Fusion Serendipitous Recommendation System: An Adjustable Balance Between Relevance and Surprise

Yijie Ding Xunzhi He Jiaheng Dai
Department of Computer Science and Engineering
University of California San Diego, CA 92093
{yid016,xuh011,j5dai}@ucsd.edu

Project Code

Abstract

In various scenarios, online recommendation systems are increasingly expected not only to exploit users' interests but also to facilitate exploration. Therefore, it becomes crucial to prioritize not just accuracy but a balanced serendipity, which we define as relevance, diversity, and unexpectedness. Current methods often construct serendipity vectors based on heuristics and alter the normal recommendation outcomes. which offers explainability but lacks alignment with genuine customer feedback. To bridge this gap, we proposed the Serendipitous Recommendation System that employs 1. a Sequential Category-Prediction model implemented using a Long Short-Term Memory (LSTM) to capture users' short-term demand, and 2. a Latent Factor Model (LMF) trained on real customer review sentiments for estimating user unexpectedness based on their long term preference. Recommendations are generated by aggregating the relevance and unexpectedness score using an adjustable weight λ. We then conduct experiments to compare the SRS with a Bayesian Personal Ranking (BPR) baseline using the serendipitous metrics and empirically discuss the effect of λ on recommendation results. Our results demonstrate both SRS's efficacy and explainability in enhancing recommendation serendipity, offering a novel approach to consumer product suggestions.

1 Introduction

1.1 Background

In today's consumer culture, there's a growing fascination with discovering the unique and unexpected. More and more, people are seeking experiences and products that stand out from the mainstream. They're on a quest to uncover hidden treasures, indie brands, and uncharted territories, moving away from the well-trodden path of high ratings and popularity. This shift poses a challenge to traditional recommender systems, which typically play it safe by suggesting items that are already well-known and highly rated. Here are two examples of traditional recommender systems:

- **Amazon Personalize**[1] Amazon Personalize enables developers to quickly build and deploy sophisticated recommendation systems using machine learning. This service can be customized based on individual needs, allowing for the delivery of appropriate customer experiences at the right time and place.
- **Netflix's Recommendation**[2] Netflix's recommender system personalizes viewing experiences by analyzing user interactions (like viewing history and ratings), similar preferences of other members, detailed information about titles (genre, actors, etc.), and the viewing context, including the time of day, devices used, and viewing duration.

While the magic lies in the unveiling of the obscure – those rare finds and secret delights that traditional algorithms often overlook but which resonate with the individuality of the adventurous shopper. This new craving for the unconventional is reshaping the way recommendations are curated, emphasizing the joy of personal resonance over collective approval. Here is a real life example:

- **SSENSE**[3] The article by Alexis Anzieu in SSENSE-TECH emphasizes the need to integrate serendipity into recommendation algorithms. It argues that current systems often create echo chambers by only suggesting familiar content. Incorporating serendipity would allow these algorithms to offer more diverse and unexpected recommendations, enhancing creativity and discovery.

1.2 Task description

To address this, we aim to develop a serendipitous recommender system between unexpectedness and relevance, which would offer unexpected and delightful recommendations to users. However, according to the history it is vague to define what is serendipity. After reading loads of related works, we think that a serendipitous recommendation is characterized by being **relevant**, **diversity**, and **unexpectedness**.

1.3 Dataset

In this study we use the dataset - Amazon Product Reviews [4]. This dataset contains product reviews and metadata from Amazon, including 142.8 million reviews spanning May 1996 - July 2014. Since we need to do the sentimental task, so this dataset really meets our demand.

We use two subsets of this dataset:

- **5-core** Subset of the data in which all users and items have at least 5 reviews, which contains 41.13 million reviews. It contains:
  - 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 for 9.4 million products. It contains:
  - 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

1.4 Exploratory Data Analysis

The data cleaning and processing for the 5-core dataset, containing 41,130,000 reviews, involved generating 9,000,000 random indices to select reviews while ensuring computational feasibility.

Due to memory concerns, we selected 9,000,000 records from the 5-core dataset and filtered the data frame to only retain users with more than 15 items in purchase history, as cold-start problems are

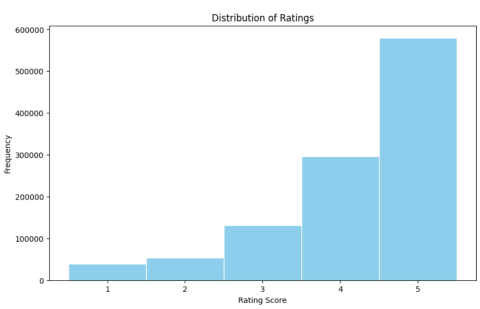

Figure 1: Rating Score distribution

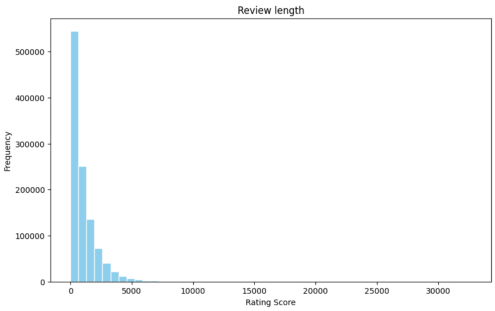

Figure 2: Review length distribution

excluded from the focus of this paper. This process yielded a dataset with 314,565 unique reviewers. Finally, we enriched the dataset by merging it with metadata containing item titles. An overview of this cleaned dataset is presented in fig. 4. Key findings from the analysis are as follows:

- **Rating Distribution**: A bar-plot (fig. 1) showed a positive skew in Amazon item ratings, with 5-star ratings being predominant. This indicates a general customer satisfaction or a tendency of satisfied customers to post reviews.
- **Review Length Distribution**: Reviews mostly fall below 5000 words, with a right skew (fig. 2). This length is typical for product reviews and is relevant for sentiment analysis.
- **Review Time Distribution**: Reviews are mostly concentrated around 2013-2014, indicating an increase in reviewing activity in recent years (fig. 3).

These insights inform our analysis and model development strategy.

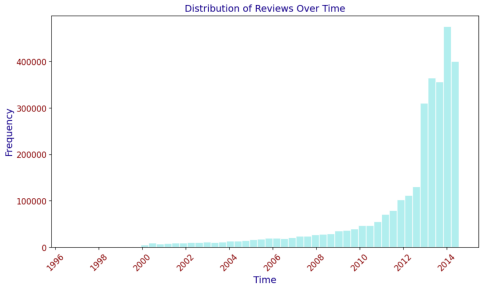

Figure 3: Review time distribution

#	Column	Dtype
0	reviewerID	string
1	asin	string
2	reviewerName	string
3	reviewText	string
4	overall	float64
5	summary	string
6	unixReviewTime	float64
7	imUrl	string
8	categories	set[string]
9	title	string
11	review_length	int64

Figure 4: DataFrame Information

After recognizing that the distribution of features, we observed the top 10 items and reviewers ranked by the number of reviews they've received to check for extreme outliers. fig. 5 fig. 6

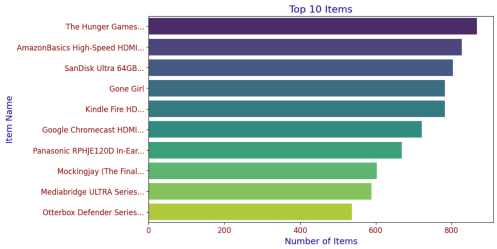

Figure 5: Top 10 Items by number of reviews

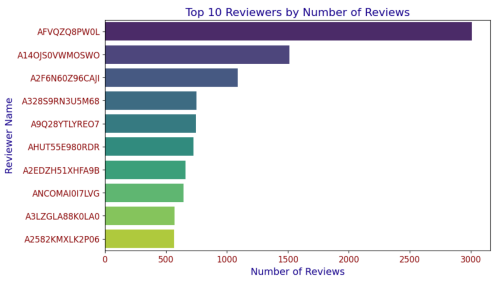

Figure 6: Top 10 Users by number of reviews

2 Related Work

The state-of-the-art serendipity recommendation is hard to define. Since different researchers have different definition of serendipity, we are hard to say which one is better. We found a couple of studies have been done well.

- **Directional and Explainable Serendipity Recommendation** [5]: This study introduces a model named DESR (Directional and Explainable Serendipity Recommendation), which offers a novel approach for generating serendipitous recommendations. It consists of four key components:

  Long-term Preferences Extraction: This part uses an unsupervised method based on Gaussian Mixture Models (GMM) to infer users' long-term preferences, resulting in 'preCaps'.

  Short-term Demands Capture: It employs a capsule network to capture users' short-term demands ('deCaps'), chosen for its effectiveness in representation and explainability.

  Recommendations Generation: It combines 'preCaps' and 'deCaps' to calculate a serendipity vector, which is then used to generate recommendations ('reCaps').

  Back-routing for Explanations: It provide explanations for the recommendations, a back-routing strategy traces the path from 'reCaps' back to 'itemCaps'.

- **HAES** [6]: In their research, the authors introduced HAES, standing for Hybrid Approach with Elastic Serendipity, specifically tailored for movie recommendation. This concept recognizes that individuals vary in their receptiveness to different types of surprises. It acknowledges that what constitutes a surprise can differ from one person to another. The core idea of HAES is to leverage users' historical responses to surprises to predict the kind of surprise value that might be appealing or acceptable to them. Essentially, HAES aims to personalize the recommendation process by understanding and predicting each user's unique threshold or elasticity for surprising elements in movies.

Our study's conclusions align closely with those from the HAES research, although our definition of serendipity differs. Both studies focus on enhancing recommendation systems by integrating unexpectedness without compromising relevance. In the HAES study, their analysis on the MovieLens-latest-small dataset reveals that the random-based (RAND) and novelty-based (NOV) methods outperform the accuracy-based (ACC) approach in terms of genre accuracy. This outcome highlights the potential of serendipity recommendation systems to boost performance when applied appropriately. Similarly, in our model (detailed in the following section), we demonstrate this concept through graphical representations, showing how effectively incorporating serendipity can enhance the overall performance of recommendation systems.

3 Methods

3.1 System Setting and Problem Definition

We consider the scenario in a comprehensive consumer goods recommendation system (e.g. the Amazon Retail Platform), where there are users (U), items (I) across different domains, and the user-item interactions (T) such as review text, and the review time. One single prediction task can

be described as, given a user $u \in U$, we want to recommend k items $I_{rec} = (i_1, i_2, ..., i_k) \subset I$ to optimize the Serendipity metrics defined below.

3.2 Serendipity Metric

The Serendipity Metric is a function of I_{rec}, I_u, and i_{target} to a scalar, where I_{rec} is the k items recommended, I_u is all the reviewed items by user u, and i_{target} is the ground-truth next item that the user purchase. In section 1.2, we define serendipitous by being relevant, diverse, and unexpected. Consequently, we purposed the following definitions of Relevance @ k, Diversity @ k, and Unexpectedness @ k.

First, for every item i in the set of all items I, there is an associated set of categories C for that item (e.g. for the book Harry Potter, C = {Book, Thriller, Fantasy, Fiction}), only keep those from the top 1000 most frequent categories. Then, we can convert C to an embedding vector $E \in R^{1000}$ where E_j = 1 if the item has the top j^{th} category else 0.

Relevance @ k: In a realistic scenario, it's nearly impossible for the recommender to accurately predict the ground-truth next item. Therefore, it's sufficient for our recommendations to match the categories of the ground-truth, i_{target}. The formula for relevance is given by the average embedding similarity between the k recommended items and the ground-truth item.

$$R = \frac{1}{k} \sum_{i=1}^{k} \text{sim}(E_{\text{target}}, E_i)$$

Where sim is the cosine similarity function between two vectors, defined as:

$$\text{sim}(A, B) = \frac{A \cdot B}{\|A\| \cdot \|B\|}$$

Diversity @ k: The diversity of our recommendations is given by the total number of categories present in recommended items, namely:

$$D = \left| \bigcup_{i=1}^{k} C_i \right|$$

Unexpectedness @ k: our measure of unexpectedness is obtained through comparing the categories of each recommended items and the categories of each historical purchase, summing the results, and dividing by a constant:

$$U = \frac{1}{k} \sum_{i=1}^{k} \sum_{j \in I_u} \text{sim}(E_i, E_j)$$

I_u is the set of historically purchased items by the user of interest.

Consequently, we define **Serendipity @ k** to be the joint sum of the 3 metrics above:

$$S = R + D + U$$

3.3 Base Model

Bayesian Personal Ranking[7] is one of the most widely adopted recommendation algorithm nowadays, and has demonstrated great recommendation performance in the system setting of interest. It serves as the baseline model to evaluate our proposed Serendipitous model. The comparison will focus on two key aspects: the Serendipity Metrics defined in section 3.2, which quantify the novelty and unexpectedness of recommendations, and the overall recommendation quality from a human perspective.

The objective function of the model, which we aim to minimize is defined as the following:

$$\text{obj} = \sum_{u,i,j \in T} -\ln\left(\sigma\left(\gamma_u \gamma_i - \gamma_u \gamma_j\right)\right)$$

For each user u, we construct tuples (u, i, j), where i is an item that the user had consumed, and j is an item that the user had not consumed. σ is a weighting factor, γ_u,γ_i, γ_j are vectors of latent factors respectively associated with u, i, and j.

3.4 Serendipitous Recommendation System (SRS)

We purpose the SRS to generate serendipitous recommendations, whose framework is illustrated in fig. 7.

3.4.1 Framework Overview

Inspired by Li et al.'s work on Explainable Serendipity Recommendation[5], we believe that users make purchase decisions based on both their short-term demand, and their long-term preferences. Therefore, to generate recommendations that are both relevant and unexpected to them (i.e serendipitous), our system consists of a Sequential Prediction model that generates relevance scores for all items, and a latent unexpectedness estimator that generates unexpectedness scores for all items. Finally we integrate the 2 scores with a balancing factor, and output the top k items.

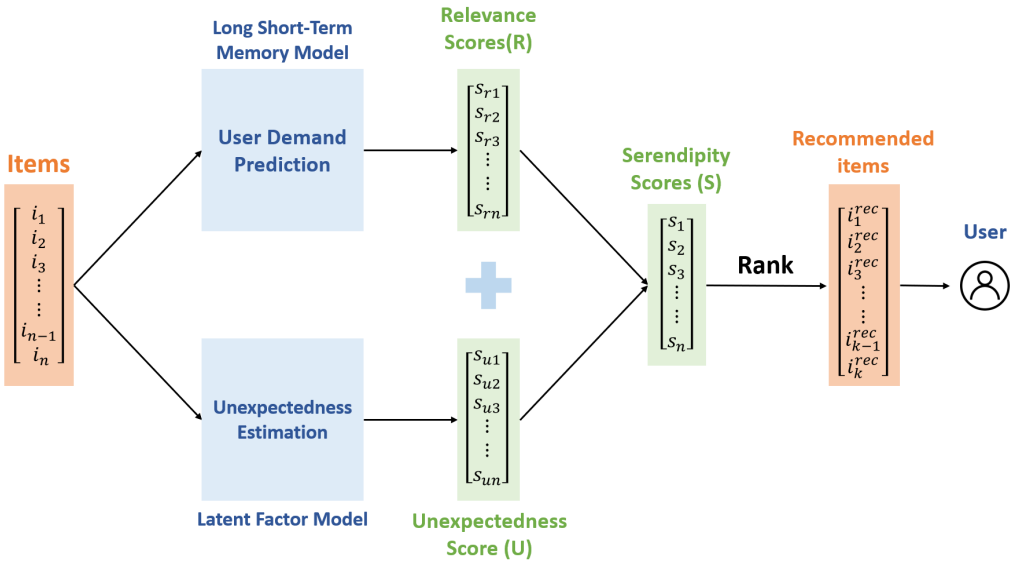

Figure 7: The SRS framework

3.4.2 Model details

User demand prediction We capture users' short term need by employing Sequence Modeling of the item categories. This is because users have shared purchase patterns, as illustrated by fig. 8. For example, we expect one to be likely to purchase a trash bag after purchasing a trash can.

To achieve this, we first convert item categories into embedding vector $e \in R^{1000}$ since we only wish to keep the most frequent 1000 item categories. e_j = 1 if the item has the j^{th} category, 0 otherwise.

We then train a Long-Short Term Memory Model[8] (LSTM) on N samples to predict e_{t+1} given (e_1, ... , e_t), the embedding vector sequence of the user's purchase history. The problem in essence is a binary classification for each entry in the predicted embedding, so we adopt the joint binary cross entropy as the loss function:

$$Loss(\hat{Y}, Y) = \sum_{L} \sum_{p} -y \cdot log(\hat{y}) - (1-y) \cdot log(1-\hat{y})$$

LSTM is designed to capture long-range dependencies in sequential data. It consists of three main gates, the input gate (i_t), the forget gate (f_t), and the output gate (o_t), as well as a memory cell (c_t), which is updated over time using the following equations:

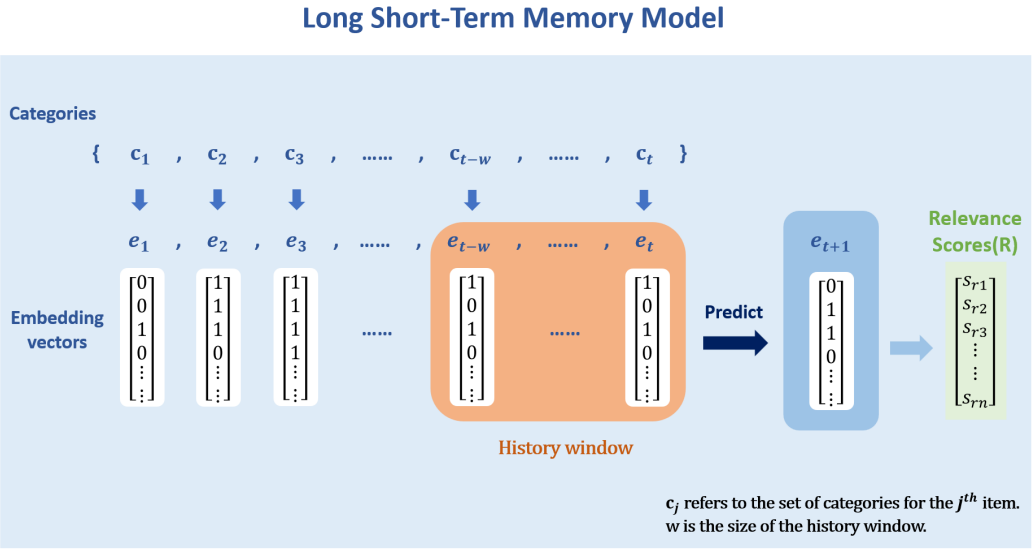

Figure 8: Long short-term memory model

$$
\begin{aligned}
i_t &= \sigma(W_{ix}x_t + W_{ih}h_{t-1} + b_i) \\
f_t &= \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f) \\
o_t &= \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_{cx}x_t + W_{ch}h_{t-1} + b_c) \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

Here, x_t represents the input at time step t, h_t is the hidden state, σ denotes the sigmoid activation function, $\odot$ denotes element-wise multiplication, and W and b are the weight matrices and bias vectors, respectively, for each gate.

Next, with the predicted embedding e_{t+1}, we generate the relevance scores of all items based on their similarity of category embeddings to e_{t+1}. Let n be the total number of items in the system, $E \in R^{m \times 1000}$ be the embedding matrix of all items.

$$R = Ee_{t+1}$$

where $V \in R^m$ is the relevance score vector.

Unexpectedness estimation We estimate the users' unexpectedness towards items using latent representations, assuming that users' long term preferences are nearly static and won't be significantly influenced by temporal information. The flow is illustrated in fig. 9.

To extract unexpectedness from the interactions, we conduct sentiment analysis of the review text using distilroberta-base model from Hugging Face and extract the output logits within the 'surprise' class. We choose the distilroberta-base model due to its comparable sentiment classification performance to the roberta-large model, and its distilled version ensures faster inference time. Then, we train a Latent Factor Model using a training set of tuples (u, i, s), where u is the user, i is the item, and s is the surprise score of that review text.

Let $f(i, j)$ be the unexpectedness score prediction, α represents a global average across users and items, β_u measures how much a user rates above the mean, β_i measures how much an item is rated above the mean, $\gamma_u, \gamma_i \in R^k$ are latent factors.

$$f(i, j) = \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i$$

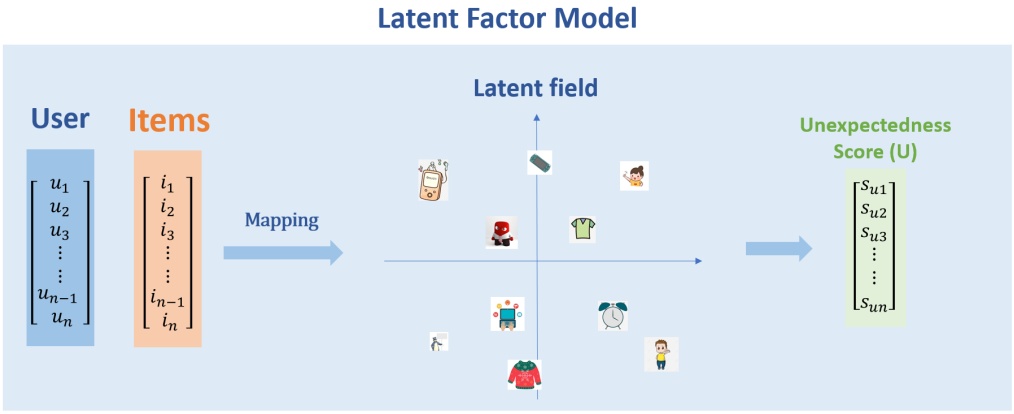

Figure 9: Latent Factor Model

The objective function which we aim to minimize is the sum of squared errors across user-item pairs, added with a regularization factor. It can be seen as a loss function that evaluates the performance of model predictions against actual surprise scores.

$$reg = \lambda \left[\sum_u \beta_u^2 + \sum_i \beta_i^2 + \|\gamma_u\|_2^2 + \|\gamma_i\|_2^2 \right]$$

$$\text{objective} = \sum_{u,i} \left(f(i,j) - R_{i,j} \right)^2 + reg$$

Using an unexpectedness regressor model, we can generate unexpectedness scores for all items. Let $\Gamma_I \in R^{m \times k}$ be latent factor matrix for all items. Let $B_I \in R^m$ be latent bias matrix for all items.

$$U = \alpha + B_I + \Gamma_I \gamma_u$$

Recommendation generation Finally, we aggregate the relevance score and unexpectedness score using a weighted sum S:

$$S = R + \lambda \cdot U$$

Then we rank all items based on the aggregated score, and output the top k items. Here, λ is an user adjustable parameter to control the balance between the relevance and unexpectedness of the recommendation results.

4 Experiments

4.1 Base Model

We implement Bayesian Personal Ranking through the Implicit library. Initially, we create a sparse matrix to represent user-item interactions, and then fit a BPR model on this matrix with latent factor dimension of 5. We evaluate different latent factor dimension, and observed that the model with dimension of 5 generates most reasonable recommendations from the human perspectives. For example, the latest 10 purchases by a typical user ("A37E6RW5BUX4U0") are shown in fig. 10, and the recommended items in fig. 11. The baseline model's emphasis on relevance is evident, since both historical purchases and recommended items are predominantly comprised of books.

Figure 10: Last 10 items purchased

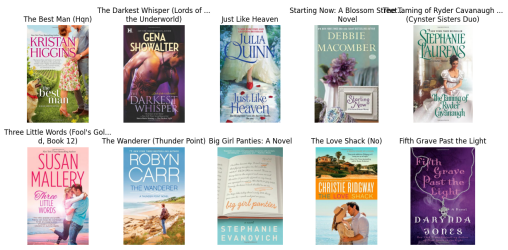

Figure 11: 10 recommended items

4.2 Serendipitous Model

LSTM Training For LSTM training, we apply a 0.99 - 0.01 train-test split against users (i.e. time sequences for 99% of the users would be included in the training set, and whole sequence for the chosen users will be used.), and a batch size of 128. Our LSTM architecture has a hidden size of 512, and 2 hidden layers. Using SGD optimizer with learning rate 0.01 and gradient clipping at 5 to prevent overfitting, we train the LSTM model for 100 epochs on Google Colab using a V100 GPU. The final training and validation losses are 79.8356 and 78.4033 respectively, and the train and valid loss graph is shown.

Notably, we experiment with window sizes of 3, 5, 7, 10 for the sequential input into LSTM, and observe that relatively small window size of 5 work the best. We speculate that larger window size introduce information from purchase history farther ago, which has less predictive power to the next item's category, introducing more noise into model's judgement. On the other hand, window size of 3 exhibits slightly lower performance due to absence of some recent useful information.

We also experiment to incorporate the time interval between each category vector as a feature for LSTM, aiming to train the model to assign weights to previous purchase according to time distance when making predictions. However, this approach does not yield a significant improvement in the model's performance.

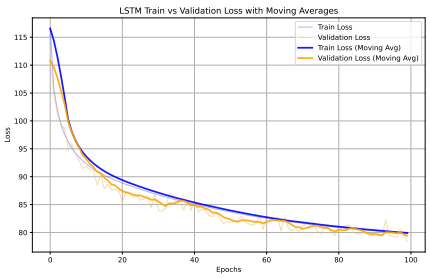

Figure 12: Long short-term memory Loss

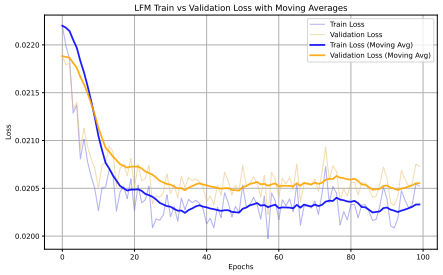

Figure 13: Learning Curve for Training and Validation Sets

LFM Training We implement Latent Factor Model with the Tensorflow library, using the Adam optimizer with a momentum decaying rate of 0.01, a latent dimension of 100 and a learning rate of $1e-5$. We train the model for 100 iterations on Google Colab using a V100 GPU. In each iteration, we construct a set of 150000 random samples from the training set, introducing some noise to the model training and enhance its robustness. The final training and validation losses are 0.02052, and 0.02073 respectively.

During training, we discover that the generated surprise score labels are noisy, making the model hard to converge, even given a small learning rate. Through experimentation, we surprisingly find out that a large dimension size for latent factors can effectively mitigate the problem. We hypothesize that a dimension size of around 100 forms an local minimum besides the initial state, suggesting it might be most aligned with the natural way of representing the complex unexpectedness compatibility factors.

Figure 14: Learning Curve for Training and Validation Sets

Figure 15: 10 recommended items by the base model

Generation Result In section 4.1, we saw that the base model is able to make coherent recommendations based on users purchase history. However, it falls short when the user's purchase history is miscellaneous, the prediction get uncertain because it cannot put on more weight on most recent purchases. Moreover, the genre of the recommended items is uniform, and lack novelty and surprise. For example, as shown in fig. 15, the model is unable to distinguish that the user bought books long time ago, and their interest shifts primarily to household gadgets.

Figure 16: $\lambda = 0$

Figure 17: $\lambda = 1$

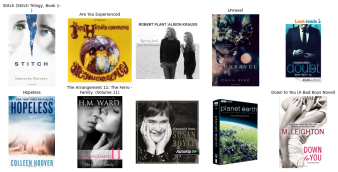

Figure 18: $\lambda = 10$

One the other hand, by adjusting the serendipitous factor λ for SRS, we obtain a range of recommendation results across different weight ratios between relevance and unexpectedness. When $\lambda = 0$, as shown in fig. 16, the SRS recommend uniformly the filter for vacuum, as the sequential model predicts that it's time for the user to change a vacuum filter for the vacuum cleaner previously purchased. When $\lambda = 1$, as shown in fig. 17, the model starts to recommend a balanced mixture of household products across several different genres, as it weighs both relevance and unexpectedness equally. Setting $\lambda = 10$, as shown in fig. 18, we get 10 items in different forms (i.e. books, musics, movies, ...) and across different genres, which the model predicts to be most 'happily' surprising for the user, without much consideration of the user's short-term need. The generation results aligning exactly with our intuition.

We further apply the Serendipity Metric defined in 3.2 to study the effect of the serendipitous factor λ on the recommendation behavior, as illustrated in fig. 19. From the graph, we believe $\lambda = 0.5$ gives the serendipitous recommender the best balance between relevance, diversity, and unexpectedness in terms of the evaluation metrics.

4.3 Model Comparison

To directly compare the SRS against the base model side-by-side, we randomly sample 1000 users. For each user, we generate recommendations on both models, given the user's purchase history. Then we evaluates the recommendation quality using the Serendipitous Metric on the actually purchased item. The model comparison between the base model, and the SRS using Serendipitous Metrics is illustrated in fig. 20. The plot reveals that the SRS out-performs the base model in all of the metrics we defined associated with Serendipity.

5 Conclusion

In our study, we developed a Serendipitous Recommendation System (SRS) using LSTM for relevance prediction and a Latent Factor Model for unexpectedness. Notably, guided by most previous studies which construct serendipitous vectors using heuristics, we conducted sentiment analysis to gather

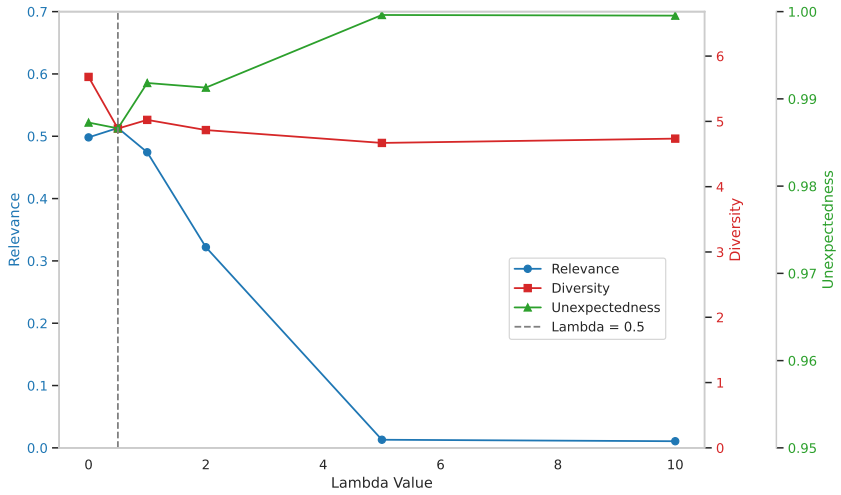

Figure 19: Effect of Lambda on metrics @ k = 10

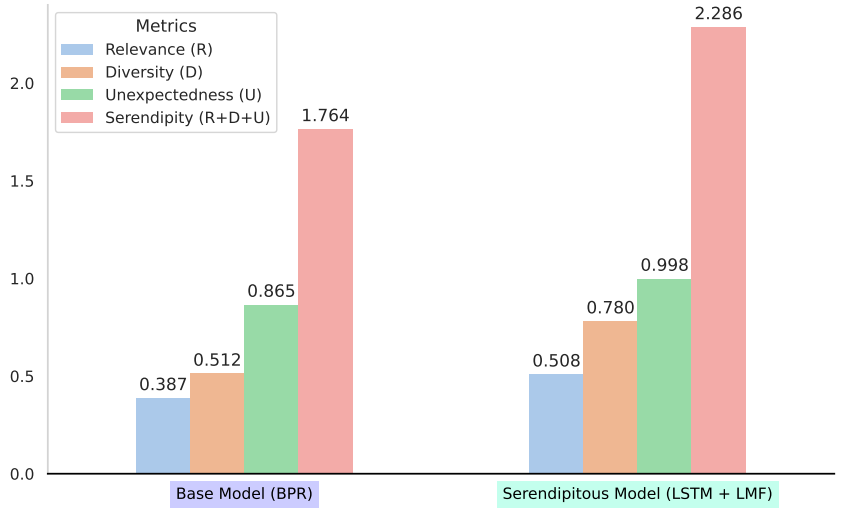

Figure 20: Model Comparison @ k = 10

ground-truth unexpectedness labels to train the model towards being "serendipitous". Our experiments, particularly the comparison plot, revealed significant findings. The SRS demonstrated a balanced performance in terms of relevance, diversity, and unexpectedness, especially at a serendipitous factor (λ) of 0.5. This balance is crucial in meeting user expectations for novelty without losing relevance. The comparison with the Bayesian Personal Ranking baseline underscored the SRS model's ability to offer more diverse and engaging recommendations. These results highlighted the potential of incorporating serendipity in recommendation systems for enhanced user experience.

6 Future Work

Train an end-to-end Serendipitous Model Integrating the two separate models into a cohesive end-to-end framework could offer several benefits. This approach could streamline the training process and potentially yield more harmonized results. Consider exploring neural network-based collaborative filtering methods, as recent advancements have successfully employed neural networks like CNN[9], LSTM[8], RNN[10], and AutoEncoders for improved recommendation systems. These models, such as Neural Collaborative Filtering (NCF)[11] and Neural Network Matrix Factorization (NNMF)[12], have shown high efficacy in personalizing user experiences.

Reinforcement Learning with Human Feedback in Unexpectedness Modeling The Amazon Review Dataset, while extensive, does have limitations, particularly in capturing the element of surprise in user reviews. Not all users explicitly express their surprise in review texts, even when they experience it. To address this challenge, it would be beneficial to construct a Preference Dataset derived from user interactions, and leverage tools like Reinforcement Learning with Human Feedback (RLHF) [13]. RLHF is an emerging area that could significantly enhance surprise modeling in recommendation systems. Implementing reinforcement learning that incorporates human feedback could address the limitations of surprise detection in review texts. This approach would allow the model to adapt based on user interactions, potentially improving the quality of recommendations. However, the challenge lies in effectively capturing and interpreting user feedback, which may not always be explicitly expressed in review texts.

Leverage Language Models for Sequence Modeling The use of advanced language models like BERT [14] and other transformers-based model in sequence modeling solutions like Bert4Rec[15] can offer new dimensions into the recommendation system. These models can better understand user preferences and behaviors by analyzing complex patterns in user data. Given their success in various domains, integrating such models could significantly enhance the system's ability to generate more relevant and unexpected recommendations.

References

[1] Amazon Web Services. Aws personalize. https://aws.amazon.com/cn/personalize/, 2023.

[2] Netflix, Inc. How netflix's recommendations system works. https://help.netflix.com/en/node/100639, 2023.

[3] Alexis Anzieu. Introducing serendipity into recommendation algorithms. https://medium.com/ssense-tech/introducing-serendipity-into-recommendation-algorithms-fb92af88ee0b, Aug 2019.

[4] J. J. McAuley and Jure Leskovec. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In *Proceedings of the 22nd International Conference on World Wide Web*. WWW, 2013.

[5] Li Xueqi, Jiang Wenjun, Chen Weiguang, Wu Jie, Wang Guojun, and Li Kenli. Directional and explainable serendipity recommendation. In *Proceedings of The Web Conference 2020*, pages 122–132, 2020.

[6] Li Xueqi, Jiang Wenjun, Chen Weiguang, Wu Jie, and Wang Guojun. Haes: A new hybrid approach for movie recommendation with elastic serendipity. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1503–1512, 2019.

[7] Steffen Rendle et al. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.

[8] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition, 2014.

[9] Author's First Name Initial. Author's Last name. A comprehensive guide to convolutional neural networks – the eli5 way. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[10] Robin M. Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview. *ArXiv*, abs/1912.05911, 2019.

[11] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pages 173–182, 2017.

[12] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. Neural collaborative filtering vs. matrix factorization revisited. In *Fourteenth ACM Conference on Recommender Systems (RecSys '20)*, pages 1–9, New York, NY, USA, 2020. ACM. Virtual Event, Brazil.

[13] Zihao Li, Zhuoran Yang, and Mengdi Wang. Reinforcement learning with human feedback: Learning dynamic choices via pessimism. May 2023. Manuscript.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. Google AI Language, 2018. Available: jacobdevlin, mingweichang, kentonl, kristout@google.com.

[15] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer, 2019.