FeedMe: AI Food Recommender System

Andrew Bejjani
Electrical and Computer Engineering
American University of Beirut
Beirut, Lebanon
aab68@mail.aub.edu

George Kanaan
Electrical and Computer Engineering
American University of Beirut
Beirut, Lebanon
gek12@mail.aub.edu

Hadil Abdel Samad
Electrical and Computer Engineering
American University of Beirut
Beirut, Lebanon
hma142@mail.aub.edu

Mohamad Al Katrangi
Electrical and Computer Engineering
American University of Beirut
Beirut, Lebanon
maa342@mail.aub.edu

Mariette Awad
Electrical and Computer Engineering
American University of Beirut
Beirut, Lebanon
ma162@aub.edu.lb

***Abstract*—Have you ever felt undecisive about what to cook? Can you imagine that with just one click you will be guided to make a choice? In this paper, we will propose using machine learning a food recommender system that is able to suggest various food recipes based on ingredients and users' preferences.**

***Keywords*—food recommendation, recommender system, ingredients, rating, food, content-based filtering, similarity.**

I. INTRODUCTION

People are usually uncertain about their choice of food. While trying to cook something at home, you will always feel hesitant about what to cook, and which kind of recipe you should go for. According to Gorospe [1], selecting the right recipe by choosing a list of ingredients is very difficult for a beginner cook, and it can be a problem even for experts. In addition to that, some available ingredients can be used to prepare recipes that the cook may not like. In this situation, knowing the preference of the cook is crucial to find the best recipe needed.

In this project, we will implement using machine learning a Recommender System to suggest the optimal recipe given needed ingredients and users' preferences.

II. RELATED WORK

To be able to achieve our goal, we referred to several papers that targeted this topic. To illustrate, a food recommender system was previously designed based on weighted ingredients, body mass index and allergies. It was implemented using the Random Forest algorithm which was able to attain a precision of 97.7381% [1]. Another paper was published, that discussed the link between ingredients and their respective cuisines. The authors were able to investigate different machine learning classifiers like Naïve Bayes, Neural Networks and Support Vector Machines. After testing, they concluded that the Support Vector Machines had the best outcome among the rest [2]. A third group tackled the problem using Keyword-based Vector Space Model, where they implemented an application that proposes recipe based on users' profiles and satisfactory features [3]. In another article, they tackled a movie recommendation system where the type of genres that the user might choose to watch were taken into consideration. To do this, a content-based filtering strategy based on genre correlation has been used [4]. In addition, another essay outlined the content-based method, its benefits, and drawbacks, and how to use it to provide movie recommendations based on this concept. The report also detailed all other approaches, their methods, and their restrictions [5]. On that note, a similar project was launched which worked on improving content-based filtering algorithm using Artificial Bee Colony optimization. It described what content-based filtering is and how a similarity metric is attained based on correlation between what the objects are and what the user inputs as profile [6]. In that way, we were able to come up with a possible implementation that builds on what we have researched about. For our project, we will be utilizing several metrics for testing like Neural Networks and content-based filtering methods.

III. PROJECT PLAN

In this project, we will use a data set called "Food.com Recipes and Interactions" that consists of more than 180,000 recipes collected throughout 18 years of uploads [7]. We're going to perform preprocessing and feature engineering techniques to extract the needed features for our project due to the surplus amount of unused data. After creating a new data frame, through merging and filtering the data we want, we will split our data to train and test sets. After training the data we will use various metrics to test our model. We will experiment with Neural Networks and similarity metrics. In this way, we would have achieved a food recommender system that learns from previous ratings on food recipes and predicts what new recipes the user might like.

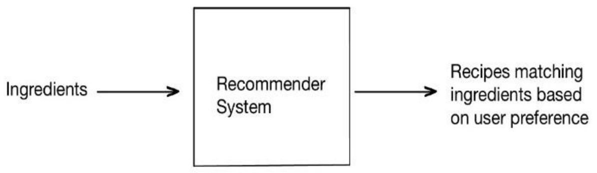


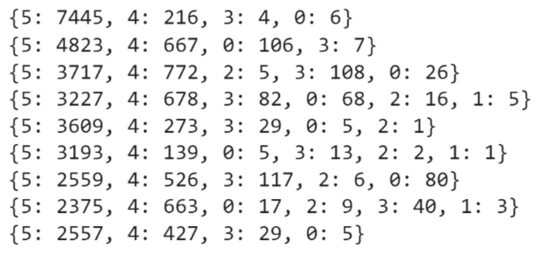
Fig. 1. Diagram Representation of Model's Inputs and Outputs

The above figure represents an overview of our model. Our inputs will be the ingredients, and once the model learns important features such as the preference for each recipe for the user, it will be able to recommend recipes that the user hasn't tried yet based on his preference.

While thinking of the implementation, we encountered some challenges. We found several datasets but choosing the right ones and preprocessing them before training can be a tough task. Moreover, for a Content-based filtering system to work, we need to select the features we are going to work with, and we need each recipe to contain user rating and preference. In our case, the features might be the ingredients. This will give rise to a new challenge where the data under the selected features will be sparse since we have a lot of ingredients, and each recipe will contain a very small percentage of these ingredients. Furthermore, the data that we have might act to our disadvantage. It contains different users where each user has ratings on a scale from 0 to 5, and the weights of these ratings are focused more between 4 and 5. Hence, when we want to recommend our output will be biased since most of the recipes are rated 5, and a very small proportion of the recipes are rated between 2 and 3. Finally, we can't define a notion of good and bad since the user who rated the recipe 4 might consider it as decent and another user might consider it as bad, so we can't decipher an actual qualitative meaning from our quantitative data.

IV. Data Set Description and preprocessing

As previously mentioned, our data set "Food.com Recipes and Interactions" consists of seven different CSV files [7]. For our project, we will only deal with RAW_interactions, RAW_recipes csv files. Our goal is to find the user with the maximum number of rated recipes. We first used the RAW_interactions file: we grouped the data according to the "user_id" column, and then we sorted the users by the number of ratings to check the one with the highest rating. However, we realized that this user is biased with his ratings, and he's not the only one. We found that out by storing 30 users with the highest number of ratings in a dictionary, with the keys being the users' ids and the values being the ratings. We chose to store the first 30 users since if we picked a higher number, the total number of ratings wouldn't be sufficient for training and testing. After looping over the dictionary, we determined the frequency of each rating per user (how many times the user rated 5, 4, 3, etc.)



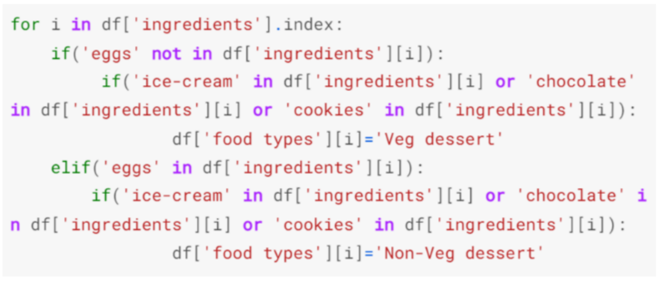
```
{5: 7445, 4: 216, 3: 4, 0: 6}
{5: 4823, 4: 667, 0: 106, 3: 7}
{5: 3717, 4: 772, 2: 5, 3: 108, 0: 26}
{5: 3227, 4: 678, 3: 82, 0: 68, 2: 16, 1: 5}
{5: 3609, 4: 273, 3: 29, 0: 5, 2: 1}
{5: 3193, 4: 139, 0: 5, 3: 13, 2: 2, 1: 1}
{5: 2559, 4: 526, 3: 117, 2: 6, 0: 80}
{5: 2375, 4: 663, 0: 17, 2: 9, 3: 40, 1: 3}
{5: 2557, 4: 427, 3: 29, 0: 5}
```

Fig. 2. Distribution of ratings for different users

Figure 2 describes the distribution of the ratings for users with the highest ratings in our dataset. As we can see 80% of the data lies between 4 and 5, the remaining lies between 0 and 3.

Hence, we tried to find a user that isn't too biased. Based on that, we picked the least biased user. In our case, it's the third one. After specifying our user of interest, we displayed the dataset while adjusting the user_id for the specific chosen user.

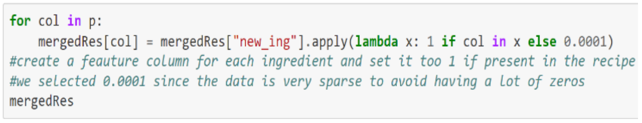
We started working on the RAW_recipes CSV file, where we have the list of ingredients. The ingredients for each recipe weren't of list type, but of string type. Hence, we transformed it to the list type and stored it in a new column to manipulate and access it easily. In addition, we dropped the columns that aren't of interest for this data set in the RAW_recipes file (minutes, contributor_id, nutrition, etc.) and similarly for the other dataset in the RAW_interactions file. Finally, we merged the two datasets into one dataset, and we called it "mergedRes." In this new dataset, we can find the users' ids, ratings, ingredients, and recipes as columns of the dataset. Here we came to a crucial part of the preprocessing, where our goal is to create a finalized dataset, with each ingredient being a column, which takes 0s and 1s as values to indicate the availability of the ingredient in the recipe.



```
for i in df['ingredients'].index:
    if('eggs' not in df['ingredients'][i]):
          if('ice-cream' in df['ingredients'][i] or 'chocolate'
in df['ingredients'][i] or 'cookies' in df['ingredients'][i]):
                df['food types'][i]='Veg dessert'
    elif('eggs' in df['ingredients'][i]):
        if('ice-cream' in df['ingredients'][i] or 'chocolate' i
n df['ingredients'][i] or 'cookies' in df['ingredients'][i]):
                df['food types'][i]='Non-Veg dessert'
```

Fig. 3. Use of Nested Loops in Previous Models

As shown in figure 3, a model that used this dataset, implemented such loops to categorize certain recipes into different labels [8]. We were inspired by this example which led us to use something similar to it to encode the used ingredients for each recipe.



```
for col in p:
    mergedRes[col] = mergedRes["new_ing"].apply(lambda x: 1 if col in x else 0.0001)
#create a feauture column for each ingredient and set it too 1 if present in the recipe
#we selected 0.0001 since the data is very sparse to avoid having a lot of zeros
mergedRes
```

Fig. 4. Encoding our Ingredients to New Features

In our case, we used the built in apply function as shown in figure 4. This helped us in filling the entries with 1s based on the availability of the ingredients in the recipes and 0.0001 based on their absence, after. We chose the value of 0.0001 due to the sparsity of the data in the columns of the ingredients.

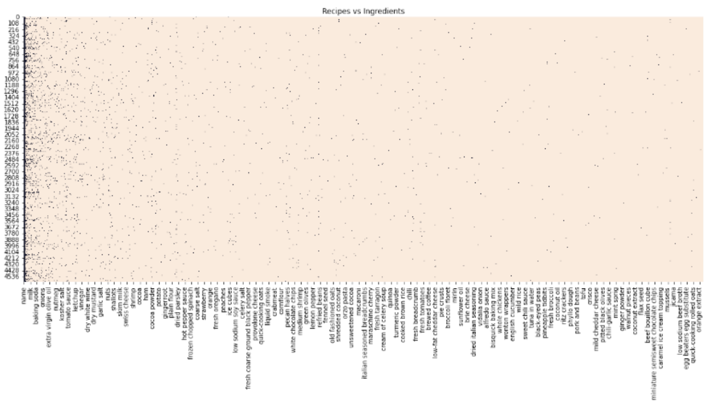
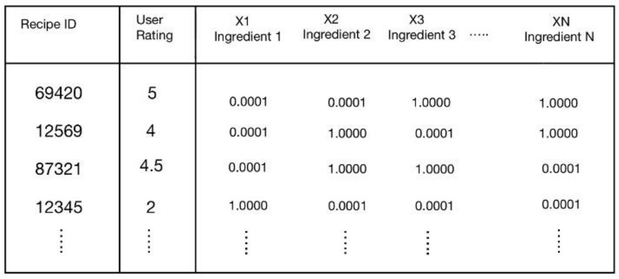


Fig. 5. Sparsity of the Data

Figure 5 models the ingredients vs the recipes. We can notice how the black dots, which are the 1s in our data frame, represent a minority in our data and are very sparse from one another. This justifies filling our 0s with 0.0001.



Recipe ID	User Rating	X1 Ingredient 1	X2 Ingredient 2	X3 Ingredient 3	.....	XN Ingredient N
69420	5	0.0001	0.0001	1.0000		1.0000
12569	4	0.0001	1.0000	0.0001		1.0000
87321	4.5	0.0001	1.0000	1.0000		0.0001
12345	2	1.0000	0.0001	0.0001		0.0001
⋮	⋮	⋮	⋮	⋮		⋮

Fig. 6. Final Data Frame Schematic

Our final data frame, represented in Fig. 6, should be our final product after feature engineering, which should be ready for model testing and experimentation.

V. Planned Experiments

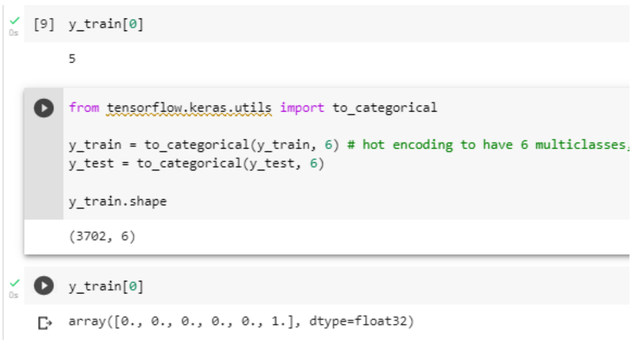
After preparing the dataset, we will split our data into training and testing sets, in which the input training and testing data will contain values for our features which are the ingredients, and our output training and testing data will include the ratings for each recipe. Our goal is to calculate the possible ratings for recipes the user hasn't tried yet so that the user will know beforehand which recipes he might like or dislike. We will test that through our input test set and evaluate the accuracy for our model by comparing our predicted ratings with our test ratings.

For our model, we were inspired by the approach of movie recommender systems, where the model uses the related genres as input features, and the ratings of previous movies as the output. In article [9], to obtain vector form features of each element, the authors proposed a method that trains a neural network model called Word2Vec CROW with content data (such as cast, crew, etc.) as the training data. The method then takes advantage of the linear relationship of the learned feature to calculate the similarity between each movie.

In our case, we will implement a food recommender system, where the ingredients of each recipe serve as a movie genres, and the recipe ratings as the movie ratings. This will be done by calculating the weight for each ingredient. Our challenge here will be to learn those weights to find the optimal rating for untried recipes. We will use as our main model Neural Networks by using different optimizers to find suitable parameters for the ingredients. In addition, we will use the Jaccard similarity metric to recommend specific recipes based on similarities between recipes. If the user wants a recipe close to one he has in mind, we will be able to recommend to him the closest recipe based on the ingredients of both and by considering his preference.

VI. Neural Network Model

Before launching our Neural Network, we prepared its corresponding inputs and outputs. We created an "X" array that contains the values for our features and a "y" array that includes the ratings. We then split our data into train and test sets where the test size is 20% of our data. Since our output is an integer that ranges from 0 to 5 inclusive, we will have our output as 6 nodes, where each node, if assigned the value 1, represents a certain rating.



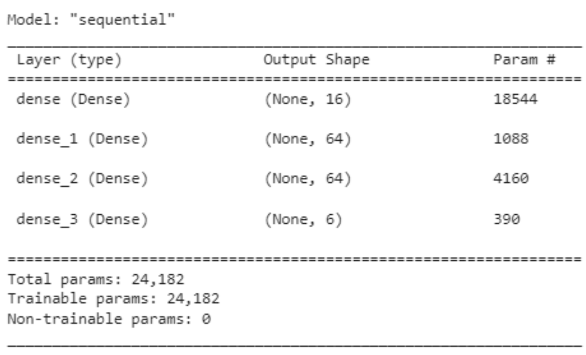
```
[9] y_train[0]

    5

from tensorflow.keras.utils import to_categorical

y_train = to_categorical(y_train, 6) # hot encoding to have 6 multiclasses,
y_test = to_categorical(y_test, 6)

y_train.shape

(3702, 6)

y_train[0]

array([0., 0., 0., 0., 0., 1.], dtype=float32)
```

Fig. 7. One Hot Encoding the Model Output

As shown in Fig. 7, if the predicted rating is 5 the output of our neural network will be 000001. This is done to ensure that the output will be an integer and not a float.

After preparing our inputs and outputs, we worked on two optimizers: Stochastic Gradient Descent and Root Mean Squared Propagation. The Stochastic Gradient Descent (SGD) iteratively modifies a machine learning network's setup to minimize some notion of error. The Root Mean Squared Propagation (RMSprop) is a variation of the gradient descent that adapts the step size for each parameter using a decaying average of partial gradients. We experimented with both optimizers, and for that we prepared two Neural Networks with different layers for each optimizer.

For the SGD optimizer, we created a Neural Network with four layers. The first three layers are made up of 16, 64 and 64 neurons respectively, and rely on the 'ReLU' activation function. The fourth layer consists of 6 neurons and relies on the 'SoftMax' activation function. The result of this activation function is the desired output.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                18544

 dense_1 (Dense)             (None, 64)                1088

 dense_2 (Dense)             (None, 64)                4160

 dense_3 (Dense)             (None, 6)                 390

=================================================================
Total params: 24,182
Trainable params: 24,182
Non-trainable params: 0
_________________________________________________________________
```

Fig. 8. Description of Model Layers

After executing this Neural Network model, we observed that there are in total 24,182 trainable parameters as shown in Fig. 8.

For the RMSprop optimizer, we created a Neural Network with 4 layers. The first 3 layers are made up of 32, 128 and 128 neurons respectively, and rely on the 'ReLU' activation function. The fourth layer consists of 6 neurons and relies on the 'SoftMax' activation function. The result of this activation function is the desired output. We applied the L2 penalty on the layers' parameters for regularization, to avoid overfitting.

VII. Results of the Neural Network Model

For the SGD optimizer, we fitted the model from the training data using 20 epochs and a validation split of 0.2. Then we evaluated the model using our test data to compute the model's accuracy and loss.

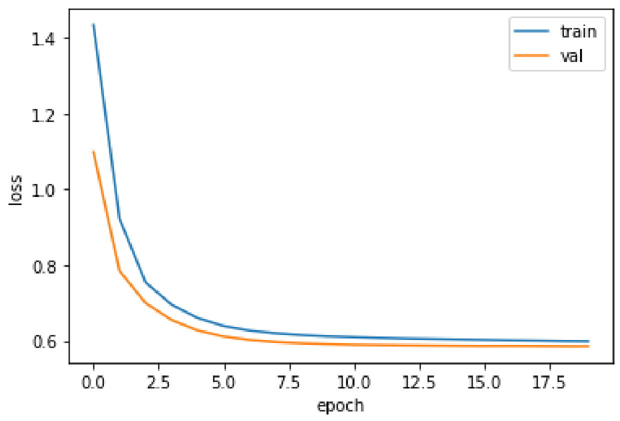


Fig. 9. Training and Validation Loss Curves Given SGD

We plotted our losses for both the training and validation sets, as shown in Fig.9. As we can see the validation loss is less than the training loss, which indicates a lack of overfitting. Both curves reach a flat surface as the number of epochs increases which indicates convergence to a local minimum and hence a minimization of error.

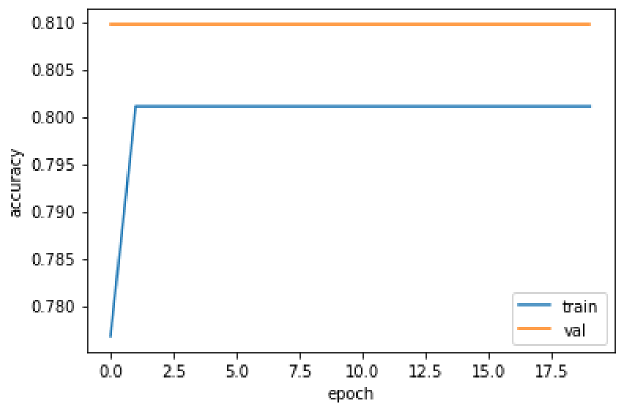


Fig. 10. Training and Validation Accuracy Curves Given SGD

Then, we plotted the accuracy of both the training and validation sets against the number of epochs. As we can see in Fig.10 when we increase the number of epochs, the training accuracy stabilizes below the validation accuracy, and we obtained a model accuracy of around 80%. From these obtained results we infer that the model behaves well under the SGD optimizer and yields very good accuracy. Hence it serves as a reliable model for many users.

For the RMSprop optimizer, we fitted the model from the training data using 15 epochs and a validation split of 0.2 also. Then we evaluated the model using our test data to calculate the model's accuracy and loss.

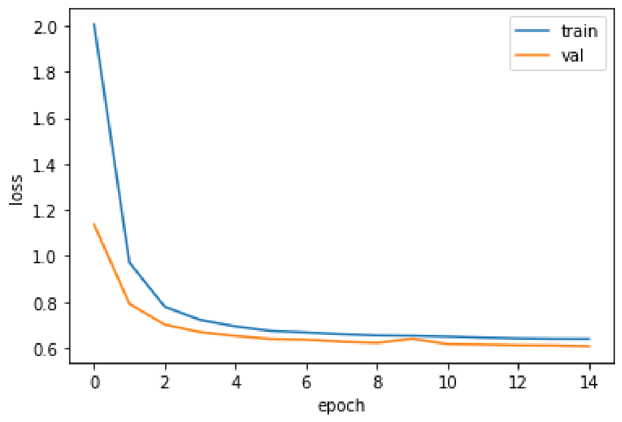


Fig. 11. Training and Validation Losses Curves Given RMSprop

After that, we worked on something similar to what we did for the SGD optimizer. We plotted the loss curves for both the training and validation sets, as shown in Fig.11. Also, the validation loss here is less than the training loss, which indicates a good result and similarly a lack of overfitting. Both curves reach a flat surface as the number of epochs increases which indicates convergence to a local minimum and hence a minimization of error.

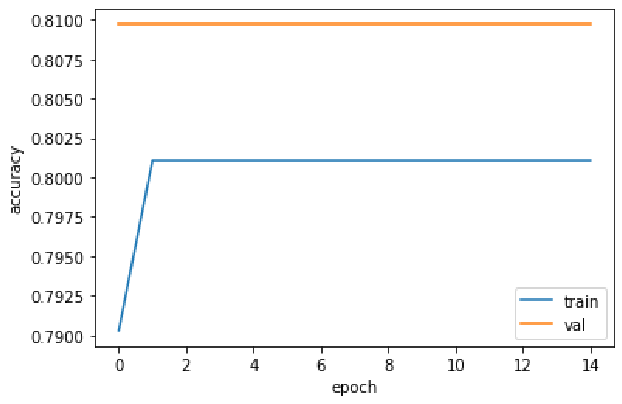


Fig. 12. Training and Validation Accuracy Curves Given SGD

After that, we plotted the accuracy of both the training and validation sets against the number of epochs. As shown in Fig.12 we obtained approximately the same result as with the SGD and with a model accuracy of around 80%. From these obtained results we infer that, like the SGD, the model behaves well under the RMSprop optimizer and yields very good accuracy. Hence it serves also as a reliable model for many users.

Both optimizers yield very good results and one can opt for any of them if the model layers are selected properly. Furthermore, in article [10], Giovanni experimented with a dataset of books containing 1 million ratings for 10,000 books for more than 50,000 users. Through collaborative filtering, he did a matrix factorization network that achieved a validation loss of 0.93 as well as a deep neural network recommender which obtained a validation loss of 0.76. Compared to our work, our model attained a less validation loss of around 0.6 for both optimizers. However, our work only included one user, whereas his work contained 50,000 users.

VIII. Jaccard Similarity Metric

For our second model, we used the Jaccard similarity metric, which calculates the similarity between any two given data points.

$$\begin{aligned} D(P,Q) &= 1 - J(P,Q) \\ &= 1 - \frac{|P\cap Q|}{|P\cup Q|} \\ &= \frac{|P\cup Q| - |P\cap Q|}{|P\cup Q|} \\ &= \frac{|P\cap Q| + |P-Q| + |Q-P| - |P\cap Q|}{|P\cap Q| + |P-Q| + |Q-P|} \\ &= \frac{|P-Q| + |Q-P|}{|P\cup Q|} \end{aligned}$$

Fig. 13. Jaccard Similarity [11]

We define Jaccard similarity by taking the intersection of two sets of data over their union as shown in Fig. 13. The Jaccard distance is calculated by subtracting the Jaccard similarity from 1. However, for us to calculate the Jaccard distance between two sets of data, we need our data sets to be of binary forms. For this reason, we created a new dataset with the values 0.0001 adjusted to 0. To build our model, we started by performing the Jaccard distance between all recipes in our dataset and converting it to square form. Then, we computed the corresponding Jaccard similarity.

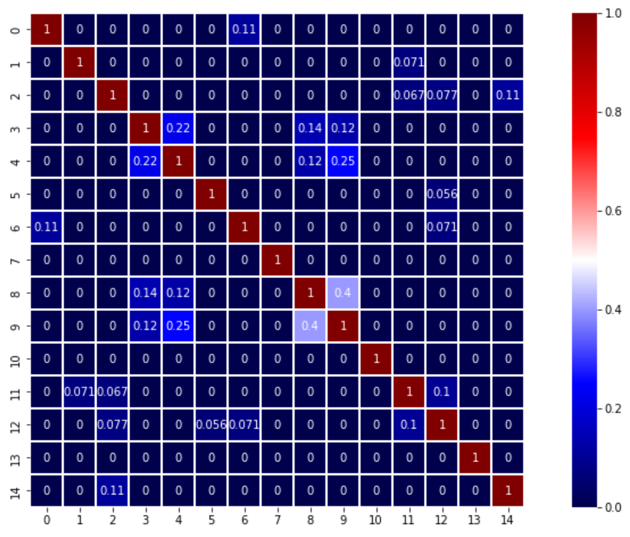
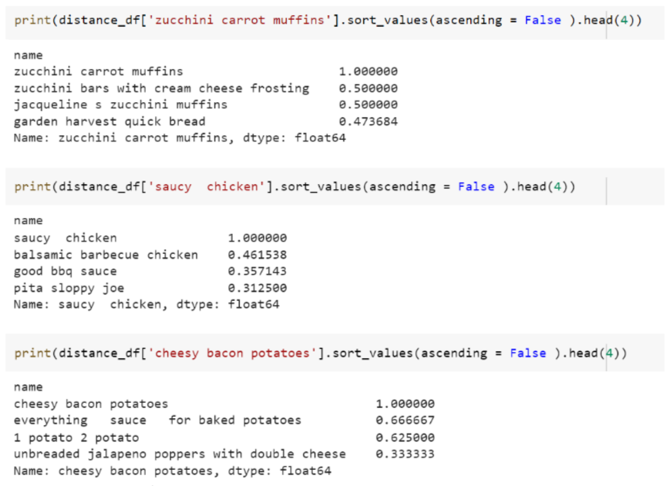


Fig. 14. Similarity diagram for 15 recipes

To visualize our work clearly, we plotted the similarity for 15 recipes from a portion of all the recipes as we can see in Fig. 14. We can observe that the diagonal of the matrix is all 1s since the recipes are compared to themselves. Also, the other values are mostly 0s, and this is justified since our data is very diverse, and our recipes range from breakfasts to desserts to main dishes to sauces, etc. Hence, if we're comparing a chocolate brownie to a chicken breast it's logical to obtain a similarity of 0. However, if we pick one recipe and sort its similarity with other recipes, we can obtain very promising results.


```
print(distance_df['zucchini carrot muffins'].sort_values(ascending = False ).head(4))

name
zucchini carrot muffins                   1.000000
zucchini bars with cream cheese frosting  0.500000
jacqueline s zucchini muffins             0.500000
garden harvest quick bread                0.473684
Name: zucchini carrot muffins, dtype: float64
```

```
print(distance_df['saucy  chicken'].sort_values(ascending = False ).head(4))

name
saucy  chicken             1.000000
balsamic barbecue chicken  0.461538
good bbq sauce             0.357143
pita sloppy joe            0.312500
Name: saucy  chicken, dtype: float64
```

```
print(distance_df['cheesy bacon potatoes'].sort_values(ascending = False ).head(4))

name
cheesy bacon potatoes                              1.000000
everything   sauce   for baked potatoes            0.666667
1 potato 2 potato                                  0.625000
unbreaded jalapeno poppers with double cheese      0.333333
Name: cheesy bacon potatoes, dtype: float64
```
Fig. 15. Testing for Similar Recipes

As shown in Fig. 15, if we sort the similar recipes for some of our dishes, we obtain a similarity between 30% and 70%, which is a very descent result. For example, if our user wants something similar to "cheesy bacon potatoes", he can go for "everything sauce for baked potatoes" which has a similarity of 66.67%.

IX. Conclusion

In this paper, we proposed a food recommender system using two different metrics, the Neural Network model and the Jaccard similarity metric. For the Neural Network model, the inputs represented the ingredients, and the resulting output represented the rating. Based on this rating, the user will either stick to the given recipe or deviate from it since it's not of his preference. We worked on two optimizers: SGD, and RMSprop. We obtained promising results with a validation loss of around 0.6 and a validation accuracy of around 80% for both optimizers. For the Jaccard similarity metric, we worked on calculating the Jaccard similarity, after getting the Jaccard distance between the given recipes. Also, this metric gave us promising results with a similarity between 30% and 70% for similar recipes. Based on the work we have done and the results we have obtained, our work can be further investigated in the future. All this work was done based on one user. What if it was expanded to tackle multiple users? Future work with this progress can involve broadening the scope to account for multiple users and a lot more recipes just like article [10] did. However, in the original data set we worked with from [7], there were users who only had two or three ratings. So, for such an expansion to occur, more data for more users needs to be gathered and incorporated with the data we have. With the results we have obtained, such an expansion can produce a promising outcome.

X. References

[1] E. S.-L. Abel Martinez-Gorospe, "Food recommender system based on weighted ingredients, body mass index and allergies; using the Random Forest algorithm," IEEE, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9534806.

[2] G. R. I. I. K. T. B. R. S. S. Kalajdziski, "Cuisine classification using recipe's ingredients," IEEE, 2018. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8400196..

[3] M. Sedlák, "Content-based Recommender," Masaryk University, 2021. [Online]. Available: https://is.muni.cz/th/vdqix/433342_matus_sedlak_dt.pdf.

[4] S. N. S. K. S. A. S. V. B. Reddy, "Content-Based Movie Recommendation System Using Genre Correlation," in *Smart Intelligent Computing and Applications*, Singapore, Springer, 2018, pp. 391-397.

[5] K. K. A. S. S. K. A. Y. Nitasha Soni, "Machine Learning Based Movie Recommendation System," *IEEE Xplore*, no. IEEE, p. 6, 2021.

[6] D. S. Mahmoud and R. I. John, "Enhanced content-based filtering algorithm using Artificial Bee Colony optimisation," IEEE, 2015. [Online]. Available: https://ieeexplore.ieee.org/document/7361139.

[7] S. LI, "Food.com Recipes and Interactions," Kaggle , 2019. [Online]. Available: https://www.kaggle.com/datasets/shuyangli94/food-com-recipes-and-user-interactions?select=RAW_interactions.csv.

[8] A. MISHRA, "Food Recommender," Kaggle, 2020. [Online]. Available: https://www.kaggle.com/code/aayushmishra1512/food-recommender.

[9] Y.-L. W. M.-K. H. C.-Y. T. Hung-Wei Chen, "Fully content-based movie recommender system with feature extraction using neural network," *IEEE*, p. 6, 2017.

[10] G. Marchetti, "Recommendations with Neural Networks," Medium, 29 January 2019. [Online]. Available: https://medium.com/@gmarchetti/recommendations-with-neural-networks-ad25ea9b6380.

[11] M. Harmouch, "17 types of similarity and dissimilarity measures used in data science," Towards Data Science, 14 March 2021. [Online]. Available: https://towardsdatascience.com/17-types-of-similarity-and-dissimilarity-measures-used-in-data-science-3eb914d2681.