrastive Divergence: 11](#_Toc9682511)

[**Findings and Suggestions:** 11](#_Toc9682512)

[**Conclusion:** 12](#_Toc9682513)

# **Abstract:**

Most accurate recommender systems are black-box models, hiding the reasoning behind their recommendations. Yet explanations have been shown to increase the user’s trust in the system in addition to providing other benefits such as scrutability, meaning the ability to verify the validity of recommendations. This gap between accuracy and transparency or explainability has generated an interest in automated explanation generation methods. Restricted Boltzmann Machines (RBM) are accurate models for CF that also lack interpretability. In this paper, we focus on RBM based collaborative filtering recommendations, and further assume the absence of any additional data source, such as item content or user attributes. We thus propose a new Explainable RBM technique that computes the top-n recommendation list from items that are explainable. Experimental results show that our method is effective in generating accurate and explainable recommendations.

# **Introduction:**

Explanations for recommendations can have several benefits, such as: helping the user make a good decision (effectiveness), helping the user make a faster decision (efficiency), and revealing the reasoning behind the system’s recommendation (transparency) (Tintarev & Masthoff, 2011; Zanker, 2012). As a result, users are more likely to follow the recommendation and use the system in better ways (Tintarev & Masthoff, 2007; Herlocker et al., 2000). For instance, the Netflix recommender system justifies its movie suggestions by listing similar movies, obtained from the user’s social network. Amazon’s recommender system shows similar items to the ones that the user (or other similar users) have bought or viewed, when recommending a new item using neighbourhood based Collaborative Filtering (CF).

CF approaches provide recommendations to users based on their collective recorded interests on items, typically relying on the similarity between users or items, giving rise to neighbourhood-based CF approaches, which can be user-based or item-based. Neighbourhood-based CF methods are white-box approaches that can be explained based on the ratings of similar users or items.

Most accurate recommender systems are model based methods that are black-boxes. Among model-based approaches are Restricted Boltzmann Machines (RBM) (Hinton, 2010) that can assign a low dimensional set of features to items in a latent space. The newly obtained set of features capture the user’s interests and different items groups; however, it is very difficult to interpret these automatically learned features. Therefore, justification of the recommendation or the reasoning behind the recommended item in these models is not clear.

RBM approaches have recently proved to be powerful for designing deep learning techniques to learn and predict patterns in large datasets because they can provide very accurate results (Hinton & Salakhutdinov, 2006). However, they suffer from the lack of interpretation of the results, especially for recommender systems.

Lack of explanations can result in users not trusting the suggestions made by the recommender system. Therefore, the only way for the user to assess the quality of a recommendation is by following it. This, however, is contrary to one of the goals of a recommendation system, which is reducing the time that users spend on exploring items. It would be very desirable and beneficial to design recommender systems that can give accurate suggestions, which, at the same time, facilitate conveying the reasoning behind the recommendations to the user. A main challenge in creating a recommender system is to choose an interpretable technique with moderate prediction accuracy or a more accurate technique, such RBM, which does not give explainable recommendations.

# **Objectives of Research:**

To provide an effective way of creating a personalized shopping experience for each customer which helps Amazon increase average order value and the amount of revenue generated from each customer.

# **Problem Statement:**

Our research question is: can we design an RBM model for a CF recommender engine that suggests items that are explainable, while recommendations remain accurate? Our current scope is limited to CF recommendations where no additional source of data is used in explanations, and where explanations for recommended items can be generated from the ratings given to these items, by the active user’s neighbours only (user-based neighbour style explanation)

# **Review of Literature:**

Over the past decade, e-commerce has grown at a very healthy rate of almost 18 percent a year [3]. As of 2013, it already accounted for an estimated 8 percent of total retail sales. And the larger share of that basket belongs undoubtedly to Amazon, one of the largest online retailers on the planet. You can imagine Amazon’s success is not some random anomaly. They are one of the most analytical companies around and nothing emphasizes more their data driven approach than their recommendation algorithms.

Amazon employs many forms of recommendations to engage the user, increasing average value order and inviting them to acquire the latest (and certainly more profitable) items [4]. Their recommendation system is based on a number of simple elements: what a user has bought in the past, which items they have in their virtual shopping cart, the items they’ve rated and liked, and what other customers like them have viewed and purchased [5]. The result is a highly personalized shopping browser: a gadget enthusiast may find Amazon pages heavy on device suggestions, while a driving lover could see those same pages offering up car products.

There has been a lot of work done in this field. For example, one very popular algorithm is Collaborative Filtering. One type of collaborative filtering is user-based collaborative filtering, which starts by finding a set of customers who have purchased and rated similar items with the target users purchasing history. The algorithm aggregates items from these similar customers, and uses the ratings from other similar users to predict the ratings from this user. Another type of collaborative filtering is item-based collaborative filtering, which was first brought up by Amazon [4] and focuses on finding similar items instead of similar customers. For each of the users purchased and rated items, the algorithm attempts to find similar items. It then aggregates these similar items and recommends them.

There are also other algorithms that try to exploit graph structures to predict links or ratings. Random walks algorithms [2] could be used in predicting links in complex graphs in a very efficient manner. And also, if we model the user and product graph as a bipartite graph, then it is also feasible to use Bipartite Projection algorithm [5] to calculate the relevance between two customers. So, the predicted rating is essentially based on the other relevant customers’ ratings. In later sections of this paper, we will introduce three models and algorithms which are derived from the prior work mentioned above with application-specific improvements.

# **Data Collection:**

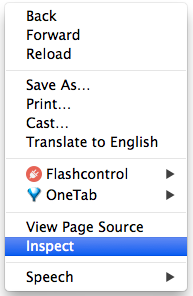
For Data Collection we performed web scrapping, the first thing that we need to do is to figure out where we can locate the links to the files we want to download inside the multiple levels of HTML tags. There is a lot of code on a website page and we want to find the relevant pieces of code that contains our data. It is important to understand the basics of HTML in order to successfully web scrape.

## **Step 1: Find the URL that we need to scrape**

For this example, we are going scrape **Amazon** website to extract the Price, Name, and Rating of Laptops. The URL for this page is <https://www.amazon.com/s?k=shirts&ref=nb_sb_noss_1>

## **Step 2: Inspecting the Page**

On the website, right click and click on “Inspect”. This allows us to see the raw code behind the site.



Once we’ve clicked on “Inspect”, we should see this console pop up.

We notice that on the top left of the console, there is an arrow symbol.

https://cdn-images-1.medium.com/max/800/1*OBTSehekWVX6rSXUaibd1A.png

We click on this arrow and then click on an area of the site itself, the code for that particular item will be highlighted in the console. We’ve clicked on the very first product in search results and the console has highlighted in blue the link to that particular item. Now that we’ve identified the location of the links.

## Step 3: Finding the data we need

We extract the Price, Name, and Rating which is nested in the “div” tag respectively.

## Step 4: Write the code

First, we import all the necessary libraries:

From selenium import webdriver

From BeautifulSoup import BeautifulSoup

Import pandas as pd

To configure webdriver to use Chrome browser, we have to set the path to chromedriver

Now we open the URL, it’s time to extract the data from the website. As mentioned earlier, the data we want to extract is nested in <div> tags. So, we find the div tags with those respective class-names, extract the data and store the data in a variable.

## Step 5: Store the data in a required format

After extracting the data, we store it in a format. This format varies depending on our requirement. For this example, we will store the extracted data in a CSV (Comma Separated Value) format. To do this, I will add the following lines to my code:

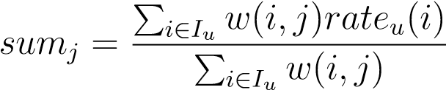
# **Methodology:**

The first recommendation system we build is inspired by Amazons item-based collaborative filtering [4]. In Amazons algorithm, they represent each item with a vector showing who bought/reviewed the item. Similarity between these two products is defined by the cosine of the two vectors. After calculating similarity between all product pairs, we will have an item-item matrix showing the similarity between the items. Finally, the similarities can provide a good reference on some of the other products that a customer would buy.

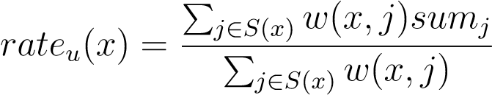
The original algorithm is used to predict the next product that a customer would buy. To adopt it in our application, which is to predict the rating given by some customer for some product, we create an algorithm that make use of the item-item similarity. First lets define some terms that we will use later. Let w(i, j) be the similarity between item i and item j; Iu is the set of products customer reviewed, excluding the one we are going to predict with; rateu(i) is the rate for product i given by customer u. S(i) is the most similar items with item i, including i itself, according to the item-item similarities. Finally, let x be the item that we are trying to predict for customer u. We calculate a weighted sum for each j ∈ S(x). For each item i that customer u have a rating, we give them weights using the similarity between i and j

FXVWRPHUV

SURGXFWV



This weighted sum basically indicates that, for an item *j* which is similar to *x*, what would the rating be for *j* given all the ratings for *i* ∈ *Iu*. Then, we take another weighted sum over each *j*, where the weights are given by the similarity between *j* and *x*. The rating for item *x* is then given by:



## **Exploratory Data Analysis:**

## **Data Modelling:**

### Gibbs Sampling:

Gibbs sampling is a simple MCMC algorithm for producing samples from the joint probability distribution of multiple random variables. The basic idea is to construct a Markov chain by updating each variable based on its conditional distribution given the state of the others. In the following, we will describe this procedure by explaining how Gibbs sampling can be used to produce samples (approximately) from the Gibbs distribution of an MRF.

We consider an MRF X = (*X*1*,...,XN*) w.r.t. an undirected graph *G* = (*V,E*), where *V* = {1*,...,N*} for the sake of clearness of notation. The random variables *Xi*, *i* ∈ *V* take values in a finite set *Λ* and  is the joint probability distribution of X. Furthermore, if we assume that the MRF changes its state over time, we can consider *X* = {X(*k*) |*k* ∈ N0} as a Markov chain taking values in *Ω* = *ΛN*. Then X) describes the state of the MRF at time *k* ≥ 0.

Between two successive points in time, the new state of the chain is produced by the following procedure. First, a variable *Xi*, *i* ∈ *V* is randomly picked with a probability *q*(*i*) given by a strictly positive probability distribution *q* on *V* . Then, the new state for *Xi* is sampled based on its conditional probability distribution given the state (*xv*)*v*∈*V* \*i* of all other variables (*Xv*)*v*∈*V* \*i*. We have

) because of the local Markov property of MRFs.

The Transition probability *pxy* for two states x*,*y of the MRF X with x 6= y is

(so that ∀*v* ∈ *V* with *v* 6= *i*: *xv* = *yv* *P*xy = (15)

0*,* else *.*

And the probability, that the state of the MRF x stays the same, is.

### Restricted Boltzmann Machines:

An RBM is an MRF associated with a bipartite undirected graph as shown in Fig. 5. It consists of *m* visible units V = (*V*1*,...,Vm*) representing the observable data, and *n* hidden units H = (*H*1*,...,Hn*) to capture the dependencies between the observed variables. In binary RBMs, our focus in this tutorial, the random variables (V *,*H) take values (v*,*h) ∈ {0*,*1}*m*+*n* and the joint probability distribution under the model is given by the Gibbs distribution *p*(v*,*h) = *Z*1 *e*−*E*(v*,*h) with the energy function

*n m m n*

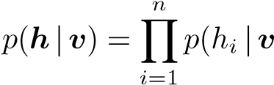
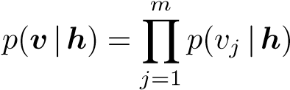
*E*(v*,*h) = −XX*wijhivj* − X*bjvj* − X*cihi .* (18)

*i*=1 *j*=1 *j*=1 *i*=1

For all *i* ∈ {1*,...,n*} and *j* ∈ {1*,...,m*}, *wij* is a real valued weight associated with the edge between the units *Vj* and *Hi*, and *bj* and *ci* are real valued bias terms associated with the *j*th visible and the *i*th hidden variable, respectively.

**Fig.** The network graph of an RBM with *n* hidden and *m* visible units.

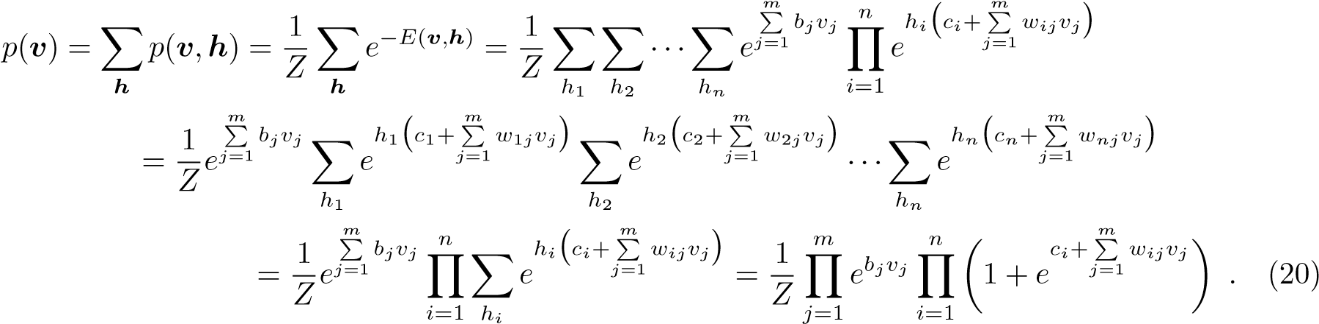
The graph of an RBM has connections only between the layer of hidden and the layer of visible variables, but not between two variables of the same layer. In terms of probability, this means that the hidden variables are independent given the state of the visible variables and vice versa:

) and  *.*

Thus, due to the absence of connections between hidden variables, the conditional distributions *p*(h|v) and *p*(v |h) factorize nicely, and simple expressions for the factors will be given in Sec. 4.1.

The conditional independence between the variables in the same layer makes Gibbs sampling especially easy: instead of sampling new values for all variables subsequently, the states of all variables in one layer can be sampled jointly. Thus, Gibbs sampling can be performed in just two steps: sampling a new state h for the hidden neurons based on *p*(h|v) and sampling a state v for the visible layer based on *p*(v |h). This is also referred to as *block Gibbs sampling*.

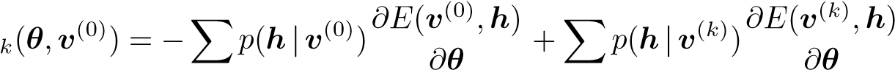
Now, how does the RBM distribution over V (e.g., the space of images) look like? The marginal distribution (7) of the visible variables becomes



### Contrastive Divergence:

Obtaining unbiased estimates of the log-likelihood gradient using MCMC methods typically requires many sampling steps. However, it has been shown that estimates obtained after running the chain for just a few steps can be sufficient for model training. This leads to *contrastive divergence* (CD) learning, which has become a standard way to train RBMs.

The idea of *k*-step contrastive divergence learning (CD-*k*) is quite simple: instead of approximating the second term in the log-likelihood gradient by a sample from the RBM-distribution (which would require running a Markov chain until the stationary distribution is reached), a Gibbs chain is run for only *k* steps (and usually *k* = 1). The Gibbs chain is initialized with a training example v(0) of the training set and yields the sample v(*k*) after *k* steps. Each step *t* consists of sampling h(*t*) from *p*(h|v(*t*)) and subsequently sampling v(*t*+1) from *p*(v |h(*t*)). The gradient, w.r.t. θ of the log-likelihood for one training pattern v(0) is then approximated by

CD

In *batch learning*, the complete training data set *S* is used to compute or approximate the gradient in every step. However, it can be more efficient to consider only a subset *S*′ ⊂ *S* in every iteration, which reduces the computational burden between parameter updates. The subset *S*′ is called a *mini-batch.* If in every step only a single element of the training set is used to estimate the gradient, the process is often referred to as *online learning.*

# **Findings and Suggestions:**

We built a testing framework before implementing all these algorithms. The test framework allows us to to cross-validations in our experiments. To be more specific, we divide the customers in our dataset randomly into groups of equal size. Then we pick one of the group, and randomly remove one rating from each customer in the group. The removed ratings become our test dataset. Each data in the test dataset is a triplet of product, customer, and rating. The product and customer are sent to our recommendation system for predicting, and the rating is our ground truth for testing.

# **Conclusion:**

We presented an explainable RBM approach for CF recommendations that achieves both accuracy and interpretability by learning an RBM network that tries to estimate accurate user ratings while also taking into account the explainability of an item to a user. Both rating prediction and explainability are integrated within one learning goal allowing to learn a network model that prioritizes the recommendation of items that are explainable