Optimizing Phone Selection: Personalized Recommendations Base On Historical User Rating

Nguyen Ho Bao Khanh
nhbkhanh21@vp.fitus.edu.vn

Nguyen Phan Minh Triet
npmtriet21@vp.fitus.edu.vn

Truong Hoang Kha
thkha21@vp.fitus.edu.vn

Vu Viet Hoang
vvhoang21@vp.fitus.edu.vn

Abstract

This paper proposes a personalized phone recommendation engine to optimize phone selection using Matrix Factorization in Collaborative Filtering. This technique mainly focuses on the similarities between the preferences and behaviors of different users. At first, we will apply Matrix Factorization to the rating dataset to identify all the latent factors that explain user-item interactions. We will then create a similarity matrix to select phones to recommend. The dataset is divided into two parts: the training set and the test set. After training we will use the test set to Evaluate our method. Our results show that the suggested method can provide more accurate and personalized recommendations, making it a valuable tool for anyone looking to buy a new phone. This paper's new contribution is applying matrix decomposition in collaborative filtering to recommend suitable products, addressing the issue of efficient and personalized product recommendations in e-commerce, particularly in the mobile phone sector. The results demonstrate the potential of this technique to provide accurate and personalized product recommendations in other applications.

1. Introduction

Currently, the world has been stepping on the path of digitization, most of the countries in the world are racing with each other along the path of the digital economy and with it, a series of modern and most advanced technologies have come out. Accordingly, smartphones were born to help improve people's lives, from making calls and sending messages to browsing the internet and using a variety of applications, smartphones give people use endless possibilities.So the telephony industry has always been very competitive, leading to companies Smartphone manufacturers always improve products every day with many different models, prices and quality depending on the needs of users. With various choises and constant changes in price, it's always difficult to find yourself a suitable phone.

Find your perfect phone: compare prices and features to meet your needs. Now that, there are many different phone models with many advantages and disadvantages, it will be difficult for consumers to choose the product they want. There was a series of experimental reports from experts about the way consumers buy phones[1] [2] [3]. And also have a lot of studies whenever customer buying behavior depends on features[4] or brand [5] [6].

All of the above tests have the same assessment that the number of customers around the world today are choosing phone products according to criteria such as price, brand, camera quality, quality display, and battery life. There are many websites now that can provide comprehensive reviews and feature ratings of various phones such as Consumer Reports, Tom's Guide, TechCrunch, etc. However, for reference, all different phone models will waste a lot of time.

That's why we have created a tool to help you make the right choice for you quickly and efficiently, by analyzing the features and prices of popular smartphone brands, we provide valuable insights into consumers' preferences when choosing their ideal smartphone. With the rise of the smartphone market, this tool will become a useful tool for users and can help promote the development of the mobile phone industry. Therefore, this study is of considerable importance for the mobile phone industry and may also provide a model for future similar studies.

We used a method of comparing and analyzing the features and prices of different mobile phones along with a quantitative research approach that involved gathering and analyzing user data on their preferences and behaviors using recommendation systems. First, we collect hardware data of a variety of phones, their prices, and features from a variety of sources. Then, clean the collected data to ensure accuracy. The performance will be evaluated through actual tests that have been published and ranked through the Antutu benchmark application to measure and compare. Next is to use data analysis tools to analyze the results. This tool will include all the information database of different phones in the market such as specifications of phone models, prices, and consumer reviews. And in order to improve the accuracy of the recommendation system, we conducted a survey with a sample of users who have had experience with different recommender systems in different contexts and chose came up with the Matrix Factorization algorithm, which splits the interaction matrix between the user and the item into two low-ranking matrices to predict user preferences recommendations for users. By applying this technique, we aimed to enhance the accuracy of our recommendation system and provide more tailored recommendations to users. The web interface will take over the user's search needs. After that, algorithms, comparison calculations will be run in turn to filter the available phone database on the system and sort phone models, and give suggestions to users. These proposals are evaluated based on many factors. Finally, use a clustering algorithm to group similar phones based on features like camera quality, battery life, and price to help users make informed decisions.

One of the limitations of this study is that it will be limited to a specific time period because of variability. Market dynamics as well as prices are inevitable, prices and features of mobile phones may vary in many countries and regions around the world, this study will be limited to one country certain and may not cover many countries around the world, focusing on a certain phone model is not representative of all other models on the market, so unexpected product failures may occur. With the breakthrough in modern technology, advanced inventions, new features, and new initiatives will always be created that affect the proposed system that is currently unpredictable.. such as folding phone models, roll phones, and ultra-thin phones,...

Although there are many such limitations, if in the future we apply some solutions such as increasing the range, and the number of samples and analyzing many other factors such as brand, and operating system, ... to provide a more complete and accurate analysis, we will always monitor and continuously update the current trends in market to easily give advice to customers, then this research will bring great benefits to modern industry.

In conclusion, this tool will help users easily find the right phone based on price and features, is a useful tool to help consumers choose a phone that suits their needs and budget. This research will play an important role in the field of consumer electricity and it will bring a huge benefit to both customers and consumers.

2. Related Work

The more store and brands come out, the harder it is to retain customers. According to a study by McKinsey & Company, Gen Z (those born between 1997 and 2012) are not loyal to any particular brand and are more willing to try new products and services.[7] This has led to increasing demand for personalized recommendation systems. According to an Accenture report, it was " 91% of consumers are more likely to shop with brands who recognize, remember, and provide relevant offers and recommendations."[8] . In the context of mobile phones, recommendation systems can help customer to find the most suitable phone, and avoid wasting money. It can also help retain customers and increase revenue for business owners.

There was 4 method to make a phone personalized recommendation system including Content-Base Filtering, Collaborative Filtering, Content-Aware, and Hybrids Side information. In this work, we use matrix factorization in Collaborative filtering using pure SVD combine with KNN as a baseline and we will explain why.

Let's talk about Content-Base Filtering first. It is a technique using the attribute of phone and customer preferences. It can give recommendations very fast, by counting similarities between phones and giving recommendations. But if the customer didn't provide enough information or there was lacking information in phone attributes, the content-based methods won't work well. Meanwhile, Collaborative Filtering with Matrix Factorization mainly focuses on similarities between the preferences and behaviors of different users which make it more accurate, and more diverse than Content-Base Filtering, and because it dispenses the user rating matrix into a smaller matrix, the lacking of information won't be essential anymore. In a study by Adomavicius and Tuzhilin (2005)[9], two methods of Collaborative Filtering (CF) and Content-Based Filtering (CBF) were compared. The results show that CF gives better suggestions than CBF in the case of a large number of products and users. However, CF is also problematic when products are underrated, so additional information from external sources is needed to improve recommendation efficiency.

The second one is User-based and Item-based. They are two traditional and simple methodologies to build recommender systems. However, both of these methods may be affected by misjudgments or be limited by data shortages. In a study by Koren and Bell (2009)[10] compared Collaborative Filtering methods based on User-based and Item-based evaluation. The results show that the User-based method is more effective when the number of users is more than the number of products, while the Item-based method is more effective when the number of products is more than the number of users. Therefore, new approaches such as Matrix Factorization have been proposed and developed in recent studies. With large and sparse data sets, this method can achieve higher accuracy than User-based and Item-based. However, this method may require more computation time to perform the training and prediction.

And the third one is Deep learning The choice of the appropriate method revolves around the specific requirements of the problem and the availability of data. Because of the feature interactions in solving this particular problem, prioritizing these factors will replace extracting complex patterns from unstructured data or processing large-scale datasets. As a result, the use of matrix factorization analysis is necessary.

3. Methodologies

3.1. Baseline

At first, we choose Content-Base Filtering as the baseline. But what is Content-Base Filtering? It is the system that takes input data[1] as an Items-Attribute matrix, counting the similarity between each item and based on the user's historical information to make recommendations. So in our work, at first, the user will input all the features of their dream phone. Our system will take sentences like "I like an average price and average size " and then analyze it using nltk library. At first, I will use word_tokenize() function to break the sentence into individual words, and use pos_tag() function to assign a grammar tag (e.g. noun, verb, adjective) to each word based on its context in the sentence. And then look for noun + is + adj pair and adj + noun pair. After that, I will use the noun as key and the adj as value to create a Python dictionary (e.g. the short battery $\rightarrow$ e[battery] = short). After that, I will assign the value for each adj word. It means if the user wants some attribute that is average, the system will find the average value of the attribute, and replace the adj with the value. We then use Euclidean distance to calculate the similarity between their dream phone and all the phone in the dataset. The mathematical formula of it was:

$$E(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$$

In our work, the xi is the value of that attribute in the dataset and yi is the value of the same attribute but in user data. Last but not least, we sort the dataset base on similarity in ascending order. But it wasn't enough because it is non-personalized recommendations which means all users receive the same recommendations. It also didn't have diversity recommendations and can only evaluate by the rating of users after they do as the recommendations said. But, how could we know if the people will do as the recommendations said or how many phones must be bought to test the system? It makes the system hard to evaluate than others. That is the reason why we do some more research and finally selected the algorithms which are easier to evaluate, each user will receive a different recommendation. That algorithm is Collaborative Filtering.

[1]https://www.kaggle.com/datasets/anas123siddiqui/mobiles?fbclid=IwAR1LwZvB997RvLaBAQR59uc7UAuVI7gwL3PkO31VvkHKw-_nVwncOM0v3q8

3.2. Main algorithms

3.2.1 Introduction

We choose Collaborative Filtering as our main algorithm. It is an algorithm that takes input data as the user's rating matrix and it will make recommendations for one user based on a group of users. In Collaborative Filtering fields, we chose

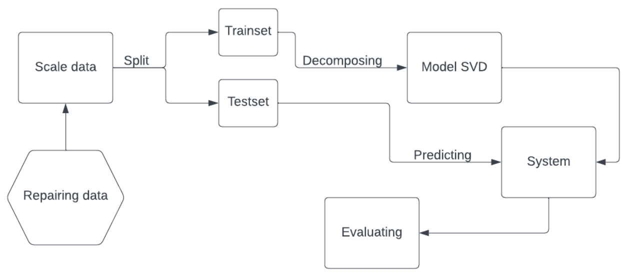

Figure 1: Evaluate system

Matrix Factorization. It is like the combination of Collaborative Filtering and Content-Base filtering. The Matrix Factorization idea is borrowed from Content-Base Filtering by decomposing the user-item rating matrix into two lower-dimensional matrices, and the connection of the two matrices is a latent factor which is kinda same as the attribute in Content-Base Filtering. It decomposes the user-item rating matrix into two small matrices representing users and items. But how can it be decomposed? There are a lot of algorithms to help decompose data like SVD, SVD++, Funk SVD, and Asymmetric SVD. And this SVD "family" was working very well in The Netflix Prize [10]. It is the competition launched by Netflix in 2006, aimed at improving the accuracy of its movie recommendation system. Funk SVD, Asymmetric SVD, and SVD++ are all algorithms using ALS to find the latent factor. ALS is an iterative algorithm that will separate the user-rating matrix into matrices X and Y. It'll fill random data into X and Y, after that keep X and fix Y, keep Y and fix X until two matrices multiply each other approximately the user-rating matrix. At first, the latent factor will be one and increased until the best X, Y matrix is found. Meanwhile, Pure SVD is a batch algorithm that computes the SVD of a matrix all at once, without the need for iterative updates or optimization steps. It does not explicitly specify a number of latent factors like other variants of SVD. However, the concept of "latent factor" can still be applied in pure SVD, as the algorithm efficiently learns hidden features or dimensions to capture underlying patterns in the data. These features can be considered similar to "latent factors" learned by other matrix analysis techniques, although they are not so clearly labeled in the context of pure SVD. The reason why Pure SVD is a useful algorithm for small data because it computes only one when using ALS to compute small data can cause wasting of time and effort since it recomputes the X and Y over and over again. And our dataset is small data so we chose Pure SVD in our systems. After choosing the right algorithms, we need to evaluate the performance of these systems Figure 1. In our systems, we choose RMSE mathematical formula which is the error metric to evaluate and it stands for Root Mean Square Error. Return to our

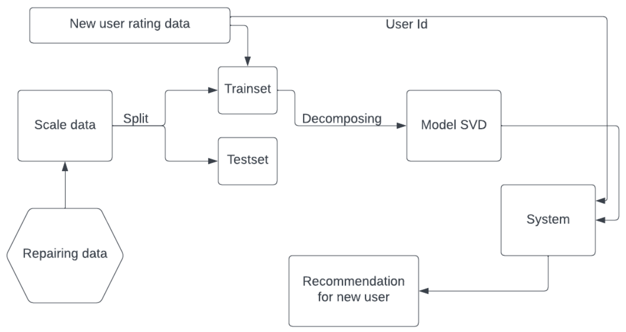

Figure 2: Recommandation system

system, in summary, we'll prepare data, train the system, evaluate the system, and after that, we start to recommend things Figure 2 2. But when recommending to new users, we need to add the user's historical information to the training set and recalculate to make the recommendation.

3.2.2 A Closer Look at Our Project

Our system includes 2 datasets is : cellphonesratings.csv and cellphonesdata.csv. The first dataset surveys 258 users' likelihood to purchase 10 cell phones at given prices, while the latter includes data on 34 popular cell phones in the US in 2022, with 13 features and aggregated prices from Amazon and Best-Buy. The first thing we need to do is prepare data. We'll take cellphones ratings as input, but this dataset has some ratings greater than 10, so we need to scale the rating to make it range from 1 to 10. In order to do that, we will the Reader class. This class has 5 parameters but we will only input the rating_scale parameters. If well don't define it, the library assumes a default rating scale of (1, 5), meaning that ratings must be integers between 1 and 5. So we just use it and defined its rating scale of(1,10). After scale rating, we will transfer the original dataset to the data in which the rating is scaled. The dataset object is being used in this case to load the df DataFrame into the Dataset object, using the load_from_df() method with the reader object created earlier as a parameter to parse the data. When our data is ready, we use the 'train_test_split' function from sklearn.model_selection module to split a dataset into a training set and a test set. This function takes several arguments but in our works, we only use 3 main arguments: data, test_size, and random_state. The first two are quite understandable. The data is the whole data set, the test_size is the size of the test set and in our work, the test size will be 20% of the whole dataset. But the random_state is more special. In order to avoid data bias, the splitting function will shuffle the whole dataset. If we don't use the random_state arguments, the function will shuffle the dataset using the computer's internal clock. It means each time you execute the system, it will shuffle differently. As a result, the reproduction process in the system is inconsistent and not reproducible. Consistent and reproducible make it easier to evaluate systems against each other and it is quite crucial in Machine Learning. So to avoid those things, we use the random_state arguments. It is the random seed for the random number generator which help shuffle the dataset in a fixed way. It means each time the system is executed, it will be shuffled the same. But the split part will split differently in each execution so each answer will be different even using the same random_state. An after the shuffle part is the split part. In the split part, this function will split randomly but ensure that each user is included in either the training set or the testing set, but not both, to avoid data leakage. After preparing the data, we'll use the train system by SVD algorithms in the Surprise library. So let us explain in more detail about it. This SVD algorithm will decompose the train set to 3 small matrices that are U, $\sum$, V^T The formula is

$$A \approx U \cdot \Sigma \cdot V^T$$

Where A is the input data matrix, U is left singular vector , V^T is right singular vector, $\sum$ is singular values. It can be written more succinctly by

$$\begin{aligned} R &\approx Q \cdot P^T, \\ R &= A, \\ Q &= U, \\ P^T &= \Sigma \cdot V^T. \end{aligned}$$

And this is an unbiased version of this algorithm. So what are data bias and data unbias? Each people will rate the item differently, some rate the item they hate most 0 stars meanwhile others may rate 5 stars because they don't want to hurt the person who produced it. So biased version will take into account the bias of each user which means the individual preferences and behaviors of each user and item. Data unbias, on the other hand, refers to using a fixed global bias for all users and items, which does not take into account individual preferences and may result in less accurate recommendations. So the biased version of this algorithm will have a formula

$$R'_{ui} = \mu + b_u + b_i + q_i^T p_u$$

- Rui': The predicted rating for user u and item i
- μ: The overall average rating for all items in the dataset.
- bu: The user bias term represents the individual preferences and behaviors of this user. If $bu > 0$, they tend to give higher ratings to items than the overall average rating. On the contrary. If $bu < 0$ they tend to give lower ratings to items.
- bi: The item bias term represents how much the item i is bias toward other item. If $bi > 0$, the item tends to be rated higher than the overall average, and if on the contrary. If $bu < 0$, the item tends to be rated lower than the overall average.
- qi: The item latent factor vector for item i.
- pu: The user latent factor vector for user u.
- T: The transpose operator.

And the SVD algorithms in the Surprise library is biased version. The unbiased version can be achieved by setting the biased parameter to False. And of course, we use the biased version.

Then, we will continue to the next part which is the prediction part, and we also use the test method from the algorithm base class in the Surprise library. It will have 2 parameters which are test set and verbose. The first parameter will take our test set and test the algorithms. The second is used to print details for each prediction which is false as default. And this method will return a list of prediction objects that contains all the estimated ratings, true ratings, the id of the user, and the id of the item. When we complete training, we need to find a way to evaluate and optimize our system. After considering a lot of metrics and methods, we choose RMSE which is an error metric. The formula of this metric

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(true_i - est_i)^2}{n}}$$

Where true is the true value of the rating, est is the estimated rating and n is the number of samples in the dataset. And in our works, we use accuracy.rmse() function to calculate the RMSE. It will take the prediction object as the parameter. And the better the recommendation system, the closer the estimated rating is to the real rating. So our goal is to minimize the distance between the true and estimate rating. The true rating symbol is The true rating symbol is r_{ui} and the estimated rating symbol is $\hat{r}_{ui}$. which is equal to $\mu + b_u + b_i + q_i^\top p_u$. But all the μ, b_u, b_i, and T operand are constants, so in the minimized term, we can skip them and focus on minimizing the q and p matrix. So the formula to minimize the metric

$$\sum_{(u,i)\in K} (r_{ui} - q_i^T p_u)^2$$

But when minimizing the system, we should be careful. Because the closer the estimated rating is to the real rating in the training and test set didn't mean the estimated rating for the new user will be closer to their real feeling about that item. But how could that happen? In our data, there will always have some "noise" which is the false value. If the estimate fits the true rating very well, it means it fits with all the "noise" and because of that, the system will be poor performance in real situations and create "over-fitting". To avoid "over-fitting", we can use regularization techniques such as adding a penalty term to the loss function. So we add to the loss function the two norms of the similarity matrix which is

$$\lambda_1 \sum_{u=1}^{U} |p_u|^2$$

Which is a characteristic of the Ridge Regression regularization technique. The overall of the formula is

$$\sum_{u,i\in K} \left(r_{ui} - q_i^T p_u\right)^2 + \lambda_1 \sum_{u=1}^{m} ||p_u||^2$$

If we don't use the regularization techniques, the system will be "over-fitting" but if the lamda is so big, the system will ignore the q_i and q_u matrix which is the similarity matrix, and instead rely solely on the item features or other input data to make predictions. It maybe is good if the similarity between the 2 items is not well-defined and the items feature is play an important role. But usually, the similarity between 2 items is more important than the item's feature. So we need to find a suitable lamda that balances minimization of error and reduction of over-fitting We can reduce even more risk of over-fitting by adding one more norm of the similarity matrix which is $\lambda_2 \sum(q_i^2)$ The part of adding one more norm to the formula is a characteristic of Elastic Net. The overall formula will be

$$\sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda_1 \sum_{u} |p_u|^2 + \lambda_2 \sum_{i} |q_i|^2$$

The equation can be simplified like this: $\min$ factors "error" $+\ \lambda$ "length". So in other to find matrix q_i and p_u that will have the equation we will need to use Stochastic gradient descent[11] which is the update of Gradient descent. Descending gradients start with an initial set of parameters and calculate the gradient of the cost function toward these parameters. It then updates the parameters by subtracting part of the gradient from the current values until the function is close or equal to zero. The slope at the starting point will be steeper, but as new parameters are generated, the steepness should gradually reduce until it reaches the lowest point on the curve, known as the point of convergence. But stepwise subtraction will slow down the calculated gradient. So the SGD was born to improve that. SGD will gradients start with an initial set of parameters just like GD but after that, it will randomly select a mini-batch of training data from the dataset and subtract the gradient of that mini-batch. And it will repeat steps until convergence or a predefined number of iterations is reached. In short, based on what we just explained, SGB will compute faster than GD, so we use SGD in our work. In conclusion, to optimize our system, we need to optimize SVD algorithms using SGD. Luckily, the Surprise Library supported us to do that. If we use the default SVD algorithms in the Surprise library, the number of iterations of the SGD procedure will 20. So, based on what I just analyzed in our work, we will use 4 parameters which are: _factors, n_epochs, lr_all and reg_all to optimize our system where: n_factors: the number of factors which is 100 as default

- n_epochs: the number of iterations of the SGD procedure which is 20 as default.
- lr_all: The learning rate for all parameters which is 0.005 as default.
- reg_all: The regularization term for all parameters which is 0.02 as default.

Each dataset will have different patterns so it is often necessary to experiment with different values to find the best model. That is why we must modify the n_factors.

After all the preparation part, evaluation part, we start to make a real recommendation for users that are already in the dataset. But before making a recommendation, we need to map the id of the phone in the user-rating matrix to the model which was in the item-attribute matrix. Because if we recommend the phone id, the user doesn't know what phone is it. We will use the dict and the zip function to create a Python dictionary named mapping. The zip() function creates tuples of the corresponding elements from the two lists meanwhile the dict() function then converts these tuples into key-value pairs in a dictionary. After this part, we start to make a recommendation by finding the user-item pair only includes items that the user has not yet rated using the filter function. It takes 2 arguments including function and sequence. The function will take each element in the sequence, if that element meets the condition in the function, it will be returned. After that, we use the test method to test all the items that users haven't rated yet. In our work, we will input 2 parameters in this method which are "key=operator.itemgetter(3)" to sort in the predicted rating and "reverse=True" to sort the list in descending order. After calculating, we will print the model and the estimated rating. To print the model, we will use a Python dictionary named mapping that we have just created and input the id of the phone to it, which is in column 1. After that, we will round to 3 digits after the ",".

Now, we finally can recommendation in the old data. But all we want is the model that the user can use, which means, we must take the user's historical rating in the phone, put it in the trainset, and use the SVD algorithms to calculate the rating for that user. In order to do that, firstly, we print all the models from the item-attribute matrix. And then, the user will find the id of the phone they already buy, and input it into the system with their rating. The phone id, rating, and user id which is default assigned 259 in our systems, will all be stored in the tuple variable. The trainset variable is a complex object which is not designed to be modified directly by adding or removing individual ratings. So, in order to add a rating, we use the build_testset() method which will return a list of tuples and in our work, it is stored on the n_testset variable. We then add the new data to that variable using the append() method which will append the new data to the end of the list. After adding, we will scale the rating again from 1 to 10 and finally use the SVD algorithms to calculate their rating, and then make recommendations.

Afterward, the main algorithm of our work is pure SVD combined with SGB[12]. It'll recommend a phone better than the first work as it is a personalized recommendation and can be easily evaluated using the RMSE.

4. Experiments

4.1. The experiments were conducted on a machine with the following configuration:

- Processor: Intel Core i5 12900F.
- Memory: 8 GB DDR4 RAM.
- Graphics Card: NVIDIA GeForce RTX 3060 Ti, 8GB GDDR6 VRAM.
- Operating System: Windows 11

4.2. Datasets

We evaluated the proposed model on two datasets, namely cellphonesratings.csv and cellphonesdata.csv[2].

Cellphonesratings.csv: This dataset contains a cellphone rating survey of 258 users. Each participant was given 10 random cell phones and was asked to indicate the likelihood that they would buy each cell phone at a given price, on a scale of 1 (very unlikely) to 10 (very likely).

[2]https://www.kaggle.com/code/stpeteishii/cellphone-recommendation-usingsurprise/input

Cellphonesdata.csv: This dataset contains data about the most popular cell phones in the United States in 2022. The data for each cell phone includes the most notable features such as performance rating (AnTuTu), memory size, camera resolution, battery size, screen size, release date, and more. The price per phone is aggregated from Amazon and Best-Buy (as of August 22). Overall, the dataset consists of 34 cell phones with 13 features.

4.3. Data preprocessing

However, the current dataset contains some error values, such as missing values, invalid values, or duplicate values. Missing values can cause errors in the model or make the model unstable. So to solve this problem, the first step is to read the input dataset with pandas, skip lines containing errors. In addition,we randomize the lines from the data set, which minimizes the chance of errors.

After reading the input data, the next step is to preprocess the data before it is fed into the model. We have to scale the rating to [1-10] instead of [1-18] in the datasets

During data processing, error and missing values are handled by replacing them with the average value of the corresponding column. Additionally, NaN (Not a Number) objects are disposed of by replacing them with the value 0.

After preprocessing the data, we divide the dataset into two sets of train and test with the ratio 80-20. The train set is used to train the model, and the test set is used to evaluate the model 1.

	Total	Trainset	Testset
Total number of rating	990	792	198
Total number of user	99	99	86
Total number of cellphone	33	33	33
Number of cellphone		33	33
Number of users		99	86
Number of ratings		792	198

Table 1: Number of data in dataset, trainset and testset

4.4. Result

To evaluate the effectiveness of the tool, we conducted a test comparing our proposed method with other existing methods. In our tests, we used a dataset of popular smartphones as well as customer reviews. We randomly split the data set into training and test sets in a ratio of 80:20. We used the training set to train our machine learning models, then used the test set to evaluate the performance of the tool.

We evaluated our tool against the RMS[13]. These metrics help us understand how effective our tool is in recommending the right phone to users based on their preferences.

RMSE is a widely used metric in system recommendations, and it provides a good measure of a system's overall performance by measuring the average difference between predicted and actual ratings.

However, the choice of metrics depends on the usability of the tool and the goals of the proposed system. Because RMSE has:

First, it provides a clear explanation of the rating scale, which is important in many real-world applications. Second, the RMSE is sensitive to outliers, which can be important in some cases. Third, easy to understand and interpret, even for non-technical stakeholders. So RSME is needed instead of Recall or F1-score

Our results with recommendations based on old data 2 and recommendations based on the new user input with the Number of phones you already buy and rate it = 2, the cellphone id = 1, user rating = 5, cellphone id = 0, user rating = 10 3 show that our tool can recommend the right phone to the user with great accuracy. In particular, our tool is superior to other existing methods, such as content-based system

User Id	Recommendation	Estimated Rating
0	iPhone SE (2022)	7.107
	iPhone 13	6.653
	iPhone 13 Pro Max	6.472
	Pixel 6	5.851
	12 Pro	5.796
1	Galaxy S22	7.81
	iPhone 13 Mini	7.809
	iPhone 13	7.744
	Find X5 Pro	7.623
	iPhone SE (2022)	7.544
6	iPhone 13 Pro	8.02
	Moto G Play (2021)	7.323
	12 Pro	7.313
	Redmi Note 11	7.295
	iPhone 13 Pro Max	7.188

Table 2: New recommendation base on old data

User Id	Recommendation	Estimated Rating
259	X80 Pro	8.208
	11T Pro	8.115
	12 Pro	7.926
	Galaxy A53	7.871
	iPhone 13 Pro	7.746

Table 3: New recommendation based on new user data

Besides that, we also compare RMSE with MAE (Mean Absolute Error) and MSE (Mean Squared Error)

RMSE and MAE are often used to evaluate the user rating prediction problem in recommendation systems and used to evaluate the accuracy of the user's product rating predictions. So using these three measures is suitable for the problem we are doing, there are many articles about the comparison between the two above measurements [14] [15] [16], but between these two measures it will have The specific differences are as follows:

- RMSE (Mean Square Error) measures the deviation between predicted and actual values on a scale from 0 to infinity. A lower RMSE indicates greater accuracy. It is calculated as the average squared error between the prediction and the actual value.
- MAE (Mean Absolute Error) MAE calculates the mean of the absolute error between the prediction and the actual value on a scale of 0 to infinity. A lower MAE value indicates a higher accuracy of the proposed algorithm.
- MSE (Mean Squared Error) is a widely used error measurement in machine learning that evaluates the quality of a predictive model by calculating the average squared error between predicted and actual values. A smaller MSE indicates the better predictive quality of the model.

Based on the table 4 of test results of RMSE and MAE, we can see that: with RMSE values ranging from about 2.1 to 2.1, MAE values ranging from about 1.6 to 1.7, and MSE values ranging from about 4.2 to 4.5. RMSE, MSE and MAE both show a difference in the performance evaluation of algorithms, however RMSE shows a larger difference than MAE and MSE. This shows that RMSE is a better criterion to evaluate models in the problem of phone selection optimization with coefficient analysis matrix

	RMSE	MSE	MAE
Run times	2.1564	4.2098	1.6778
Run times	2.1116	4.5187	1.7032
Run times	2.1563	4.5389	1.6071
Run times	2.1218	4.5297	1.6527
Run times	2.1566	4.3465	1.6945

Table 4: Runtimes of metric RMSE, MSE and MAE

The experiment showed that the recommendation algorithm using RMSE performed better than the baseline algorithm using MAE, and also outperformed the algorithm using MSE. RMSE was chosen as the evaluation metric because it reflects the accuracy of the model and the importance of differences between actual and predicted values in the context of phone purchase decisions. Additionally, RMSE is more flexible and easy to calculate than other metrics. However, RMSE has limitations such as bias towards popular items and lack of interpretability for users. Overall, RMSE is suitable for evaluating recommendation algorithms, but its limitations should be taken into account

5. Conclusion and Future Works

The recommendation engine provides valuable insights to consumers when choosing the right smartphone by analyzing users' usage habits for popular phone models. It will ask the user to enter the rating of the used phone and offer suggested phone models. Although the research is limited in terms of input data processing, it cannot completely replace store consultation. But in the future, there may be some methods to help extend the capabilities of the tool:

First, adding a natural language processing (NLP) model will allow the tool to collect detailed requests for preferred phone models and use this information to make recommendations.

Second, expand the database to include more phone models and brands in the world. It will monitor the market continuously to keep the tool up to date with changes in pricing and features.

Third, analyze other factors such as brand reputation, and customer service to give a more general view of each phone model. This is achieved by combining more variables.

In summary, the Recommended Tool is a useful tool that will play an important role in the consumer electronics sector and will benefit both customers and manufacturers.

References

[1] Mesay Sata. Factors affecting consumer buying behavior of mobile phone devices. *Mediterranean Journal of Social Sciences*, 4(12):103, 2013.

[2] Safiek Mokhlis and Azizul Yadi Yaakop. Consumer choice criteria in mobile phone selection: An investigation of malaysian university students. *International Review of Social Sciences and Humanities*, 2(2):203–212, 2012.

[3] CS Craig and James F Engel. Consumer decision-making: On the importance of price. *ACR Special Volumes*, 1971.

[4] Owusu Alfred. Influences of price and quality on consumer purchase of mobile phone in the kumasi metropolis in ghana a comparative study. *European Journal of Business and Management*, 5(1):179–198, 2013.

[5] Nguyen Van Thuy, Nguyen Thi Ngoc Anh, and Ngo Thi Xuan Binh. Impact of brand equity on consumer purchase decision: A case study of mobile retailer in hochiminh city, vietnam. *Journal of Eastern European and Central Asian Research (JEECAR)*, 9(2):229–239, 2022.

[6] Xiaoling Guo, Andy Wei Hao, and Xiaoyan Shang. Consumer perceptions of brand functions: an empirical study in china. *Journal of consumer marketing*, 28(4):269–279, 2011.

[7] Tracy Francis and Fernanda Hoefel. True gen': Generation z and its implications for companies. *McKinsey & Company*, 12, 2018.

[8] Personalization pulse check by accenture published in 2018. `https://www.`

accenture.com/_acnmedia/pdf-83/accenture-pulse-check-infographic.pdf.

[9] Gediminas Adomavicius and Alexander Tuzhilin. 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):734–749, 2005.

[10] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[11] Gradient descent. https://www.ibm.com/topics/gradient-descent#:~:text=Gradient%20descent%20is%20an%20optimization,each%20iteration%20of%20parameter%20updates.

[12] Gradient descent. http://snap.stanford.edu/class/cs246-2015/slides/07-recsys1.pdf.

[13] Deng-Neng Chen, Paul Jen-Hwa Hu, Ya-Ru Kuo, and Ting-Peng Liang. A web-based personalized recommendation system for mobile phone selection: Design, implementation, and evaluation. *Expert Systems with Applications*, 37(12):8201–8210, 2010.

[14] Timothy O Hodson. Root-mean-square error (rmse) or mean absolute error (mae): when to use them or not. *Geoscientific Model Development*, 15(14):5481–5487, 2022.

[15] Tianfeng Chai and Roland R Draxler. Root mean square error (rmse) or mean absolute error (mae). *Geoscientific model development discussions*, 7(1):1525–1534, 2014.

[16] Dulakshi Santhusitha Kumari Karunasingha. Root mean square error or mean absolute error? use their ratio as well. *Information Sciences*, 585:609–629, 2022.