Master Thesis Report

Combining transaction and page view data for more accurate product recommendations

Soroush Rohani

August 11, 2023

Supervisor: Martin Rosvall
Examiner: Ludvig lizana
Spring 2023
Master Thesis, 30 ECTS
Master of Science in (engineering physics), 300 ECTS

Abstract

Recommendation systems are primarily used in e-commerce and retail to guide the user in a vast space of available items by providing personalized recommendations that fit the user's interests and need. Numerous types of recommendation systems have been introduced over the years. The most recent development in the field is the sequential recommendation system. Sequential recommenders account for the order in which the user has interacted with items to infer the user's intent, allowing them to provide recommendations accordingly. The data analytic company Siftlab AB has already developed such a recommendation system; however, its application has been limited to transaction data(data depicting only purchases). As a result, the model cannot take advantage of the predictive values of different event types. This thesis introduces a weighted multi-type technique that allows Siftlab's recommendation model to leverage page views alongside purchases in data from an interior design store. We also developed tools and techniques, such as correlation and angle separation analysis, to enhance our examination of user-item behavior. Our research findings indicate that including page view events in training hurts recall, while their inclusion in the prediction stage yields slight improvements. We discovered a rapid decline in correlation between purchases and page views as we considered page views occurring relatively further back in time. Performing a time-based correlation analysis, it became evident that there is a robust time dependency between purchases and page views. Utilizing this time dependency, we enforced a time-dependent threshold on the page views we included in the prediction stage to eliminate irrelevant page view events, further enhancing the model's predictions. We also captured seasonalities phenomena distinctive for an interior design store. Although the result of this work might only be valid for a single data set, we anticipate our work to be the first step in the right direction since the technique we introduce here can be effortlessly adapted to analyze other event types in other data, thus uncovering patterns that can further elevate the model's performance.

Contents

1 Introduction

In e-commerce and retail, recommendation systems enhance user experiences by providing personalized recommendations based on individual preferences and helping customers find desired products. These systems are primary examples of machine learning and artificial intelligence, showcasing their efficacy in delivering tangible benefits to businesses [1]. Recommendation systems should guide users in a personalized way to interesting objects in a large space of possible options [2].To maintain relevance, recommendations must align with the evolving user preferences discerned from past interactions since they are subjected to change over time due to various factors such as shifts in hobbies or seasonal transitions. Consequently, the Effectiveness of recommendation systems relies on their ability to detect underlying sequential patterns, enabling them to recognize changes in user preferences and generate appropriate recommendations [4].

However, different types of recommendation systems use different methods to infer preferences. There are two primary types of recommendation systems: personalized and non-personalized [3]. The non-personalized recommendation systems utilize techniques that involve aggregating all users' previous behavior to estimate the popularity of products at different times [3]. Hence, the recommendations are based on how much a product is trending at a given time while neglecting the user's interest, or in other words, approximating it by the average interest of all users during a specific time frame.

There are many types of personalized recommendation systems [5, 6, 7, 8], such as traditional collaborative filtering [9] and content-based recommendation systems [10]. The content base recommendation provides recommendations by matching characteristics and attributes of items to the historical behavior of the user [2]. For instance, if the user has predominantly watched action sci-fi movies, the model will mostly recommend movies in the same genre. As a result, the model suffers from a phenomenon called user bubble [12], which refers to the situation where users are continuously exposed to a limited set of recommendations that reinforce their existing preferences, preventing the exploration of new items and reducing their chances of expanding their range of interests.

Collaborative filtering recommendation systems generate recommendations by harnessing a group of users' collective behavior and preferences [11]. This user group is determined based on similarity measures, where the system identifies users exhibiting similar preferences and groups them together. This similarity is calculated based on their interactions with items, such as ratings, reviews, clicks, or purchases. For instance, if users A and B exhibit similar interests in a particular product, they may also share the same interests in other products. In this manner, the model approximates a user's interest by considering the average preferences of its corresponding user group. A common way to identify a collaborative filtering recommendation system on online platforms is the statement "Users who are similar to you also liked"(user-based) [13] or "user who liked this product also liked"(item based) [14].

While content-based and collaborative filtering recommendations have the potential to offer personalized recommendations, they are regarded as traditional models in the field of recommendation systems. This is primarily because these models statically treat user preferences, meaning they generalize the user's preference based on the features of items they have been interacting with or the collective behavior of specific users. Generalizing user preferences hinders the model from capturing the temporal dynamics of preferences, resulting in item recommendations that are contextually unrelated to the user's current need [4].

A new category of recommenders called *sequential recommendation systems* have been introduced to address the issue. Sequential recommenders account for the order in which the user has interacted with items. A sequential recommendation system suggests items by considering the history of a user's interactions which we refer to as *context*. The system analyzes the user's context to understand their interests and behavior patterns. It then utilizes this information to suggest relevant items or content that aligns with the user's current context and anticipated needs.

Siftlab AB offers one such sequential recommendation model, which attempts to predict the next item the user will buy using a model proposed for YouTube recommendations [17], which is a modified derivation of Skip-Gram model [18, 16]. The model learns the context of a user's watch history and provides suitable recommendations accordingly. In addition, Siftlab's model employs time series analysis [22] to assess time-dependent popularity for each item to prevent overestimating the personal affinity to those items.

However, Siftlab's implementation only uses users' long-term transaction history(Purchases). Consequently, the model carries three crucial limitations. First, it cannot distinguish between different event types during the training and prediction stages, meaning a "deletion from cart" event impacts the prediction as much as a "transaction" event. Second, it depends on long-term interaction sequences, which causes the model to underperform when it comes to new users due to their short transaction history. Third, it defines an upper bound for the number of previous interactions it considers in its predictions to avoid accounting interactions that are too old and thereby irrelevant to the user's current interest. However, the chosen upper bound value is arbitrary and independent of time, meaning the model solely relies on the internal order of interactions while disregarding how far back in time the interaction has occurred. As a result, there remains a significant risk of incorporating highly irrelevant interactions. This issue becomes even more evident since the model does not consider the order or relative positions of the interactions included in the context. Thus, all context interactions have the same weight in determining the user's current interest regardless of how long before they have occurred.

Siftlab AB has taken note of the model's disadvantages and is eager now to investigate the model's performance on data that includes event types other than transactions. As a first step in the right direction, they provided me with a data set consisting of event types: transactions and page views. Siftlab is specifically interested in predicting only the next transaction, meaning the page views are only allowed to assist the model in its prediction task during both training and evaluation but are not considered prediction targets. Hence, Siftlab's primary wish is to investigate whether the model's performance will improve by adding page views to the data and, if so, what possible model enhancements we can implement to enable maximum leverage from page views in our predictions.

To achieve their objectives, I implemented a weighted multi-type approach in both the training and prediction stages. This approach enables the model to effectively differentiate and assess the predictive values associated with the two types of events. To better understand how page views can contribute to the model's performance, I conducted a test where I modified the context arrays by substituting older purchases with more recent page views. The main goal is to compare the predictive values of recent page views with those of older purchases.

We examined the impact of incorporating page views for users with limited purchase history to investigate how adding page views can affect the model's predictive capabilities for these users. In addition to these empirical tests, we also employed correlation analysis and angle separation analysis. These methods allowed for a more theoretical exploration of the relationship between page views and transaction events as a function of the time difference and the number of interactions between the two events. The goal was to determine whether we can extract more information from page views if we threshold their inclusion in the context based on time or the internal order of interaction sequences. The reader should also be aware that all we present in this thesis have been applied to only one data set belonging to an interior design store.

2 Theory

The recommendation algorithm I generalized solves a multi-class classification problem using neural networks.

2.1 Multi-class classification problem

The multi-class classification problem is a machine learning problem where each input to the model is assigned to one and only one class among multiple available classes. One discrete example is utilizing ML models to identify the type of animal depicted in a picture. In this case, the model's task would become a multi-class classification problem, which seeks to assign an animal class to a picture input. However, how is this concept related to recommendation systems?

For simplicity, let us assume that we have a data set containing the total amount of 5 products, and then start by investigating a user that has first bought a product we refer to as A and then proceeded to buy another product we refer to as B. We then define our *prediction task*: Given that the user has bought product A, we wish to predict which product the user will buy next.

By this definition, we are treating product A as an input to which we want to assign a product class which should supposedly be the next product the user will buy. However, we already know from the data that the true answer is B. Hence, we call product B the *true class/label*, while we refer to other products as classes or negative classes.

In the context of recommendation systems, we refer A as *context/history* to the *label* B.The context reflects the user's previous interactions from which we want to estimate the user's intent or preference. Thus, our ultimate objective is to introduce a model capable of accurately predicting labels based on their corresponding contexts. However, the term "accurately" is very vague since there are several different metrics used to evaluate the performance of a machine learning model depending on the specific task or application area. Due to Siftlab's recommendation, we only used *recall* as a metric in my study, for which we refer the reader to section 2.9 for an explanation.

Furthermore, it is essential to acknowledge that in a multi-class classification problem, the model usually bases its predictions on the output of a Softmax function and returns the product class with the highest probability as its primary prediction.

2.2 Softmax function

The Softmax function, also known as the normalized exponential function, takes any arbitrary vector of real numbers $\mathbf{Z} = \{z_1, z_2, ..., z_n\} \in \mathbb{R}^n$ and converts it to a probability distribution $P = (p_1, p_2, ..p_n)$ such that the probability for each value z_i is given by

$$p_i = Softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}, \tag{1}$$

where we denote the sum in the denominator as a normalization constant since its task is to normalize the exponentiated values. The exponentiation of each value ensures that the outputs are always positive, while division by the normalization constant ensures the values are within 0 and 1. Further investigation of equation 1 also reveals that summing all probabilities will add up to 1 since

$$\sum_{i}^{n} = p_i = \sum_{i=1}^{n} \frac{e^{z_i}}{\sum_{j=1}^{n} e_j^z} = \frac{\sum_{i=1}^{n} e^{z_i}}{\sum_{j=1}^{n} e_j^z} = 1.$$

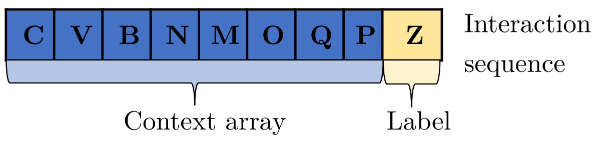


Figure 1: The figure illustrates a user's interaction sequence. Each entry represents an interaction. The capital letters denote items the user has interacted with, and the sequence is sorted, with older interactions to the left. The context portrays the user's previous behavior before the label. The context array is mapped into the embedding space by equation 2 where, for instance, θ_1 corresponds to the embedding vector of the first interaction, which in this case, we refer to as C.

In machine learning, particularly within recommendation systems, the set of unnormalized values $\{z_1, z_2, ..., z_n\}$ are produced by the model and are somehow connected to the products. However, incorporating product-ids in mathematical equations entails establishing mathematical representations of them.

2.3 Embedding space

An embedding space in its core is a vector representation of an entity. In recommendation systems, an embedding space is a vector space in which the products and users are mapped. The embedding space offers a mathematical representation of products while reflecting the extent of their similarity.

The similarity between two products is typically determined by the proximity of their respective embedding vectors. This similarity quantifies the degree of resemblance or overlap in characteristics, features, or patterns among products. Consequently, embedding spaces play a critical role in depicting product similarities based on their latent features and characteristics, serving as the primary foundation for the model's predictions.

All recommendation systems start by randomising N dimensional vectors for each item, which we denote as $\Theta = \{\theta_1, \theta_2, ..., \theta_n\} \in R^N$ and refer to them by *embedding vectors* or *product vectors.*

The product vectors will then transform during the training such that the distance between the product vectors and the users mapped into this embedding space reflect the similarity between the user's preference and the product they will interact with next. The model's user mapping method varies depending on the desired level of complexity and robustness one wants the model to acquire.

In section 2.1, we took a simple interaction sequence consisting of two interactions. Now let us take a user's interaction sequence consisting of nine interactions, where we take the last interaction as a label and the rest as context. To represent this sequence of previous interactions in our embedding space, Siftlab's model maps the context array into the embedding space by L2 normalizing the sum of its products' embedding vectors by

$$\mathbf{e_c} = \frac{(\sum_{i=1}^{l} \theta_\mathbf{i})}{\|(\sum_{i=1}^{l} \theta_\mathbf{i})\|_{L_2(\Omega)}}, \tag{2}$$

Where $\theta_\mathbf{i}$ is the L2 normalized embedding vector of the product in the ith interaction of the context array. l denotes the length of the context, and $\mathbf{e_c}$ is the *dense vector representation* of the context, which is how the model maps a user into the embedding space. See figure 1 for visualization. Note that applying equation 2 on the context array essentially creates an aggregated representation of the user's history. The assumption is that this aggregated representation captures the average characteristics or preferences of the user.

This approach is desirable when the order or sequence of the interactions is less important than the overall user preferences or behavior. It simplifies the modeling process by reducing the sequential nature of the interactions into a single representation.

2.4 Negative sampling

In most machine learning tasks, one usually deals with huge data sets, requiring methods to reduce the computational load to achieve faster training without losing significant accuracy. *Negative sampling* is a technique designed specifically for this task.

For simplicity, let us go back to the example presented in section 2.1 with an interaction sequence consisting of only two interactions: first purchase of product A followed by a purchase of product B. As mentioned before, product B becomes the label with a context that consists only of a single product A. The context vector $\mathbf{e_c}$ would according to equation 2 become

$$\mathbf{e_c} = \frac{\theta_A}{\|\theta_A\|},$$

where θ_A is the vector representation of product A.

The conventional way a neural network would approach this would be solving an n-class classification problem, where $n = 5$ is the total number of products within our data. A softmax function would determine the final prediction task by

$$\text{softmax}(r|c) = p(r|c) = \frac{e^{\theta_\mathbf{r} \cdot \mathbf{e_c}}}{\sum_{j=1}^{5} e^{\theta_\mathbf{j} \cdot \mathbf{e_c}}}, \tag{3}$$

where r is the target product with embedding vector $\theta_\mathbf{r}$ and c is the context with dense vector representation $\mathbf{e_c}$. Comparing equation 3 to equation 1, we see that $z_i = \theta_\mathbf{i} \cdot \mathbf{e_c}$ indicating that for each context vector, the set of values we convert to probabilities are the dot products between the context vector $\mathbf{e_c}$ and each product vector in Θ. The result from a dot product describes how close two vectors are. Hence, for any given context vector $\mathbf{e_c}$, the product with embedding vector closest to $\mathbf{e_c}$ will attain the highest prediction probability and thus will serve as the primary prediction of the model.

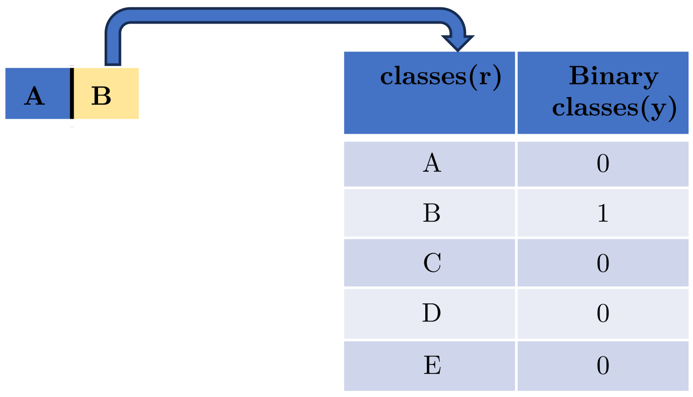
However, since we are dealing with a large data set, evaluation of equation 3 for each context(which in our case should be in the order of 10^5) would be computationally intensive. We could, however, reduce the problem to a binary classification problem by denoting the actual target/label product as 1 and all other products as 0, which reduces the number of classes to two and facilitates the classification task. See figure 2. The probability of a particular product being next is then given by

$$p(y|r, c) = \sigma(\theta_r^T \cdot e_c),$$

where σ is a Sigmoid function [15] and y is the binary class that can take on values 1 or 0. The Sigmoid function returns a value between 1 and 0 on which the model will apply a threshold of 0.5. Meaning for $(\theta_r^T \cdot e_c) < 0.5$, the sample r will be classified as a negative sample, otherwise as a positive sample.

Further inspection reveals that the ratio between the positive cases (when the target is product B) to negative cases is $\frac{1}{(n-1)}$. Since the number of products one usually deals with in a recommendation system is of the order $n = 10^5$, it is clear that a substantial imbalance exists between the negative and positive instances.

The imbalance between negative and positive instances can lead to several issues. The model may become biased toward predicting the negative class (majority class) due to its overwhelming presence in the data. As a result, the model may struggle to identify and



classes(r)	Binary classes(y)
A	0
B	1
C	0
D	0
E	0

Figure 2: The figure shows an interaction sequence to the left, with B being the label. We aim to classify the context with an item, and to this end, the left column on the table shows all available classes. We can observe how the number of classes reduces from four (A, B, C, D, F) to two(0 and 1) when we go from a multi-class classification to a binary classification.

classify positive instances accurately.

Additionally, With many negative instances, the model may have limited exposure to positive instances, which can lead to insufficient learning of the characteristics and patterns specific to the positive class, making it challenging to differentiate between positive and negative instances effectively.

Negative sampling offers an efficient solution to this problem by sampling a subset of negative instances used during training [18]. Instead of explicitly including all negative instances, negative sampling randomly selects a small number of negative examples per positive example. This approach reduces the computational burden while maintaining a balanced training set [19].

Regardless of how negative sampling improves efficiency, it still suffers several disadvantages. One is that the selection of negative samples does not reflect the accurate distribution of the products in question since it reduces the number of classes to two(ones and zeros). It can also lead to the loss of information we could gain by considering the entire set of classes. Hence, we employ a training procedure called sampled softmax to maintain the efficiency of negative sampling and preserve useful information about the class relationships.

2.5 Sampled Softmax

Sampled Softmax approximates the softmax function in equation 1 by randomly sampling a batch of classes (positive and negative samples) for each training example during the softmax calculation. The sampled softmax function then becomes

$$\text{softmax}(r|c) = p(r|c) = \frac{e^{\theta_{\mathbf{r}} \cdot \mathbf{e_c}}}{\sum_{j=1}^{k} e^{\theta_{\mathbf{j}} \cdot \mathbf{e_c}}}, \tag{4}$$

where k is the number of sampled classes.

Hence, Instead of considering all classes, only a limited number of interactions are sampled, reducing the computational burden. The underlying concept is that sampled softmax approximation captures the intricate relationships between sampled classes and can accurately estimate the full softmax function by incorporating contributions from the sampled classes.

2.6 Gradient decent

Gradient descent is an iterative optimization algorithm that determines local minimum points of multi-variable functions. In machine learning, gradient descent minimizes the loss function during each backpropagation in a neural network, which we have outlined in section 2.7. The general idea is to take repeated steps in the inverse direction of the gradient, which we will show is the steepest descent towards a local minimum.

Consider a two-variable function $f(x, y)$. We denote the partial derivatives of this function by

$$f_1 = \frac{\partial f}{\partial x}$$
$$f_2 = \frac{\partial f}{\partial y}.$$

The partial derivatives f_1 and f_2 of a function $f(x, y)$ determine how much the function changes in the x and y direction. Letting $\mathbf{v}$ denote a unit vector in an arbitrary direction, the directional derivative then becomes

$$\nabla \mathbf{f} . \mathbf{v} = \|\nabla \mathbf{f}\| \|\mathbf{v}\| cos(\theta),$$

where $\nabla \mathbf{f} = (f_1, f_2)$ is the gradient vector of $f(x, y)$ and θ is the angle between $\nabla \mathbf{f}$ and $\vec{v}$. The directional derivative is achieved if $\theta = 0$. Hence, the direction of $-\nabla \mathbf{f}$ is the steepest downward surface $z = f(x, y)$.
Considering this, we can go down the surface utilizing the recursive scheme

$$\begin{aligned} x_{i+1} &= x_i - af_1(a, b) \\ y_{i+1} &= yi - af_2(a, b) \end{aligned} \tag{5}$$

where index $i = [1, n]$ denotes the iteration number and a is a small constant. Repeating this a sufficient amount of iterations reveals the position of a local/global minimum referred to by (x_n,y_n) in the context of equation 5.

2.7 Neural networks

Neural networks(NN) are a class of machine learning models inspired by the structure and functioning of the human brain. Here we will provide a basic description of neural networks, their basic components, and their working principles to a level that guarantees comprehension of our work.

2.7.1 Overall structure

A neural network consists of three layer types: an Input layer, hidden layers, and an output layer. The input layer receives and passes the input data to the subsequent hidden layers. Dependent on the application, the inputs can represent different features or attributes of the data.

The hidden layers are the intermediate layers between the input and output layers. They consist of several neurons, and each neuron receives its input from the previous layer on which it performs mathematical operations and applies a nonlinear activation function to produce an output. The output of a hidden layer can either serve as input to another hidden layer or input to the output layer. *Forward propagation* is the computation flow from the input to the output layer.Figure 3 provides a visualization of a NN.

The output layer includes the predicted probabilities for each class in our classification problem. Hence, the number of nodes in the output layer should equal the number of classes in

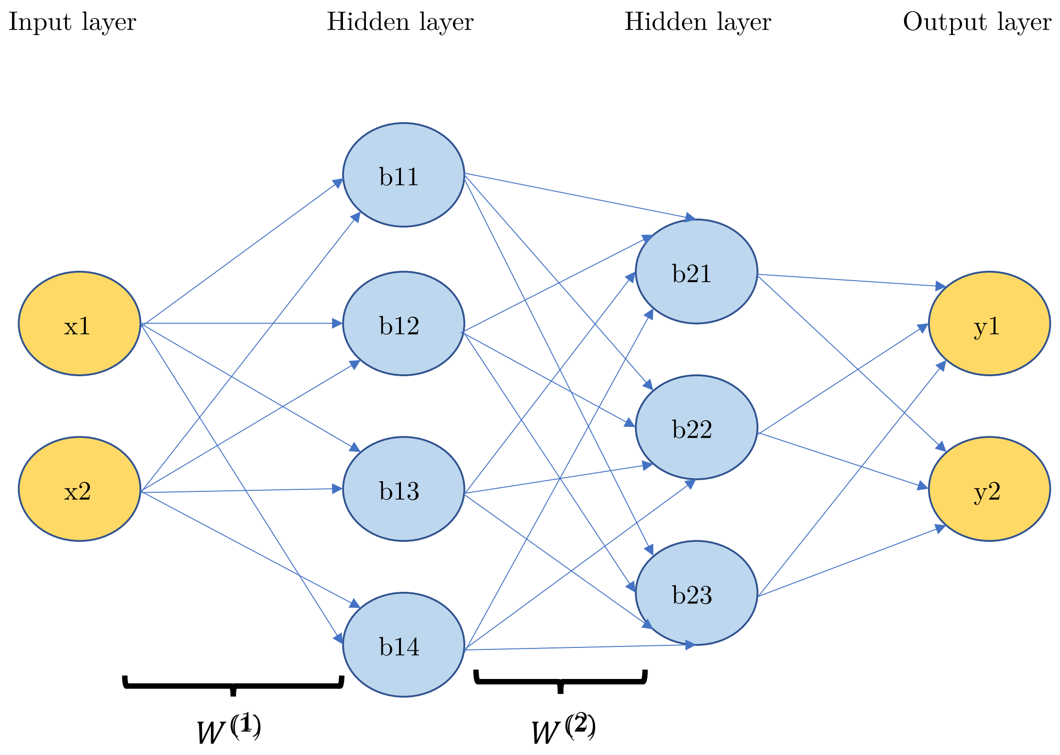


Figure 3: A demonstration of a neural network with two inputs, two hidden layers, and two output nodes. Neuron refers to the nodes in the hidden layer responsible for propagating the information from the input to an output. The arrows show the inputs to each neuron from the previous layer. The nodes in the output layer store probability $P(y_1|X)$ and $P(y_2|X)$ after each forward propagation.

our classification problem, see figure 3.

The probabilities in the output layer are compared to the ground truth, and loss is calculated using a loss function. In machine learning, we denote the class we know from the data as the ground truth as labels or true classes. We use labels as references to evaluate the model's recall and performance.

The network then minimizes the loss by doing a backpropagation [20] in which it starts from the output layer and, for each layer, calculates the gradients of the loss with respect to the weights and biases. The weights and biases are then updated using gradient descent [21], which minimizes the loss. Hence, the model adjusts its weights and biases to predict the labels according to the training data.

2.7.2 Neurons and activation functions

Neurons in a hidden layer accept multiple inputs on which they apply an activation function on the weighted sum. Given an arbitrary activation function, the output of neurons in the first layer is given by

$$\mathbf{Y}^{(1)} = f_{activation}(\mathbf{W^{(1)}} * \mathbf{X}) + \mathbf{b^{(1)}})$$

Where $\mathbf{Y}^{(1)}$ and $\mathbf{b^{(1)}}$ denotes the outputs and biases from neurons in layer 1 ,respectively. $\mathbf{X}$ are the set of inputs, and $\mathbf{W}^{(1)}$ is the weight matrix associated with the connection between the inputs and neuron in layer 1. The outputs will either be inputs to neurons on subsequent layers or inputs to the output layer. In the case of several hidden layers, the

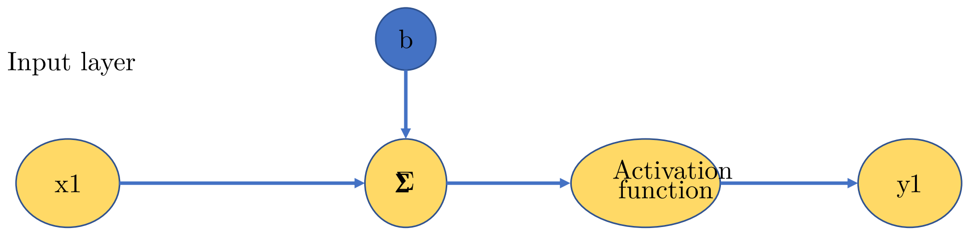


Figure 4: The figure shows a perceptron with only one input. The $\sum$ will measure the weighted sum of the input and add bias to it. It will later apply an activation function on it to produce a predicted probability for the class denoted by y_1.

output of neurons in layer j is determined by

$$\mathbf{Y}^{(j)} = f_{activation}((\mathbf{W}^{(\mathbf{j})} * \mathbf{Y}^{(j-1)}) + \mathbf{b}^{(\mathbf{j})}), \tag{6}$$

where $\mathbf{Y}^{(j-1)}$ represents the outputs from the previous layer, $\mathbf{W}^{(\mathbf{ij})}$ represents the weight matrix between neurons in layer $j-1$ and j, and $\mathbf{b}^{(\mathbf{j})}$ are the biases of neurons in layer j.

An *activation function* is a function that aims to introduce nonlinearity into the network. By analyzing equation 6 in connection to the overall structure of NN in figure 3, we can state that without an activation function, the output nodes will be a linear combination of its inputs which essentially reduce the model to a linear model, restricting its ability to detect complex nonlinear patterns [21].

2.7.3 Perceptron

A perceptron, in its most basic form, refers to a single computational unit or neuron that takes multiple inputs, applies weights to them, and produces an output. It is a special case of a neural network without any hidden layer with the difference that we can associate bias with its output node. see figure 4.

For simplicity, let's investigate a perceptron with only one input. According to figure 4, the value in the output node is

$$y = w * x + b,$$

where x is the input, w is the weight, b is the bias and y if the output.

Assuming we are using a sampled softmax loss function and we have only one label among the samples, the loss becomes

$$L = -log(P(l|x)), \tag{7}$$

where l refers to the label and $P(l|x)$ in our model is a sampled softmax function for k samples as in equation 4 which results in

$$P(l|x) = \frac{e^{(w_l \cdot x + b_l)}}{\sum_{j=1}^{k} e^{(w_j \cdot x + b_j)}}.$$

The goal is now to minimize $L(\mathbf{W}, \mathbf{b})$, where $\mathbf{W}$ and $\mathbf{b}$ are the set weight and biases used in the sampled softmax function. Considering the gradient decent depicted in section 2.6, the weights of all the samples will be updated according to equation 5 by

$$w_j = w_j - \eta * \frac{\partial L}{\partial w_j}, \tag{8}$$

where η is the learning rate and w_j is the weight for the sample j.

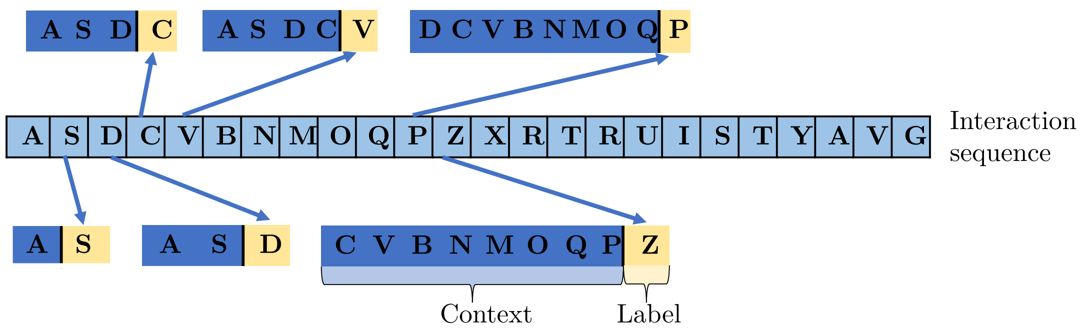


Figure 5: An illustration of how the model assigns labels and contexts to a user's interaction sequence with $MaxLen = 8$. The model takes each interaction (except the first) as a label and assigns them a context array.

Please note that minimizing $L = -log(P(l|x))$ will, in turn, maximize $P(l|x)$, which is the probability of the model predicting the label l given the input x. Since P is a probability distribution, maximizing $P(l|x)$ is equivalent to minimizing $P(j|x)$ for all $j \neq l$.Hence, during each backpropagation, the perceptron adjusts its weight to minimize the loss and increase the probability to predict the label and decreasing it for predicting negative samples.

2.8 Siftlab's model

In this model, each product has a L_2 normalized vector representation in a 32 dimensional embedding space. The baseline idea is to learn the embedding space of products to map a user's sequence of preceding interactions with products to predict which product they will interact with next. To this end, the model takes each interaction in a user's sequence as a label and assigns to it a context. To avoid accounting interactions that are too old for predicting the label, we defined a parameter called $maxLen$ which controls the maximum length a context array can attain, see figure 5.

The model applies equation 2 to map the contexts into the embedding space. However, due to the large number of products in the data, feeding all contexts and labels to the neural network would result in a multi-class classification problem with millions of classes. This approach would be computationally intensive and time-consuming. To address this issue, the model splits the labels and their corresponding contexts into batches of size 32.

For each label, the model randomly selects $k-1$ negative samples, i.e., products known not to be the next item the user will interact with after a given context. Within each batch, the model passes every context vector to a system composed of k perceptrons. Each perceptron takes the same context vector $\mathbf{e_c}$ as input and has a single output node. Among these output nodes, $k-1$ is responsible for storing the predicted probabilities of the negative samples, while one node stores the predicted probability for the label.

In this process, the model sets the weights and biases of each perceptron to the embedding vectors and trends of the corresponding product, aligning with their respective output nodes.

Ultimately, the model employs the sampled softmax function described in equation 2 to calculate losses and gradients, as depicted in equation 7. Consequently, during each back-propagation step, the weights (now the embedding vectors $\theta_{\mathbf{r}}$) of the labels and negative samples are updated according to equation 8.

Thus, the embedding space will be shaped such that given a context vector $\mathbf{e_c}$, a product vector $\theta_\mathbf{r}$ and a trend value t_r the sampled softmax

$$\text{softmax}(r|c) = P(r|c) = \frac{e^{(\theta_\mathbf{l} \cdot \mathbf{e_c} + log(t_r))}}{\sum_{j=1}^{k} e^{(\theta_\mathbf{j} \cdot \mathbf{e_c} + log(t_j))}}, \tag{9}$$

will give the highest values when $r = l$.

To accurately capture the relations between products, we train the model for several epochs where each epoch consists of all batches in random order. The randomization of batches order in an epoch prohibits the model from becoming biased toward certain patterns. To avoid overfitting, we monitor the mean loss in training after each epoch and stop the training if we do not observe any further decrease in loss.

The reader should also be aware that the model is supported by a baseline model based on time series analysis [22], which aims to measure the popularity of a product at the time of prediction. However, in this work, we will only focus on modifying and improving the machine learning model.

Ultimately, the model calculates the probability of a certain product to be the next product (which I will also refer to as predicting probability) the user interacts with by multiplying the log of trends by the softmax probability resulting in

$$P(r|c) = t_r \cdot \exp\left(\theta_\mathbf{r} \cdot \mathbf{e_c}\right) \cdot C,$$

where t_r is the trend of product r.

To prevent numerical accuracy and performance problems, we perform these calculations in log space

$$log(P(r|c)) = log(t_r) \cdot (\theta_\mathbf{r} \cdot \mathbf{e_c}) \cdot log(C). \tag{10}$$

Investigating equation 10 by considering that $\mathbf{e_c}$ and $\theta_\mathbf{r}$ are L2 normalized, we can see that the model bases its prediction on the direction of the vectors $\mathbf{e_c}$ and $\theta_\mathbf{r}$, meaning if the user interacts with products whose embedding points in a specific direction, then they are more likely to interact with other products in that direction.

Now that we have established how the model functions, it is time to define a metric we could use to evaluate the model's performance. As mentioned before, several different metrics could serve as a valuable asset. However, Siftlab was mainly interested in the *recall* metric for reasons described in the next section.

2.9 Recall

Recall in a machine learning model refers to the model's ability to predict the labels. More specifically, let us denote contexts for which the model has predicted labels correctly as true positives(TP) and contexts for which it failed to provide accurate prediction for False negatives (FN). More precisely, we call these cases FN since the model deems the label irrelevant to the user. The recall is calculated by

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

However, it would rarely happen for the label class to achieve the maximum predicting probability in equation 10 among millions of classes. Therefore, we approximate the recall using a metric called *recall in top k*, which evaluates whether the label is present within the top k predictions.

The recall measures the proportion of relevant items (labels or items we know from the data to be in the user's interest) recommended to the users. Therefore, Maximizing recall becomes imperative as it ensures that our recommendations are in harmony with users' preferences, resulting in a personalized recommendation system. Ultimately, this enhances user satisfaction and augments the likelihood of purchases, benefiting both users and the platform.

2.10 Spearman's correlation

Spearman's correlation is a statistical measure that estimates the monotonic relationship between the ranks of two variables. It is beneficial when the relative ordering of values is more meaningful than their actual numerical values and also when one is sure the data sets are not normally distributed.

Assume we have two data sets, X and Y. To calculate Spearman's correlation, we use

$$r_s = \frac{COV(R(X), R(Y))}{\sigma_{R(X)} \cdot \sigma_{R(Y)}} \tag{11}$$

where $R(x)$ and $R(Y)$ are ranked X and Y, $COV(R(X), R(Y))$ is the covariance of rank variables, $\sigma_{R(X)}$ and $\sigma_{R(Y)}$ are the standard deviations of rank variables.

3 Method

3.1 Data processing and cleansing

Siftlab provided us with a .csv file with interactions as rows and columns, including features user-id,product-id, Timestamp, and weights. To distinguish between event types, the transactions were assigned weights 1, and page views weight 0, allowing us to filter the data during training and prediction stages.

We converted the CSV file to binary and memory mapped on files for faster data processing. Memory mapping grants us efficient memory usage since we only load the needed portions instead of loading the entire file into memory. It also provides us with faster memory access making it a suitable technique to apply to large data sets.

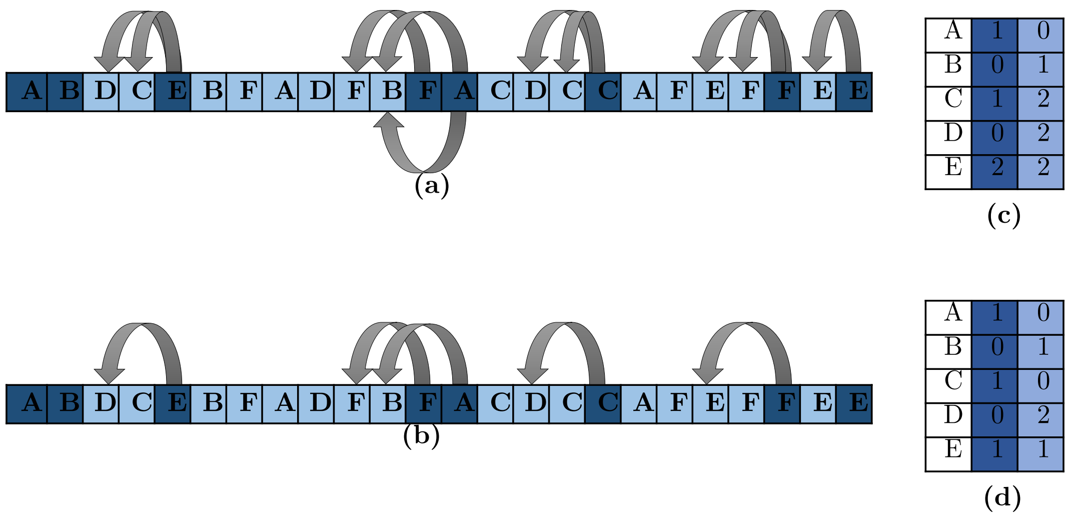
Moreover, before starting with the main work, we needed to eliminate noise sources in the data. One noise source is the website refreshes recorded as several consecutive page view events between the same user and item. We identified these instances and merged all such page view events into one to prevent website refreshes from distorting the user's preference.

Another noise source is transaction events involving the same user and product transpiring simultaneously. Such occurrences signify purchases of the same product in quantities exceeding one. We again merged these transactions to avoid inducing biases in dense vector representations of contexts.

3.2 Training and Testing

To create training and testing samples, we first sorted the data by time and splited the data into training and test sets by 80% to 20% such that the training set consisted of the older data set.

Using the weights as event-type indicators, I included or excluded page-view events during the training and prediction stages. Thus, I could explore different scenarios to discover the case where incorporating page views improved the model's recommendation.


A B D C E B F A D F B F A C D C C A F E F F E E
(a)
A B D C E B F A D F B F A C D C C A F E F F E E
(b)
(c)
(d)

(c)

A	1	0
B	0	1
C	1	2
D	0	2
E	2	2

(d)

A	1	0
B	0	1
C	1	0
D	0	2
E	1	1

Figure 6: **(a)** and **(b)** show the same interaction sequence where the transactions and page views are marked by dark blue and light blue cells, respectively. The arrows indicate the transactions and page views we count. In **(a)**, we count page views in the range $[1, 2]$ leading up to a transaction, and in **(b)**, we count page views happening 2 interaction before a transaction. Charts **(c)** and **(d)** show the counted transactions(dark blue column) and page views(light blue column) per product for case **(a)** and **(b)**, respectively. We count each interaction only once. The interaction sequence is sorted with older interactions to the left.

3.3 Weight incorporation

The model described in section 2.8 predicts the next product based on previous interactions but does not account for interaction types and their distinct prediction values. Therefore, the model will fail if the data consist of different event types. To account for the disadvantage, we modified equation 2 to instead measure the weighted context vector for each label by

$$\mathbf{e_c} = \frac{(\sum_{i=1}^{l} w_i \cdot \theta_\mathbf{i})}{\|(\sum_{i=1}^{l} w_i \cdot \theta_\mathbf{i})\|_{L_2(\Omega)}}, \tag{12}$$

where w_i is the assigned weight for interaction i.
We should notify the reader that due to simplicity and shortage of time, we kept the weights for transactions at a constant value of 1 while only varying the weights for page views.

3.4 Correlation analysis

As we increase the length of the context arrays, we are considering older interactions that might be irrelevant to the user's intent. Given that the primary focus is predicting the user's next transaction, we want to know how long page views remain relevant for this prediction task.

To answer this question, we devised a correlation study to examine the correlation between transactions and page views within a certain range. The range is either based on the time difference or the number of interactions between the two events. Once the range was defined, we counted the number of page views and transactions for each product accordingly and employed equation 11 to measure Spearman's correlation between the counted values.

We measured the correlation in several ways. First, we counted the pageviews occurring exactly d interactions before the transaction. Second, we counted page views within the $[1, d]$ leading up to each transaction. See figure 6. Finally, we counted the pageviews within a defined time range to investigate potential temporal dependencies.

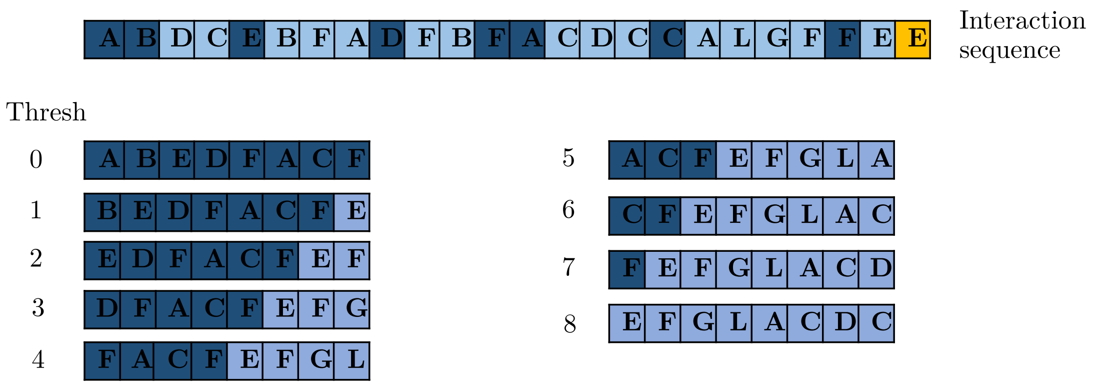


Figure 7: The figure illustrates an interaction sequence of a user on the top and the resulting context arrays below it with $MaxLen = 8$. The context array corresponds to the label marked as yellow, and the interaction sequence is sorted with older interactions to the left. Parameter $Thresh$ defines how many older transactions we push out and replace with a recent page view in the context. We should note that the only interaction we will take as a label in this study will be the last one since it full fills the criteria $n_t \geq MaxLen$ and $n_p \geq MaxLen$.

The correlation study is one of the tools we introduce to examine the relationship between page views and transactions through time or interactions. The study aims to reveal the average behavior of users and enable us to use page views most efficiently.

3.5 Angle separation

As mentioned in section 2.8, the model predicts the next product based on the direction of the vectors in the embedding space. Therefore, products whose vectors point in the same direction are more similar. An effective measure of two vectors' alignment is the angle between them. We can measure the angle between two product vectors by

$$\phi_{ij} = arccos(\frac{\theta_\mathrm{i} \cdot \theta_\mathrm{j}}{\|\theta_\mathrm{i}\| \cdot \|\theta_\mathrm{i}\|}),$$

where θ_i and θ_j are vectors for products i and j, and $\|\|$ is the L2 norm.
Hence, to explore the relevancy of page views, we measured the angle between each transaction product and products associated with page views occurring d interaction before. The average angle between transactions and page views is then given by

$$\hat{\phi}_d = \frac{1}{N} \sum_{i=1}^{N} \phi_{id}, \tag{13}$$

where N is the number of transactions with a page view event d interactions before them, and ϕ_{id} is the angle between transaction i and a page view happening d interactions prior.

3.6 Old transactions versus recent page views

To study how we can best leverage page views, we examined the impact of replacing an older transaction event in the context with the most recent page view on recall. For the experiment to produce accurate results, we filtered labels where the number of previously occurring transactions n_t and pageviews n_p is lower than $MaxLen$. By doing so, we will investigate the recall over the same set of labels each time we introduce a recent page view and eliminate an older transaction.Refer to figure 7 for visual presentation.

To distinguish between the effect of pushing out an old transaction and replacing it with a recent page view, we performed once again the same study where we instead only eliminated the oldest transaction, see figure 8.

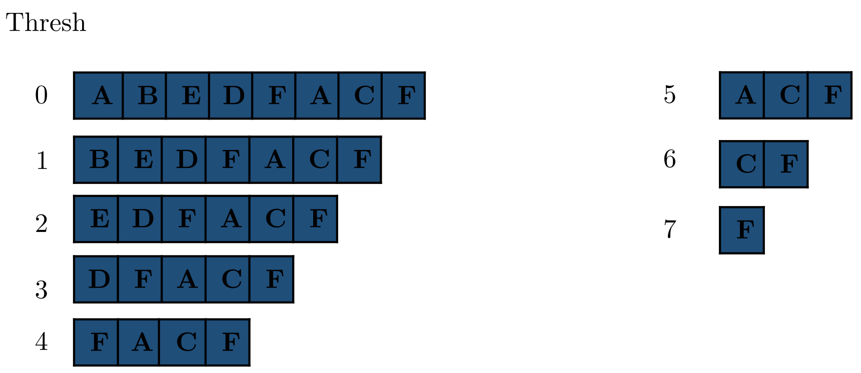


Figure 8: The figure shows the context arrays from the same interaction sequence and labels as in figure 7. The *Thersh* parameter in this study only describes the number of oldest transactions eliminated from the context.

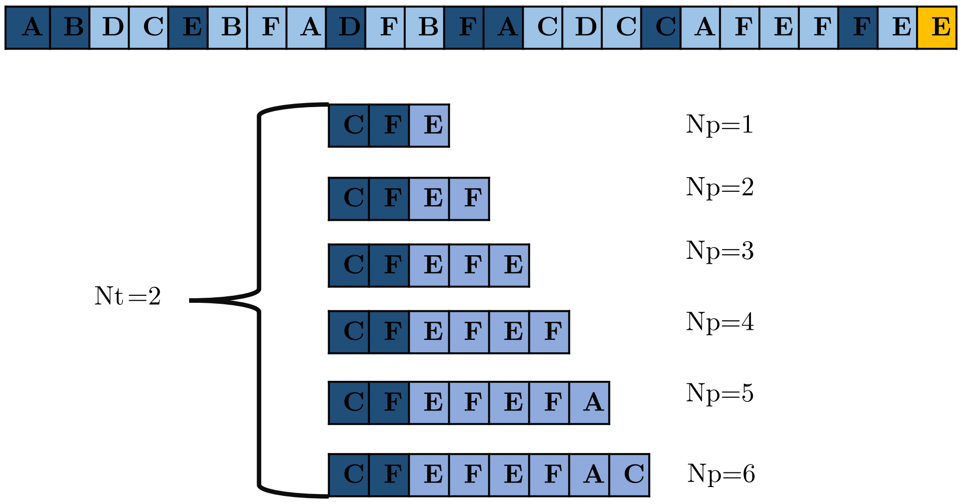


Figure 9: An example demonstrating how we perform the new user study for $N_t = 2$ and $MaxLen = 8$. The user's interaction sequence is shown on the top, while The arrays below it are context arrays to the label marked yellow.

3.7 New users

Another interesting feature is the users who have just started their journey on the website. To identify new users, we defined a new parameter called N_t, which we used to filter labels with less than N_t preceding transactions. Afterward, we constructed the context arrays by first adding the N_t most recent transactions following it by adding the N_p most recent page views and monitoring the recall of the model. The goal here is to explore the predictive value the preceding page views introduce for new users. Look at figure 9 for a visual description.

4 Results

4.1 Weight optimisation

We trained the model once on the data containing only transactions, and once on the combined data both with and without weights incorporation demonstrated in section 3.3. Subsequently, we evaluated each trained model on four distinct prediction tasks.

The first task was predicting the next interaction, meaning we were not interested in the

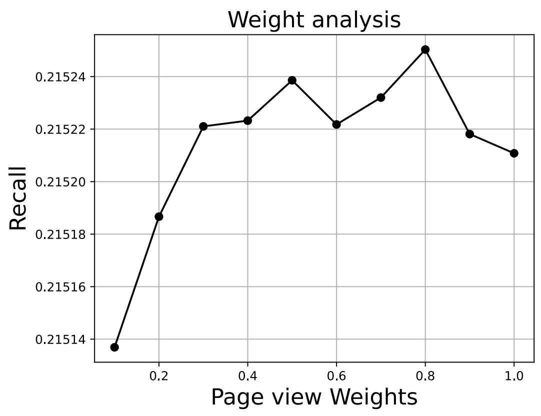


Figure 10: The figure shows how changing the weights assigned to page view changes recall.

interaction type and wanted only to identify the item the user interacted with.

The Second task centered around predicting the next transaction, for which we employed three different techniques. Firstly, we included only previous transactions as the basis for our prediction by including only transactions in context arrays. Secondly and thirdly, with both event types included, but with and without incorporating weights when constructing the context arrays demonstrated in equation 2. Table 1 illustrates the results.

Table 1: The table shows the recall of Siftlab's model with $Maxlen = 8$. Each row represents recall values corresponding to the predictions of the same trained model. Each column shows recall for the same prediction task. Combined data denotes accounting page views in context arrays during training, while weights incorporation refers to section 3.3 where transaction(T) and page views(P) weigh 1 and 0.5, respectively.

	Prediction task			
Training data	Next interaction(P+T)	Next transaction	Next transaction based on previous T and P	Next transaction based on previous T and P with weights incorporated
Combined data	0.0571	0.1901	0.1927	0.1928
Combined data with weights incorporated	0.0485	0.1914	0.1928	0.1927
Transaction data	0.059	0.2135	0.2152	0.2152

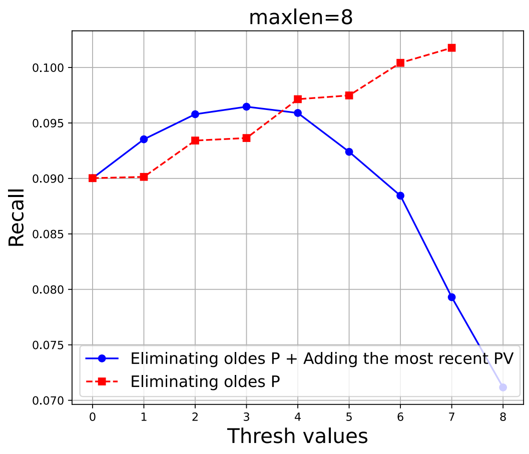
Based on our observation that the model trained only with transactions and incorporating weights in the prediction task yields the best outcome, we decided to investigate the optimization of the model by adjusting the weights we assign to page views while keeping the weights for transactions constant at 1. By monitoring the recall, we found that a weight of 0.8 gives the highest recall. see figure 10.

4.2 Older transaction VS recent page view

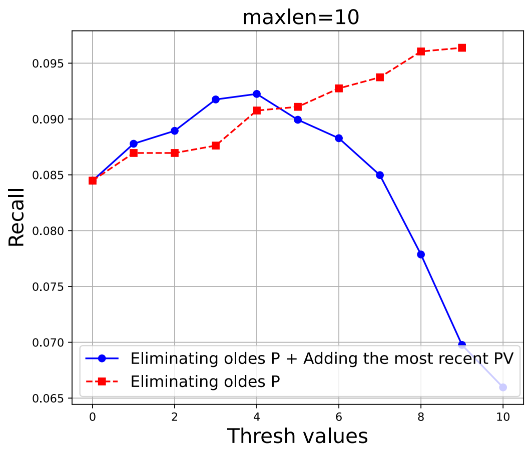
Replacing an old transaction with a recent page view enhanced the model's recall. However, as the page views we are considering becomes older, their positive impact on the recall metric gradually diminishes. As for the case where we only eliminate the earlier transaction without adding any page views, we observe a continuous increase in the model's recall. See figure 11.

4.3 New users' behavior

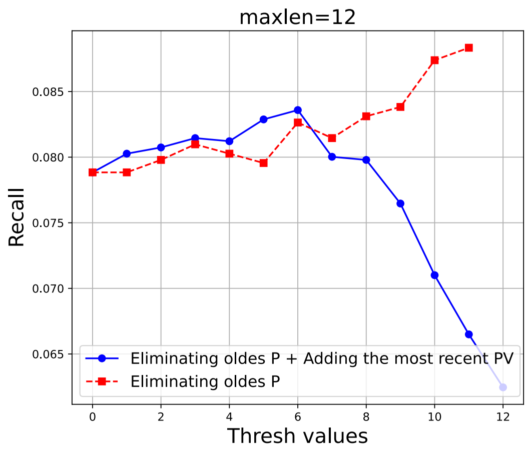
Filtering new users and performing experiments as described in section 3.6 showed that one can always expect a decrease in recall for $N_p > N_t$ while taking the last two-page view is always beneficial, see figure 12.



(a)



(b)



(c)

Figure 11: The figures show the results from the experiments described in section 3.6. The blue line depicts the scenario where we substitute *Thresh* old transactions with *Thresh* recent page views, while the red line corresponds to the case where we solely eliminate *Thresh* oldest transactions. Consequently, for each value of *Thresh*, the context arrays corresponding to the red line and the blue line contain the same transaction events, but the blue line also adds *Thresh* most recent page views to the context arrays, see figure 7 and 8.

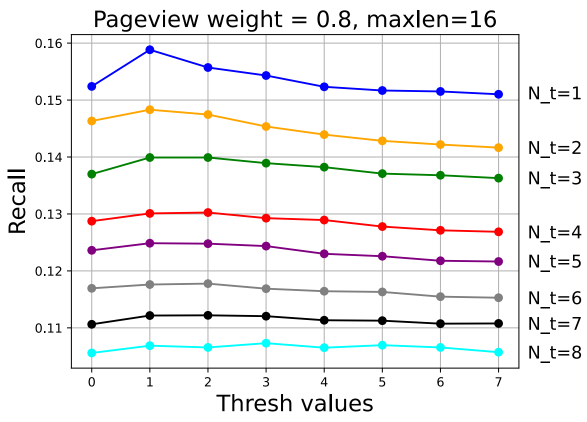


Figure 12: The figure demonstrates the variation in recall as we add N_p most recent page views for users with only N_t number of transactions in their history. Contexts arrays are not allowed to be longer than $MaxLen = 8$, which explains the constant recall after the point where $(N_p + Thesh) \geq 8$.

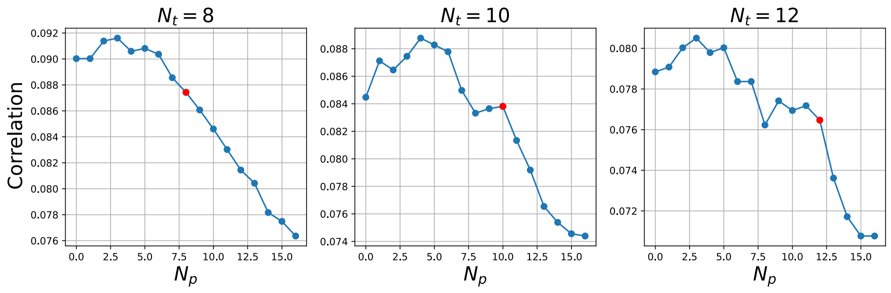


Figure 13: The plots show the same study as in figure 12 but without the upper bound constraint. The points where $N_t = N_p$ are marked red. We can see that the recall will certainly decrease after these points.

Another interesting thing we explored was taking $N_t >= 8$ values, see figure 9, and removing the constraint of the $MaxLen$ parameter on the context arrays, allowing them to be $N_t + N_p$ long. The aim is to study all users and extend the study in figure 12 by avoiding the constant recall measurements due to contexts' upper bound constraints on their length. Figure 13 illustrates the results.

4.4 Correlation analysis

The results from the correlation analysis using approaches described in figure 6 showed that as we go further back, the correlation between page views and transactions starts to decrease, see figure 14.

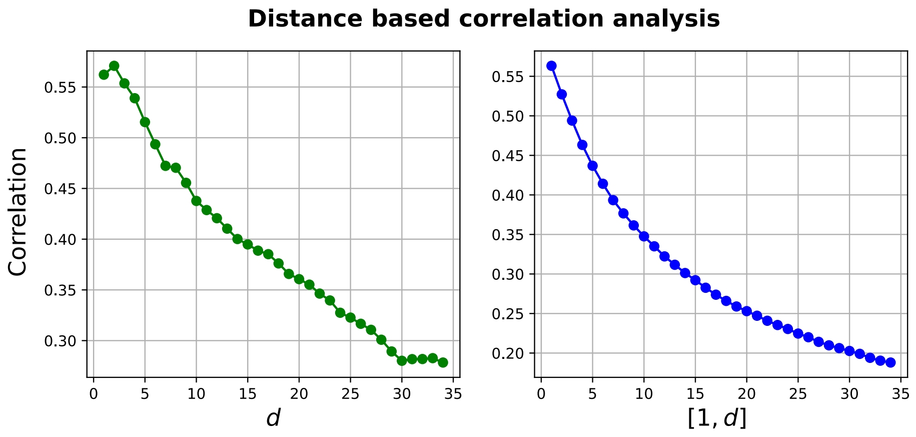


Figure 14: The plot on the left shows the correlation between transactions and page views at the exact distance d. The figure on the right shows the correlation between transactions and page views occurring in the distance range $[1, d]$.

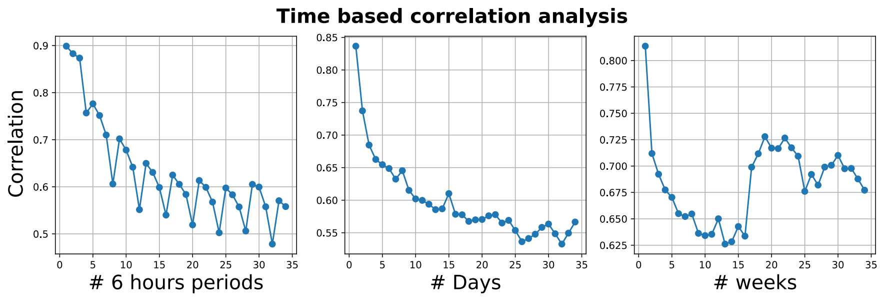


Figure 15: The plots show the correlation as a function of different periods. We can see that the correlation behaves periodically when we divide page views into 6 hours periods.

The time-dependent correlation analysis in section 3.4 showed that the correlation between page views and transactions decreases as the time difference between transactions and page view increase while taking a 6 hours period window frame showed periodic behavior, see figure 15.

4.5 Angle separation

We also measured the average separation angle for item vectors as depicted in section 3.5. The result is visualised in figure 16.

Finally, we measured the average angle separation based on time. The results are demonstrated in figure 17.

4.6 Optimised recall

To achieve the highest recall, we redid the transaction prediction task where we only took 1 recent transaction and 2 recent page views and monitored the recall to get 0.2154. The difference between this study and the one performed in section 3.7 is that we are not filtering the contexts but evaluating it on the whole data.

Finally, to investigate the robust time dependency between the event types, we thresholded the inclusion of page views by only allowing page views that occurred within the past 1500 hours before the purchase. We got 0.2155 as a recall.

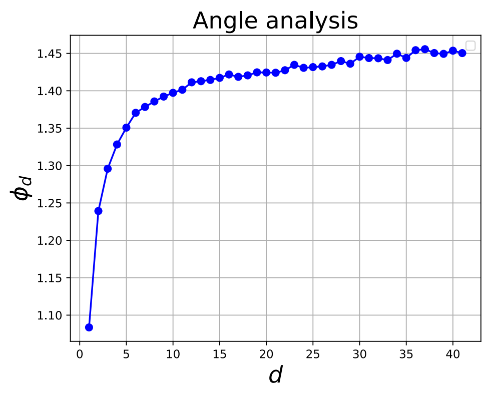


Figure 16: The figure shows the average separation angle ϕ_d for vector items associated with transactions and page views occurring d interactions before them.

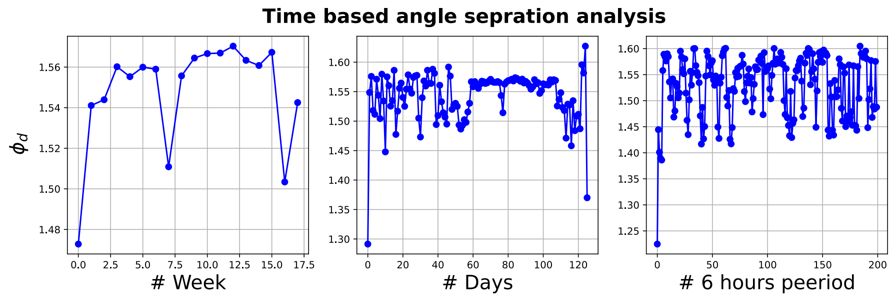


Figure 17: From left to right, the plots show the average separation angle ϕ_d for item vectors associated with transactions and page views occurring a certain number of weeks, days, and 6hours periods.

5 Discussion

5.1 Result analysis

Table 1 shows that training with only transactions gives the best recall values for all prediction tasks. When considering page views with weights incorporated, there is a slight increase in recall for all prediction tasks compared to not using weights. Additionally, considering page views in prediction improves recall for all three models.

From the distance base correlation analysis in figure 14, we deduced that the page views become, on average, rapidly irrelevant as we go further back in a user's sequence. We can further approve this observation by considering the logarithmic increase in average angle separation as we consider older page view events in Figure 16. These irrelevant page views in training decreased recall since it shifted the context vector $\mathbf{e_c}$ in directions away from the label.

The question is then: why does the performance degrade when considering page views during training but improve when using them for prediction? The answer to this question lies within the underlying dynamics of training and prediction.

During training, we infer the embedding space with respect to the given context vectors, which involves adjusting the embedding of the items to minimize the discrepancy between the predicted items and actual labels. If the page views we incorporate during the training phase are uncorrelated with the labels, we might introduce noise to the model's ability to predict labels.

On the other hand, the embedding space is constant during the prediction phase, and we are only deciding what items to involve in our context vectors which we calculate using equation 2 or equation 12 depending on whether we are incorporating weights. Hence, the improved performance during the prediction is due to the model's ability to leverage the established embedding space and use items relevant to the prediction task.

The results depicted in figure 11 demonstrate that adding page views improve results as long as the number of transactions is equal to or less than the number of page views in context arrays. We also observe the same behavior in figure 13, where incorporating more page views than transactions in the context arrays will harm the model's performance. Inspecting the correlation values in figure 14, we can state that each transaction is, on average, mostly correlated with their preceding page view before them. The same conclusion can also be derived from the angle separation in figure 16. Since the predictions are based on the average direction of previous interactions, considering more page views than transactions is equivalent to introducing vectors that point in directions far away from any other vectors in the context array, which shifts $\mathbf{e_c}$ in the wrong direction, reducing the model's recall.

Higher correlation values in Time-based correlation analysis in figure 15 compared to distance-based correlation in figure 14 indicate a strong time dependency between page view events and transactions. Meaning we would extract more information from page views if we add them based on how long before the transaction they have happened.

Another intriguing observation from the time-based correlation is the periodic behavior when taking six hours as time units, which could reflect that the users exhibit different behavior patterns during the day. One explanation is that the users may be more likely to purchase items during the day than late at night, and since some of the users in the data are companies, there is a work time dependency that we should account for.

Taking weeks as time windows in figure 15, we see the correlation decreasing followed by a sudden jump at week 15. The correlation then starts to oscillate around a mean value. Considering the data includes interior design items and the decline of recall as N_t increase in figure 12 could indicate the existence of the research and consideration Phase. Interior design purchases often involve careful planning, research, and consideration. Customers may explore various options and compare designs, colors, and styles before making a final decision. The significant increase in page views around 15 weeks before the purchase could indicate the start of this research and consideration phase. The very high correlation values observed for page views occurring only several hours before the purchase indicate the phase where the user has decided. Thus, the items the user views are similar to what he will eventually purchase.

Looking back at figure 11, we can observe the positive impact of incorporating the most recent page views on recall(Blue line in figure 11) and the positive effect of eliminating only the oldest transactions(red line in figure 11). However, As $Thresh$ increases, we start to take older page views, which degrades the model's ability to predict. On the other hand, increasing N_t also corresponds to eliminating more recent purchases. Hence, we expect the recall to start to decrease at some point. However, we do not observe such a turning point, and the recall continuously increases until we are only left with the latest purchase. The same behavior is depicted in figure 12 where higher N_t results in lower recall. Such observation is not intuitive since erasing information results in better predictions, which brings me to my next point.

All results and the discussion derived from it differ if we applied the model to other data. For instance, the relevance and usefulness of past purchase data in a recommendation model depend on the nature of the products and the behavior of the customers in that particular domain. In the field of interior design, the major factor that could contribute to the fastdiminishing relevancy of purchases is one-time or infrequent purchases. Once a customer makes a significant purchase, they might not need that item until they change their house, which is a long time and therefore does not reflect the user's current need.

Although the results are only valid on this particular data, we have introduced techniques and studies that we could apply to any data to extract useful information about customer behavior. We can observe the average length of seasonality, deduce the time and internal order dependency between event types and the performance of the model on users new to the platform, which are crucial for optimizing the hyperparameters of the model or any other model Siftlab wish to introduce in the future.

5.2 Future work

One major improvement approach would be applying the model to several different data and investigating whether the customer behavior stays the same within each e-commerce domain. If so, Siftlab can introduce an optimized version of its model depending on the client's e-commerce domain.

The second improvement is using the techniques in this work to implement and test time-decaying weights, both in the prediction training stages using separate functions for different event types. How these functions should look needs further thought, but the baseline is to introduce something aligned with the observed user behavior.

Ultimately, exploring the effect of including event types other than page views would be an interesting suggestion. Using the methods presented in this work, one should be able to enhance the model such that each event type contributes the most to perdition the next transaction.

6 Conclusion

The results indicate that we can expect minor improvements if we include page views in the prediction stage and exclude them during training. In addition, incorporating page views in the prediction stage can result in further enhancements. We have shown a strong time dependency between the page views and transactions for this particular data. Hence we can predict better if we use time instead of the sequence's internal order when choosing the page views we want to include in the contexts.

Furthermore, by commencing several empirical tests, we demonstrated that older purchases lose relevancy fast due to the data representing interior design items. We also observed periodic behavior by doing a time-based correlation analysis, which reflected the existence of seasonality in the data.

Our work here is the first step in the right direction for enhancing Siftlab's model. By utilizing the techniques I presented here, Siftlab can provide their clients with the most optimized version of their model based on the e-commerce domain and the event type included in the data. However, optimizing the model is very time-consuming, and whether the slight improvement is worth the effort is up to Siftlab to decide.

References

[1] Resnick, P., Varian, H. R. (1997). Recommender systems. Communications of the ACM, 40(3), 56-58.

[2] Ricci, F., Rokach, L., Shapira, B. (2015). Introduction to Recommender Systems Handbook. In F. Ricci, L. Rokach, B. Shapira (Eds.), Recommender Systems Handbook (2nd ed., Chapter 3, pp. 73). Springer.

[3] Khatwani, S., Chandak, M. B. (2016). Building Personalized and Non-Personalized Recommendation Systems. In 2016 International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT) (pp. 623-628). Pune, India.doi:10.1109/ICACDOT.2016.7877661.

[4] Adomavicius, G., Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering, 17(6), (pp. 734-749). doi:10.1109/TKDE.2005.99.

[5] Burke, R. (2007). Hybrid recommender systems: Survey and experiments. User Modeling and User-Adapted Interaction, 12(4), (pp.331-370). doi:10.1023/A:1021240730564.

[6] Adomavicius, G., Tuzhilin, A. (2008). Context-aware recommender systems. In Recommender Systems Handbook (pp. 217-253). Springer.doi:10.1007/978-0-387-85820-3_7

[7] Zheng, Y., Xiang, L., Yang, Y., Xing, E. P. (2018). DRN: A deep reinforcement learning framework for news recommendation. IACM. doi:10.1145/3219819.3220072.

[8] Pu, E., Chen, L. (2010). A User-Centric Evaluation Framework of Recommender Systems. In Proceedings of the ACM RecSys 2010 Workshop on User-Centric Evaluation of Recommender Systems and Their Interfaces (UCERSTI), Vol. 612. CEUR-WS.org, Barcelona, Spain.

[9] Su, X., Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. Advances in Artificial Intelligence, 2009, 4.doi:10.1155/2009/421425

[10] Pazzani, M. J., Billsus, D. (2007). Content-based recommendation systems. In The Adaptive Web (pp. 325-341). Springer. doi:10.1007/978-3-540-72079-9-11

[11] Ricci, F., Rokach, L., Shapira, B. (2015). Introduction to Recommender Systems Handbook. In F. Ricci, L. Rokach, B. Shapira (Eds.), Recommender Systems Handbook (2nd ed., Chapter 5). Springer.

[12] Nguyen, T. T., Hui, P. M., Harper, F. M., Terveen, L. G., Konstan, J. A.(2014). Exploring the filter bubble: The effect of using recommender systems on content diversity. Proceedings of the 23rd international conference on World wide web. 10.1145/2566486.2568012.

[13] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., Riedl, J. (1994). GroupLens: An open architecture for collaborative filtering of netnews. In Proceedings of the 1994 ACM conference on Computer-supported cooperative work (CSCW '94) (pp. 175-186). ACM.doi:10.1145/192844.192905.

[14] Sarwar, B., Karypis, G., Konstan, J., Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In Proceedings of the 10th International Conference on World Wide Web (WWW '01) (pp. 285-295). ACM.doi:10.1145/371920.372071.

[15] Saeed, M. (2021, 08 25). A Gentle Introduction To Sigmoid Function. Machine Learning Mastery. https://machinelearningmastery.com/a-gentle-introduction-to-sigmoid-function/

[16] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean.2013.Distributed Representations of Words and Phrases and their Compositionality.arXiv preprint arXiv:1310.4546

[17] RecSys '16: Proceedings of the 10th ACM Conference on Recommender SystemsSeptember 2016 Pages 191–198.https://doi.org/10.1145/2959100.2959190

[18] Mikolov, T., Chen, K., Corrado, G., Dean, J. (2013). Efficient estimation of word representations in vector space.arXiv:1301.3781

[19] Rendle, S., Freudenthaler, C., Gantner, Z., Schmidt-Thieme, L. (2009). BPR: Bayesian personalized ranking from implicit feedback. Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, 452-461.arXiv:11205.2618

[20] Nielsen, M. (2015). Neural networks and deep learning. Retrieved from neuralnetworksanddeeplearning.com

[21] Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep learning. MIT Press.

[22] Shumway, R. H., Stoffer, D. S. (2017). Time Series Analysis and Its Applications: With R Examples (4th ed.). Springer.