Recipes recommendation system
Final Report

Alexandre Duval, Ibrahim Chiheb, Philibert de Broglie, Sebastien Saubert

5 Deccember 2019

1 Abstract

For decades, guiding customers in their choices has grown tremendously and has become a must-do for many firms. The recommendation system they exploit to do so are increasingly complex, and many solutions have emerged on top of the traditional Content Based and Collaborative Filtering. A series of breakthroughs occurred in the late 2010's with the Netflix Prize Competition [3]. In particular, some methods based on deep learning solutions such as Autoencoders, Matrix Factorisation and Hybrid models where invented.
This project offers a journey in the world of recommendation systems. It covers various methods known as state of the art - Content Based Filtering, Collaborative Filtering, Matrix Factorization, Hybrid system and AutoEncoder - applying them on our specific use case, that is the recommendation of recipes for the food.com website. Note that we do not stop at their individual implementation and comparison, we also combine them cleverly to address some well known issues of recommender systems: lack of ratings, cold start problem, unbalanced distribution of ratings, etc. In each case, we propose a list of ten robust recommendations to the user while distinguishing between active users and passive users or fresh joiners.

2 Introduction, Motivation and Problem Definition

Nowadays, with online services, users are overwhelmed by the amount of products they are being proposed, ranging from movies and musics to housing decoration and sportswear. Given the huge variety of choices, it's often difficult to find the one that suits you the best, a task that can be very time consuming and frustrating. Indeed, the challenge is not to know if your dream product exists, it is to find it. This is why companies like Amazon, Netflix and Spotify, to avoid users navigating forever the flow of products when looking for something, provide personalised offers.

The powerful recommendation systems they use to do so have become essential tools to our society. In addition to the comfort they bring to users, they help generate phenomenal profits, which make their use even more ubiquitous. For instance, a recent study estimated the added value of Netflix's recommender system to more than one billion.

Following the current trends concerning healthy alimentation and eco responsible behavior, there is a surge in home cooking, which leads people to find new recipes that match their tastes and skills. But referring to the issue mentioned above, there is a large range of recipes you can choose from, but no way to know which one to pick. Indeed, while recommender systems are extremely popular for movies, musics or retail, they are not so developed in the cooking world, while extremely useful. This is why we built a recipe recommendation system that offers people relevant and personalised ideas of recipes.

In particular, we **want our recommendation system to show the following specific properties**:

1. Being able to recommend recipes to any new user, based for instance on recipes' popularity and ratings.
2. Being able to provide recommendations that fit the mood and desires of the user (diet, quick and easy to cook, new releases...)
3. Being able to predict the rating each user would give to all recipes to then provide each user with a personalised list of suggestions, which should contain the recipes that he is the most likely to enjoy. (main part).
4. Being able to help the user discover completely new and unpopular recipes to satisfy his curiosity and help him explore new areas of cooking.

In addition to the above, we don't want to recommend the same recipe twice, we want to provide diverse genres of recipes and we want our recommendations to be highly personalised (vary across users). We use the Root Mean Square Error (RMSE) metric, which computes the distance between predicted ratings and real ones, to evaluate the relevance of our suggestions when trying to predict his favourite recipes.

Note that we have identified **2 types of users** in our dataset that need to be distinguished. The first one, Thomas, is actively involved on the platform. He has tried and rated various recipes, which makes him easy to deal with. Indeed, we have sufficient information about his tastes to provide him with relevant recommendations. The second one, Gael, has not rated (many) recipes and is therefore not giving us enough insights. He might be new to the plateform, not a

recurrent user or simply he does not rate the recipes he tries. The fact is that we cannot predict the recipes he would like accurately. This is called the cold start problem. To get around this, we only recommend him recipes that are similar to the few recipes he actually rated; and recommend him some very popular and appreciated recipes. We even apply some filters on the latter to take his desires and mood into consideration. These suggestions are however not very personalised (except for the hybrid systems that takes into account metadata and ratings)

3 Related Work

While recommendation systems have been deeply studied for decades in domains such as retail, music or movie, it has not been the case for the cooking world and in particular regarding recipe recommendation. Our work therefore mainly consists in transferring the knowledge acquired in the cited domains to the recipe suggestion application.

The main source of inspiration resides in the movie recommendation field, mainly because one of the principal actor of the field, Netflix, launched a worldwide competition in 2007 aimed at improving its recommender system by more than 10 % according to the RMSE metric. This was called the Netflix Prize competition [3] as a one million award was promised to the winner. It lasted for three years and originated the creation of new state of the art algorithms for recommendation systems, such as Matrix Factorisation [1]. Furthermore, in part thanks to its worldwide influence, new detailed resources emerged concerning the creation of recommender systems. Most techniques from Content Based Filtering to Autoencoders [6] are well developed in the literature. Before even starting our project, we spent a lot of time documenting on each of them and we established the plan of our project using those papers.

However, we did not only transfer and learn to implement those algorithms on our cooking problem, but also innovated in several ways. Firstly, we combined different approaches (Demographic filtering, CBF, CF, Matrix factorisation, Autoencoder, Hybrid system) in our project to get a recommender system satisfying the properties mentioned in the previous parts. Secondly, without delving into details, we made an original train/test split, created a brand new similarity metric (specific to our case), changed the rating system to a simple like/dislike in some models, improved the matrix factorisation algorithm, incorporated a new loss function and a new (weighted) rating metric for recipes, etc. We will see this in more details in the following parts.

4 Our Data

The **datasets** are taken from Kaggle. They contain information concerning the ratings given by users to cooking recipes on the website *food.com*. They also include various attributes of the recipe such as the name, the time it takes to cook, the ingredients used, the calorie level, the technique to execute it, the number of steps and even some tags. Note that these pieces of information are split across datasets and mergers were frequently needed. Overall, the datasaset contains data from 2000 to 2018 that represents: 180K recipes (denoted further i) and 25K users (denoted further u) for a total reviews above 700K.

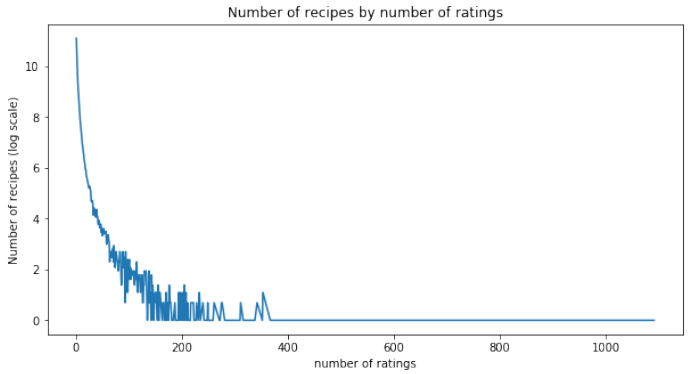

Figure 1: Recipe density per rating

The data pre-processing step includes primarily converting attributes such as recipe name, tags or lists of ingredients into exploitable data. It was accompanied by a data visualisation step aimed at understanding the problem. This phase was particularly useful to study the distribution of the ratings, which reveals a strong bias towards great ratings; as well as the sparsity of the ratings across users and recipes (see figures 1 & 2).

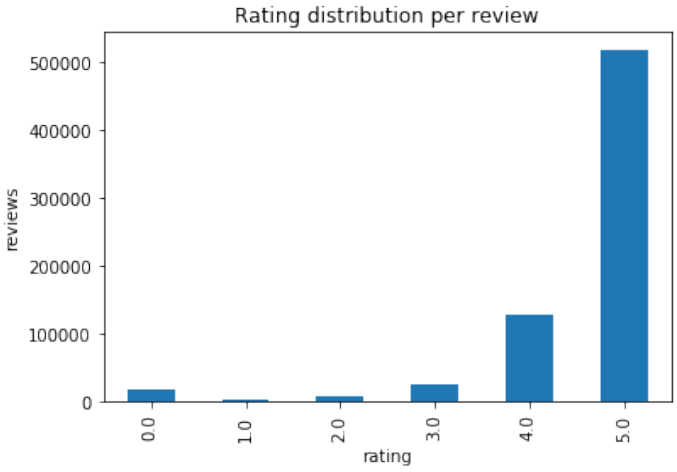

Figure 2: Rating distribution

The above step enabled us to identify two types of users and to deal with them accordingly. This especially involves creating a **new filtered dataset** made only of users having

rated at least 5 recipes (user Thomas) and recipes having been rated at least 5 times. This dataset is used as such for some tasks and is converted into a huge sparse user-recipe matrix for others, where each cell contains the associated rating (from user u to recipe i) if it exists and a NaN value if it does not. These new filtered datasets are used for the most advanced techniques of our project while the original one is only selected for the most basic steps.
To be more specific about the matrix representation M_{ui}, imagine user u_0 has rated recipes i_0 and i_1 by 4.0 and 5.0 and user u_1 has only rated rated recipes i_0 by 2.0, then $M_{ui} = \begin{bmatrix} 4 & 5 \\ 2 & NaN \end{bmatrix}$. The original matrix has dimension $180K \times 25K$ while the filtered one has dimension $11K \times 10K$ with 280K reviews.

The last point worth mentioning is the **train/test split** of our new dataset. (We don't need to split the original one as it will not be used to compute predicted ratings). Although it seems straightforward at first sight, the possibilities are multiple. The way we chose consists in selecting the latest 20% ratings made by each user (according to the date of the rating) and placing them into the test set. This method enables us to prioritise users over recipes by putting the appropriate number of ratings for each user in the test set. Furthermore, it leads to testing our predictions for each user on the latest recipes he has rated, which is perfectly consistent. One minor drawback we could argue is that not all recipes are guaranteed to be represented in both train and test set. Fortunately, even if is not essential at all, only one recipe is missing in the train set and a few in the test set.

5 Model Methodology, Algorithm and Evaluation

5.1 Demographic filtering

The first model we implement is called Demographic Filtering and offers generalised recommendations to every user, based on recipes popularity. This system is not sensitive to the interests and tastes of a particular user, meaning that Gael and Thomas both receive the same suggestions. The basic idea behind this system is that movies that are more popular and critically acclaimed will have a higher probability of being liked by the average audience. Although considered too simple to constitute a full recommender system, it is easy to implement and turns out to be essential to avoid the cold start problem. Indeed, these suggestions can be made even when the user has not rated any recipes, unlike some more advanced methods that we will investigate later. For this model, instead of choosing the rating or the number of ratings received as a proxy for popularity, we combined both to form a weighted score. Otherwise, recipes having exactly one top rating would be favoured and likely to be recommended. The **formula** is the following. The ten recipes presenting the highest score are suggested in the category "Biggest successes" .

The second improvement we have made is to add three filters on top of this score, namely "Diet", "Quick and easy cooking" and "New releases". They also output recipes presenting the best score, but among a subset of the data defined with respect to a specific variable value. Regarding "Diet", we only consider recipes presenting a low calorie level. For "Quick and easy cooking", recipes that take less than 10min to prepare and that involve less than 6 steps. Finally, for "New releases", the 10 best recipes among the 20 newest ones. Note that we avoid recommending recipes already suggested and recipes already tried by a user.

The last improvement effectuated involves selecting the 10 best recipes at random from the 100 best ones, for each category named above (except "New releases"). This allows our recommender to diversify a bit the recipes suggested at each new connection of the user.

5.2 Content Based Filtering

The Content base filtering method, will suggest recipes which are similar to the ones the user rated. As this method base its suggestions on some characteristic of a reference recipe, it therefore only need one rated recipe to generate a recommendation. For example, Gael only rated one recipe "tommy fries" (rated 4 out of 5) was recommended the "mom's camp fire potatoes" recipe as it is very similar to the one he rated. This is the reason why this algorithm is often used in the case where very few information from the user has been gathered.
Hence, in the case of our project, it was first thought to **evaluate recipes similarity based on the ingredients contained in the recipes**. Two recipes are considered to be similar if the distance metric chosen (Pearson Coefficient, Cosine Similarity...) between them two, is low. We first need to gather all the recipes (rows) and ingredients (columns) we have into one matrix. Every ingredients will then have a value of 1 when present in a recipe, otherwise it will be set to 0.

Now that we have a large matrix of "0" and "1", we can choose our relevant distance metric method.**Cosine similarity** measures the cosine of the angle between two vectors in a multidimensional space, it does not take into account the weights (size, here its 0 or 1) of the vectors. Hence, it is perfect to determine similarity by presence of a dimension (ingredient). It was therefore implemented in a function that stored the similarity result and its according recipe in a directory. Our filter then takes as inputs the ID of the user on whom to give a recommendation, the number of recommendation desired and the number of reference recipes to perform content base filtering on. The algorithm will output the recommended recipes which got the highest similarity score (can be from 1 or multiple reference recipes). At this step, our CBF was complete as it could generate

recommendations for a given user. For example, Gael who has only rated the "Tommy Fries" recipe, receives 10 recommendations.

Recipe Name	Score
Bacon cheddar potato tart michael smith	0,818171
mom's camp fire potatoes	0,796160
chessy bacon ranch pasta salad	0,774249
slow cooker cheesy bacon ranch potatoes	0,764499
arkansas breakfast casserole	0,747760
chessy italian potatoes	0,740876
contest winner twice baked potato casserole	0,732190
parsnip potatoes and bacon	0,720037
german fried potatoes	0,719324
bacon cheeseburger chicken	0,717650

Gael's CBF recommendation on "Tommy Fries"

Seeking for a top quality algorithm, it was decided to optimise our basic CBF. Because cosine similarity was taking too much time to compute, **SVD** was introduced before this step on the "ingredient matrix" keeping 2000 dimensions to improve computational time while preserving performance. Without SVD, it would take 50 seconds to produce 5 recommendations for one recipes, so to do CBF for a user checking 5 recipes and giving 5 final recommendation, it would take approximately 5 minutes. With SVD, it we would compute the matrix once (between 15/20minutes) and then it would take approximately 7 seconds to give final recommendations for a user. With more than 150k recipes and more than 280k users, it is much better to compute once for 20minutes and then 7 seconds, rather than 5 minutes every time. Also we observed no decrease in our performances while implementing SVD.

Recommendations	SVD	No SVD
5 final recommendations based on 5 recipes	4'15" (255 sec)	7 sec
10 final recommendations based on 10 recipes	7'30" (450 sec)	15 sec

CBF computation Time

In another attempt to improve our CBF algorithm we decided to take into account **more features** to our matrix so that recipes would be similar on their ingredients but also on their cooking time, number of steps, number of ingredients and their calorie level. Because the data was of a different format, the added features had to be normalised in order to work with our cosine similarity method.
Some others improvements were added to hone our CBF recommendations. In fact, we ensure that the reference recipes on which the algorithm would perform the content base filtering would be rated at least 4 and over (out of 5). We did not choose the maximum rating as some users with few ratings might not have enough 5 rated recipes. Furthermore, we ensure that no recipes already rated by our user would be recommended to him by filtering out those recipes from the ones we check similarity. The algorithm will select the references recipes randomly between the best recipes.
It was also thought to add **weights on some features** to make them more important than others in the calculations of recipes similarity. For example, it could make sense that ingredients and cooking time should have more importance to recommend a recipe than the number of ingredients. We therefore tried this but no significant changed occurred, even when combining weights on several features. Depending on what a particular user is looking for, (preference on cooking time, on number of ingredients etc...), the weights disposition would not be the same. Because the project was coming to an end, no further work was made on the features weights (even though it can be find in our code).

The main limitations of that kind of algorithm are that its recommendations because they are based on a single recipe, are not very personal, and do not allow true exploration (low diversity of recipes suggested). Some ideas to change this and hone our recommendations were brought up. One of the ideas was to create clusters on recipes so that we could offer some recommendations according to the kind of meals that exist. For example, categorising recipes as snacks, fancy diner, easy meal, vegan, italian etc.. This could have been done, by processing the recipes tags (tags, steps tokens, name tokens etc...) present in the data, but it turns out it was a hell of a pain to process this data, therefore we left it as an option but was not performed due to a lack of time. We also thought that in order to personalise recommendations it would be nice if users could choose their preferences by giving the weights to the algorithm by themselves via a GUI for example, choosing on a fixed scale the importance they give to time similarity, calorie level, ingredients etc... Again, this would benefit of recipes categories (clusters). It was noted that although for some recipes, the best similarity scores stay low (around 0,6) it does not mean that the algorithm is giving bad recommendations, it only means there are no better recipes matches in the data set.

5.3 Collaborative Filtering

We now tackle Collaborative Filtering and in particular user-user neighbourhood method. This technique recommends to a specific user recipes liked by similar users, and that has not been tried before by this user. It aims to predict the recipes a user may have interest in, by computing the rating he would give to each recipe based on similar users' ratings. Collaborative Filtering fixes most of the issues for content based filtering as it does not require much metadata. It only works with the huge sparse interaction matrix M_{ui} defined previously.

We show in our code how it works for the user Thomas (T). Doing it for all users is feasible but computationally expensive. We first need to compute the similarity metric between Thomas and all other users. An obvious step is to choose a relevant similarity metric. Although cosine similarity does

a decent job, we found a much better measure of similarity.

$$sim(T,B) = \sum_i \frac{T_i \cdot B_i}{\mu_T \mu_B} \times \frac{1}{\mid T_i - B_i \mid +1}$$

where μ_T, μ_B are the mean ratings of each user T,B and T_i, B_i their ratings for recipe i. To make it work, we replaced NaN values by -1 in the user-recipe matrix and rescaled every cell by 1.

This similarity metric is perfect because it satisfies (for each component of the sum) the following requirements:

- stays constant when two users have not rated a recipe
- stays constant when only one user rated that recipe (we want similar users to have rated some different recipes to be able to recommend some to the other)
- increases when two users rated the same recipe. Additionally, we want this increase in similarity to be inversely proportional to the difference between the two ratings given, since two users attributing very distant ratings are not so similar.
- takes into consideration differences of rating system across users (some give higher ratings than others in average). This allows us to account for preferences.

Once the similarity score computed, we select the 25 most similar users to Thomas and infer for each recipe the predicted rating that Thomas would give based on the rating these similar users gave. The predicted rating is obtained via the formula:

$$score(T,i) = \frac{\sum_{u \in S} r_{ui} \cdot sim(T,u)}{\sum_{u \in S} sim(T,u)}$$

where S refers to the set of similar users having rated that recipe.
However, we note that computing it directly would indicate a bias towards recipes having been rated with a 5.0 only by a one user. This probably allows a better exploration of range of recipes. So instead, we weight ratings of the top 25 most similar users via the similarity score and compute the mean predicted rating for each recipe. As this includes re-scaled ratings, meaning also all 0 ratings created for recipes not tried by a user, it does not yield a true predicting rating but outputs a score that is relevant to our objective. Indeed, the highest scores do correspond to the recipes he is the most likely to like, and favours popular recipes (those tried by several similar users). We then compute the prediction score of these recipes to get a better insight of the predicted ratings. Note that recipes Thomas has already cooked were disregarded because irrelevant.

This user-user neighbourhood method completes Demographic and Content Based Filtering. It is more performant as our recommendations are finally personalised to each user. Furthermore, it gets closer to reality where the recipes you try are often influenced by what your friends having similar tastes liked. However, it does not deal with the cold start problem, which is why the two first approaches are still necessary. Another undesirable aspect of this approach is the difficulty to scale the results, which is why we will see different approaches to collaborative filtering in the next sections.

5.4 Matrix Factorisation

Let's now move to the most complex method of our project: Collaborative filtering - Matrix factorisation. One way to handle the scalability and sparsity issue created by CF is to leverage a latent factor model to capture the similarity between users and recipes. We would thus like to represent users and recipes in a lower dimensional latent space of dimension k by decomposing the interaction matrix ($m \times n$ user-recipe matrix) into two submatrices of lower dimension ($m \times k$ and $k \times n$) using a method similar to SVD. Note that in this new space, user-item interactions would be modelled as inner products. In other terms, matrix factorisation approximates a large sparse matrix by a product of two long and slim matrices.

$$\begin{pmatrix} & r_{ui} & \end{pmatrix} = \begin{pmatrix} & — & p_u & — & \end{pmatrix} \begin{pmatrix} & | & \\ & q_i & \\ & | & \end{pmatrix}$$

Figure 3: Matrix factorisation

Accordingly, the first **matrix** associates each user with a vector p_u that represents the affinity (interest) of user u for each of the latent factors while the second matrix associates each recipe i with a vector q_i representing the extent to which this recipe possesses those factors. In other words, if u has a taste for factors that are endorsed by i, then $p_u \cdot q_i$ would be high as it captures the user's overall interest in the recipe's characteristics. The resulting dot product $p_u \cdot q_i$ aims to approximate r_{ui}.

The **ultimate goal** of this method is to predict the ratings a user would give to the recipes he has not tried yet by computing $\hat{r}_{ui} = p_u \cdot q_i$, $\forall (u,i) \notin K$, where K is the set of pairs of indexes (user, recipe) that already have a rating. We use the observed ratings of the train set to fit our model and the latent variables found to predict the missing ratings for each user. Indeed, the output of this method is another $m \times n$ interaction matrix but filled with a predicted rating $\hat{r}_{ui}$ for all user-recipe couples (no NaN value). We can evaluate the quality of our predictions, meaning how good we are at predicting the ratings users would give to recipes, via the Root Mean Square Error (RMSE) metric: $\sqrt{(\sum_{u,i}(r_{ui} - \hat{r_{ui}})^2)}$. The lower the RMSE, the better the performance and the more relevant our recommendations. Computing it on the test set would give us an idea of how well our model generalises.

At this point, you might wonder how do we find the latent variables of those submatrices ? Essentially, we turn our recommendation problem into an optimisation problem. This comes back to optimising and solving the following minimi-

sation problem:

$$\min_{p,q} \sum_{(u,i)\in K} (r_{ui} - p_u q_i)^2 + \lambda(||q_i||^2 + ||p_u||^2)$$

The approach we choose to minimise this equation utilises the Stochastic Gradient Descent in the network described below. For each observation, we feed to the network the corresponding u and i separately as two one hot encoded vectors, create two embeddings corresponding to p_u and q_i, flatten them, compute their dot product and output the prediction r_{ui}. All vectors are randomly initialised. And whenever the algorithm loops through all ratings in the training set, for each given training instance, the system predicts $\hat{r}_{ui}$ and computes the associated prediction error $e_{ui} = r_{ui} - p_u \cdot q_i$ Then it modifies the parameters by a magnitude proportional to gamma in the opposite direction of the gradient, yielding:
$q_i \leftarrow q_i + \gamma(e_{ui}p_u - \lambda q_i)$
$p_u \leftarrow p_u + \gamma(e_{ui}q_i - \lambda p_u)$

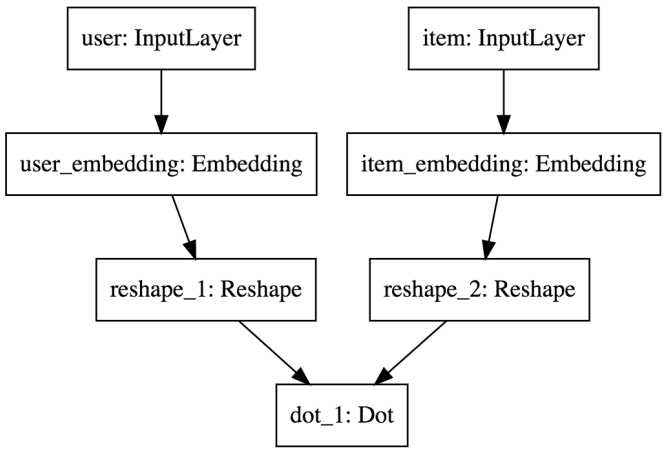

Figure 4: Matrix Factorization with dot product

The model is then tuned by GridSearchCV for the following hyper-parameters: the dimension k of the latent factors vector, lambda, alpha and the number of epochs. We are especially interested in the resulting k and number of epochs, which we set to 20 and 24 respectively in this case.

The results obtained are convincing. Using the test set as our validation set during the training, we obtain a MSE of 0.51 for the training set and 1.10 for the test set, which is desirable for this kind of applications. The predicted ratings for the test set instances are not all extremely close to reality but the order of preference is preserved, meaning that the recipes a user loved (5.0 rating) usually get a higher predicted rating than recipes he liked (3.0 or 4.0) or disliked. This is all that matters because we are only interested in suggesting for each user the recipes presenting the highest rating. We do not really care how close to the true rating the latter is.

Dealing with overfitting is one of the main concern here. Indeed, given our application, we grant a big importance to the generalisation of our predictions to the test set. Otherwise, we could train the model intensively and obtain a very low MSE (0.04) but it coincides with a surge in the validation error and predictions that are not relevant anymore... Something that needs to be highlighted is that the more we overfit, the bigger the range of our predicting readings. Conversely, the more we underfit, the closer the predictions for each user get to its mean rating. Finding the point in between is highly desirable, which is why we spent some time finding the right k and number of epochs.

To produce recommendations for Thomas, we predict all the ratings he would give to unseen recipes and output the names of the ten recipes presenting the highest score (see list Appendix). We have applied the same process for a model trained on the full dataset because ideally we would not want to exclude precious information by using a test set. The size of the validation set was progressively reduced to insure that this new model was similar the one we studied and optimised. The recipes recommended are not exactly the same but they appear to share similar characteristics. It's however difficult to measure precisely how it performs.

Let's **slightly modify this model**. Instead of using a fixed dot product to compute the predictions from the latent vectors, we add some dense layers so the network can find better combinations. Results are quite similar to the first model, we obtain a slightly lower validation MSE: 1.01 and higher training error 0.62. This induces the model generalises slightly better and overfits less. The recommendations are produced similarly and a few recipes suggested before appear again here. The set of recipes recommended seems however a bit more diverse. (see list appendix)

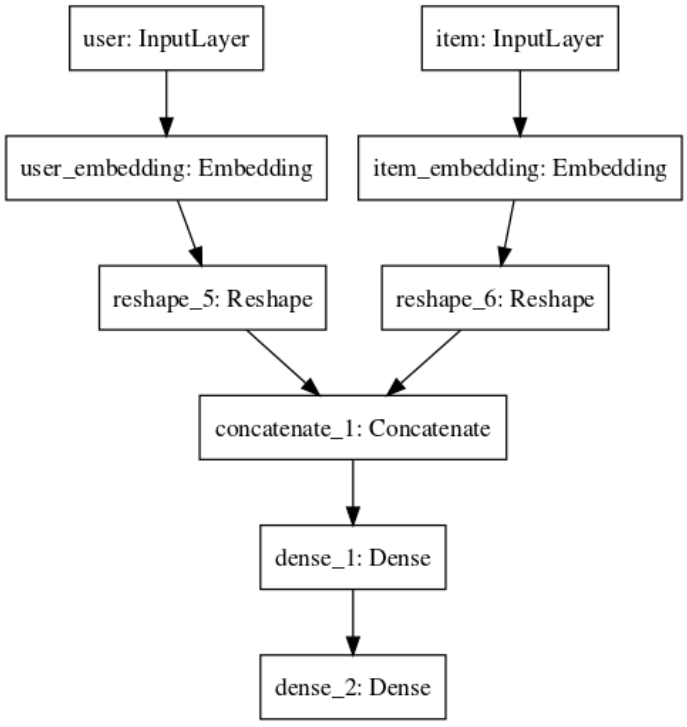

Figure 5: Matrix Factorization with concatenation

However, much of the observed variation in ratings is due to effects associated with either users or recipes, known as biases or intercepts, independent of any interactions. Indeed, there exist large systematic tendencies for users to give higher ratings than others and for some recipes to receive higher ratings. From this conclusion, we introduce $b_{ui} = \mu + b_i + b_u$ that represents the bias involved with r_{ui}. μ refers to the mean rating over all user-movie ratings while b_i and v_u refer to the respective deviation of recipe i and user u from the average, that is their specific effect. We now have $\hat{r}_{ui} = b_{ui} + p_u \cdot q_i$ and the minimisation problem changes slightly and is still solved by SGD:

$$\min_{p,q,b} \sum_{(u,i)\in K} (r_{ui} - b_{ui} - p_u q_i)^2 + \lambda(||q_i||^2 + ||p_u||^2 + b_u^2 + b_i^2) \quad (1)$$

This **new improvement** was implemented using the Surprise library package, which is specific to recommendation systems. It allows us to easily represent this new model and explore a different way of producing recommendations. When training the model on the whole dataset, we estimate the RMSE at 0.882 using cross validation. This suggests that this model achieves better performances than previous ones. Additionally, we have not mentioned it so far but it is possible to compute all predicted ratings at the same time as though it takes a bit of time.

Finally, one advantage of deep learning models is that movie-metadata can easily be added to the model. We thus bring together ideas from content based and collaborative filtering to build an engine that gave recipe suggestions to a particular user based on the estimated ratings that it had internally calculated for that user. We also tf-idf transform the short description of all recipes (or/and the list of ingredients) to a sparse vector and add information concerning the time it takes to cook, the calorie level, the number of steps.... The model will learn to reduce the dimensionality of this vector and how to combine metadata with the embeddings. These kind of **hybrid systems** can learn how to reduce the impact of the cold start problem. This new model yields very satisfying performance: loss of 0.634 and 1.03 on validation. The difference with previous models is that it can deal with users that have very few associated ratings. The suggested recipes are quite different from their predecessors but it makes complete sense given the amount of new data we just added to the model.

5.5 Auto Encoder

The purpose of an AutoEncoder is to predict, for a given user, all missing ratings. To do this, the auto encoder is composed of two parts (see figure 6). First, the encoding one which aim to reduce the ratings from an user into a lower space, producing a representation called embedding. Secondly, a decoding part that generate back the ratings (including the predictions for the missing ones).

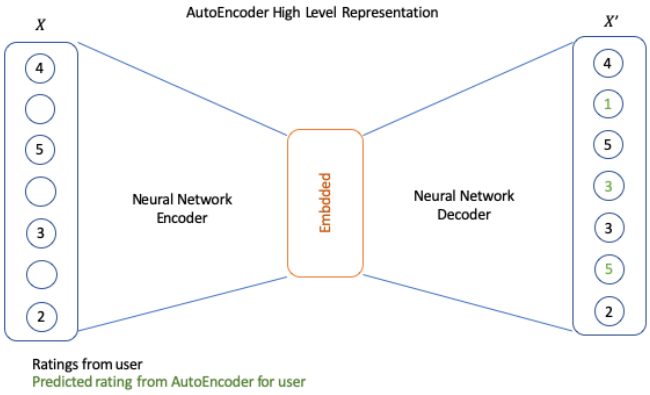

Figure 6: AutoEncoder High Level Representation

Each part is implemented with a neural network. The encoder part start with i neurons representing the number of recipes and finish with e neurones representing the embedding size.The decoder part start with e neurons and finish with i neurons. From literature, encoder and decoder parts are mainly symmetric but this configuration is not mandatory.
In our case, X represents the ratings from a given user k, e.g M_{ki}, the row k in our matrix M_{ui}. Both encoder and decoder Neural Networks are trained jointly with a Mean Squared Error (MSE) loss function at the end of the decoder that consider only the ratings performed by users. The formula is the following : $\sum_u \sum_{i \in R_u} (x_i - x'_i)^2$ with R_u the ratings performed by the user (only them). Once the model is trained, the ratings in $X' = M'_{ki}$ which are not rated in $X = M_{ki}$ by a given user are the estimated ratings from the AutoEncoder model.
For our specific use case, we applied a more restrictive strategy than describe in the data section by reducing the initial M_{ui} matrix ($25K \times 180K$ items) to a more compacted one of size $6K \times 3K$ items by considering only recipes that have at least 20 ratings and users that have performed at least 5 reviews. Then, we split this matrix in two sets, train and test with the same strategy as describe in the data section. After running a first version of the AutoEncoder with this train matrix, we got the following conclusions:

- The precision of the AutoEncoder was inversely proportional to the depth of the encoder/decoder. Finally, with our dataset, we get our best results (MSE) with the shortest path, directly from input to embedding for the encoder and embedding to output for the decoder.
- Even is the MSE was acceptable with a train error of 0.05 and a test error of 0.08 (training on 200 epochs), the model tends to predict rating of 5 for all recipes as almost all ratings in the train set is of 5.
- Considering top 10 recommendations, for several users, we clearly realized that the model provided the same recommendations for all users. Clearly, this model seems providing more in some way the most popular ratings rather than relevant and personalized recommendations.

With these results, our model or input data needed to be redesigned. From the second conclusion, we identify clearly that the partition of the ratings, even if scaled from 0 to 5, is too much unbalanced as most of the ratings lies around 5. To mitigate this, we decided to change the scale from 0 to 5 to {0, 1} by considering all {4, 5} rating as positive {1} and all {0, 1, 2, 3} ratings as negative {0}. Unfortunately, even if this modelisation shows slightly better results with a train error of 0.02 and a test error 0.06, all ratings tends to be closed to 1 and recommandations seems only slightly better personalized.
After a deeper analysis, the train error was clearly good with a value of 0.02 but for sure, a test error of 0.06 for 0 or 1 values is two much. After reflexion, we decided to consider the AutoEncoder and especially the input data not as for any classical machine learning problem by splitting the data into train and test set but by training on the whole data and accept overfitting ! Indeed, this is actually done for other models such as Collaborative Filtering where we consider all data for performing recommandations. With this new attempt, we increase also the number of epoch to 300 and testing several values of embedding size from 64 to 1024. For sure best results were obtained with highest values of embedding but an embedding of 256 neurons reached a reasonable MSE loss of 0.01 for all the data. Looking as the results, they were now personalized and more relevant for Thomas (see Table 4 in annexes).
In term of performances, the AutoEncoder is really quick to train as it takes around 3 minutes to perform the training on 300 epochs. For the predictions, they can be performed within 2 seconds for the 6K users of the M_{ui} matrix. This is a real positive thing as, as we need to train on all data to perform/update predictions, 3 minutes seems an acceptable timing.
At last, there's for sure still some room for improvements. Indeed, there are variations of AutoEncoder, especially the variational AutoEncoder[6] that involved a normal distribution for the embedding. We can also consider the temporality of the rating by giving more weights to recent ratings compare to oldest ones.

6 Conclusion

During this project, we have adopted several approaches to come up with a high end recommender system able to give a large range of suggestion to the users. We started by data visualisation and data pre-processing, which constitute two essential steps to grasp a better understanding of the problem and to generate the desirable clean datasets that will be used by our different models. Then, we implement some relatively simple models such as CBF and Demographic Filtering. Both enable to overcome the cold start problem by estimating the similarity between recipes or promoting popular recipes for all users. Nevertheless, these techniques do not handle well large dimensional data and can't generate diverse nor personalised recommendations.

While CF utilises the similarity between a specific user and all other users to predict ratings, methods like AutoEncoder and matrix factorisation represent the user-recipe matrix into a low dimensional latent space to predict ratings and generalise to unseen recipes. They generate more personalised and diverse recommendations while still being efficient on large amount of data. However, those last methods are more complex to implement and don't handle the cold start problem. This is the reason why we believe that combining some of our most tuned and best performing algorithms on specific areas should offer a great answer to our initial concerns. Indeed, when combined, Demographic filtering, CBF and Matrix factorisation can handle all problems raised correctly and deliver relevant recommendations in most of the cases. Also, we have not mentioned it so far but the filters (new releases, Diet, Quick and Easy) applied for Demographic filtering can and should be applied on some of the most advanced methods, like the Hybrid model. This will enable to generate great recommendations that additionally take into consideration the user's current mood and desires.

As this journey came to an end, we still had some areas we wanted to develop in order to improve our system. In fact, it would be relevant to start using clusters that would provide more diverse recommendations, particularly for the CBF. Furthermore, since recipes' popularity and user preferences may change over time, including temporal dynamics in our matrix factorisation model would probably have benefited our model. A last improvement would be to gather and make use of implicit data. Indeed, we could use external data about the user's browser history, behavioural information, gender etc.. to improve further our suggestions.

7 Annexes

Recipe Name	Rating
Tommy Fries	4

Table 1: Ratings from Gaels

Recipe Name	Rating	Recipe Name	Rating	Recipe Name	Rating
whatever floats your boat brownies	5	dirty shrimp in butter beer sauce	5	lil cheddar meatloaves	5
hamburgers with brown gravy total comfort food aka meat cakes	5	crock pot potato chowder	4	kittencal s best deep dark chocolate layer cake	5
oreo pudding	5	stardust chocolate pancakes	5	southwestern stuffed bell peppers	5
better than sex cake iii	5	honey roasted pork loin	5	southwest chicken black bean soup	5
almost kfc coleslaw	4	the best chocolate cake really	5	pumpkin coffee cake	4
kittencal s chicken crescent roll casserole	4	beef patties in onion gravy	4	quick yellow cake	4
anasazi enchiladas	5	taco cheesecake	5	crock pot old south pulled pork on a bun	5
copycat kfc coleslaw the real thing	5	potato casserole with corn flakes	5	cookie dough cheesecake	5
kittencal s bakery coconut cream pie	5	vanilla sweet potatoes	4	parmesan basil perch	4
brown sugar bundt cake	5	taffy apple pizza	5	nat s oven baked zucchini sticks	5
black bottom banana bars	3	gingerbread gingerbread cake	3	best seller caramel corn	3

Table 2: Ratings from Thomas

#Recommendation	Demographic Filtering	CBF
1	sesame glazed walnut shrimp	Bacon cheddar potato tart michael smith
2	live food crackers	mom's camp fire potatoes
3	scrambled egg casserole	chessy bacon ranch pasta salad
4	basted eggs	slow cooker cheesy bacon ranch potatoes
5	pumpkin white chocolate chip cookies	arkansas breakfast casserole
6	no fail easy cake mix brownies	chessy italian potatoes
7	jo mama s world famous lasagna	contest winner twice baked potato casserole
8	frozen chocolate pb graham cracke	parsnip potatoes and bacon
9	stuffed vienna bread	german fried potatoes
10	banana split muffins	bacon cheeseburger chicken

Table 3: Recommendation for Gaels from our models

# Reco	CBF	CF	Matrix Facto	AutoEncoder	Hybrid
1	grandma kay s vanilla cookies	cashew chicken curry	peaches cream pie	california style chicken	broiled lobster tails for 2
2	mocha twists	sirloin soup italiano	chocolate cake	fresh strawberry cake	fried apples stekte epler
3	cupcake princess vanilla cupcakes	tater tot cups with cheese and eggs	crunchy tossed salad	south beach muffin	applewood farm-house apple fritters
4	vanilla cupcakes w yummy penuche glaze	kittencal s rich homemade beef	paradise mango lemonade	pineapple upside down cake	julia chil s cherry clafouti
5	yummy chocolate crumb cake	pammy s crock pot chicken breast	cole slaw with beans and bacon	chicken welling-ton puff pastry wrapped chicken	fruity grilled cheese sandwich
6	coffee nut scones	chocolate chip mex-ican wedding cakes	bacon mushroom chicken	fried zucchini bat-ter	pizzal bagel bites oamc
7	buttermilk brown-ies with frosting	almost apple pie	paula el paso burg-ers	fresh cream of mushroom soup	spootktacular haloween graveyard cake
8	yummy chocolate cupcakes	roasted cauliflower	baked falafel balls	garlic cheese rolls for bread machine	not your ordinary chocolate chip cookies
9	austrian kaiser-schmarren emperor s pancake	easy grilled cajun chicken	strawberry yougurt pancakes	amish oven fried chicken	nana s chocolate frosting
10	forevermama s old time pancakes	the best peanut butter oatmeal cookies	cinamon pumpkin banana bread	our secret sirloin steak	french onion burgers

Table 4: Recommentation for Thomas from our models

References

[1] Matrix factorization techniques for recommender systems
https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf

[2] Item-based collaborative filtering recommendation algorithms
https://www.researchgate.net/publication/200121014_Item-based_collaborative_filtering_recommendation_algorithmus

[3] Winning the Netflix prize competition: a summary
https://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/

[4] Movie Recommender Systems: Implementation and Performance Evaluation
https://arxiv.org/pdf/1909.12749.pdf

[5] Hybrid Recommender System based on Autoencoders
https://hal.inria.fr/hal-01336912v2/document

[6] Item Recommendation with Variational Autoencoders and Heterogeneous Priors
https://arxiv.org/pdf/1807.06651.pdf

[7] A Neural Network Based Explainable Recommender System
https://arxiv.org/pdf/1812.11740.pdf