Completing partial recipes using item-based collaborative filtering to recommend ingredients

Paula Fermín Cueto
s1776639
s1776639@ed.ac.uk

Meeke Roet
s1771458
s1771458@ed.ac.uk

Agnieszka Slowik
s1778554
s1778554@ed.ac.uk

Abstract

Increased public interest in healthy lifestyles has motivated the study of algorithms that encourage people to follow a healthy diet. Applying collaborative filtering to build recommendation systems in domains where only implicit feedback is available is also a rapidly growing research area. In this report we combine these two trends by developing a recommendation system to suggest ingredients that can be added to a partial recipe. We implement the item-based collaborative filtering algorithm using a high-dimensional, sparse dataset of recipes, which inherently contains only implicit feedback. We explore the effect of different similarity measures and dimensionality reduction on the quality of the recommendations, and find that our best method achieves a recall@10 of circa 40%.

1 Introduction

The application of recommendation systems in the context of recipe and ingredient recommendation is becoming increasingly popular. With obesity being a major health problem across the globe and an increased public interest in healthy lifestyles, the study of algorithms that assist people in following a healthy diet is worthwhile.

Most of the existing models that use recipe data aim to recommend recipes to users based on previous ratings of other users and recipes. For example, Freyne and Berkovsky (2010) developed recommender systems to suggest healthy recipes using neighbourhood-based collaborative filtering models. Although less popular in the literature, the task of predicting combinations of ingredients is also of interest. It is based on the idea that oftentimes, people have to make a shopping list whilst taking into account the products they still have at home. In such cases, the question is what additional ingredients could be bought to make a meal.

There are websites, for example *allrecipes.co.uk* and *epicurious.com*, where one can input a list of ingredients and obtain recipes including those ingredients. However, this requires that a recipe exists in the database containing all the inputted ingredients, which may not be the case if the input is many ingredients. Hence, our goal is to build a recommendation system to suggest ingredients that form a good combination with a set of other ingredients. This recommendation system will treat recipes as *users*, while the ingredients will play the role of the *items* these recipes have 'rated'.

One of the most popular methods for recommendation systems is Collaborative Filtering (CF), which uses a matrix of all items and users to produce recommendations, where typically each element of the matrix represents how much a user likes an item. Such methods are known to require a large volume of rating data to produce meaningful recommendations. We will use a recipe dataset provided by *yummly* for the Kaggle competition '*What's Cooking?*', which contains a large collection of recipes (circa 40k) and a list of ingredients included in each recipe.

Su and Khoshgoftaar (2009) classify CF algorithms into three categories: *memory-based models*, which calculate the similarity between users and items from the entire rating matrix, *model-based*

models, which fit a parametric model to a given matrix of ratings to make predictions, and hybrid models. In the class of model-based algorithms, Matrix Factorisation (MF) is the most popular. MF characterises items and users by vectors of latent factors derived from patterns of item rating. This is the approach used for example by De Clercq et al. (2016) in a setting very similar to ours. They used a variety of MF approaches to predict combinations of ingredients for given partial recipes.

In this project, we explore the other direction: memory-based collaborative filtering (MBCF), which has the advantages that it does not rely on item content, it is easy to implement and new data can be added incrementally, as described by Su and Khoshgoftaar (2009). Memory-based (or neighbourhood-based) models can be classified into two groups: *item-based* approaches, which compute the similarity between items based on users' ratings of those items, and *user-based* approaches, which focus on finding similar users based on the items that they rated, and use those to make predictions. Although user-based models were the first MBCF approach described, they do not scale well to large datasets and they are also not well suited for dealing with highly sparse data. To counter these issues, Sarwar et al. (2001) introduced the concept of item-based collaborative filtering and showed that this approach generally outperforms user-based models. Based on their evidence, we have selected the item-based approach as our preferred method for the task at hand.

One of the main challenges we face is the fact that our ratings are binary, positive-only ('implicit feedback'). Unlike other CF tasks, such as suggesting movies for users based on the ratings that other, similar users have given to other movies, the recipe dataset does not contain explicit user feedback. A given recipe has either rated an ingredient 'positively' (value 1) or 'has not seen it' (value 0), but we do not have negative ratings: a value of 0 for an ingredient in a recipe does not imply that the ingredient would not go well with the rest of the ingredients. The problem of dealing with such implicit feedback is becoming more and more relevant nowadays, because explicit feedback, although more convenient, is not always available, for example when dealing with click or purchase data. Fortunately, it is a well-studied issue. For instance, Aiolli (2013) proposes an asymmetric cosine similarity measure that is well suited for this type of rating data.

We have opted for experimenting with different types of similarity measures to deal with the particularities of our dataset. However, this is also interesting for more general reasons, as the computation of similarity between items or users is a critical component of MBCF techniques and has a big impact on the quality of the predictions. We will thus investigate the effects of using different similarity measures on the quality of the recommendations. Furthermore, we will explore the benefits of applying dimensionality reduction techniques to deal with sparse binary data.

2 Data preparation

Our dataset is the training data from the '*What's Cooking?*' challenge on Kaggle. It contains 39,774 recipes identified by a recipe id and classified into 20 different cuisines. For each recipe, there is a list of its ingredients, and there are 6,782 distinct ingredients in total.

To apply collaborative filtering techniques on this dataset we need to convert it to a rating matrix R of dimension $m \times n$, where m is the number of recipes and n is the number of ingredients. Element r_{ui} will be a binary variable taking value 1 if ingredient i is contained in recipe u and 0 otherwise. This will thus be similar to the feature representation known as 'bag-of-words', which is widely used in natural language processing. We used the `CountVectorizer` class from scikit-learn library to create this representation.

The second preprocessing step is to clean the data from irrelevant recipes and ingredients. Firstly, we removed 3,003 ingredients that appear three times or less, we cleaned the ingredient names by removing special characters and numbers (e.g. '14 oz') and corrected for the fact that some ingredients started with a space.

After these initial pre-processing steps, the list of ingredients was still noisy. The ingredient names included words that do not contain relevant information and only indicated different versions of the same thing. For example, 'light brown sugar' and 'dark brown sugar' should be combined into one ingredient 'brown sugar', and words like 'half' or 'oz' should be removed from the name. We came up with the following approach to combine ingredients according to this intuition:

1. We create a list of smaller words contained in ingredient names by finding longest common substrings between all pairs of ingredients.

2. We simplify the name of each ingredient by keeping only the substrings that appear in more than 30 different ingredients or in more than 1,000 recipes.

By doing this we ensure that, for instance, 'chicken-broth' will not change, since both 'chicken' and 'broth' are dominant (they appear often), but 'adzuki beans' will be replaced with 'beans', because 'adzuki' is very rare. The thresholds were adjusted by qualitative evaluation of the results during trial and error. This process reduced the number of the ingredients by one-third, but the data was still very high-dimensional, and many ingredients were still relatively infrequent.

To counter this issue, and given the large collection of recipes available to us, we removed all ingredients appearing in 250 recipes or less. We also removed ingredients that we considered similar to 'stop words' in natural language processing: 'salt' or 'water' would not be very valuable recommendations for complementing a set of ingredients. Lastly, we removed all recipes that contained two ingredients or less. Recipes with no ingredients at all are errors and recipes with one or two ingredients do not allow for predictions and/or evaluation of these predictions. Our final dataset contains 37,340 recipes and 267 ingredients.

3   Exploratory data analysis

In order to understand the data we are using and to avoid wrong interpretation of future results we perform exploratory data analysis before moving to the collaborative filtering experiments. The data is binary, high-dimensional and sparse (on average, 0.03% of the features in a recipe are non-zero), which makes descriptive statistics and visualisation particularly useful in extracting main characteristics of the population.
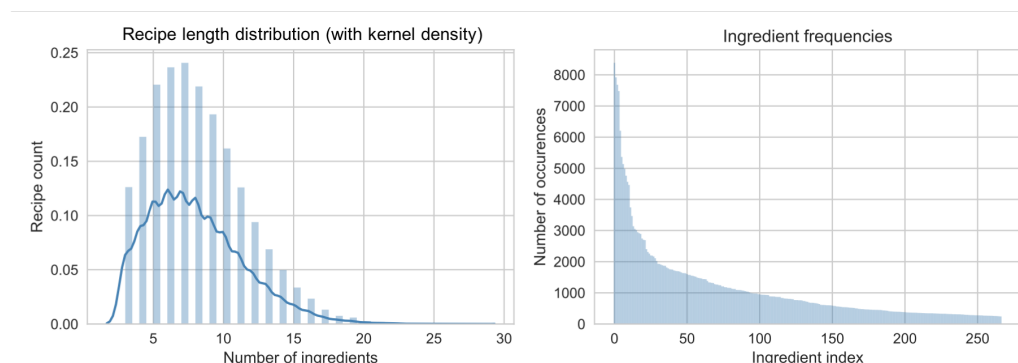

Figure 1: Data distribution.

More than half of the ingredients are present in less than 1,000 recipes (Figure 1, right). There is still a small number of very popular ingredients that can be found in more than 8,000 recipes. Those ingredients are: *butter, tomatoes, garlic, sugar, olive* and *onions*. They are likely to be recommended most often by any model, since they seem to go well with more than 21.42% of recipes in the training set. Those ingredients are also a typical base for an Italian dish, and Italian is the most popular cuisine in the training data.

The number of ingredients in a recipe (Figure 1, left) might also influence the recommendations. Both the median and the mean recipe length are 8.0 ingredients, while the minimum and maximum are 3 and 28, respectively. Recipes with less than 5 ingredients normally include a source of protein, and rice or bread, and they might not showcase all the subtle correlations that can arise when there is a higher range of ingredients to choose from. The recipe length distribution has moderate positive skewness of 0.81 and kurtosis of 0.76, suggesting that while the recipes tend to have a small number of ingredients, there are some outliers. Those recipes might introduce noise and bias to the results of similarity functions, especially to those that are based on the co-occurrences of ingredients: co-occurrence in a recipe of more than 20 ingredients in total might be less indicative of similarity between the ingredients.
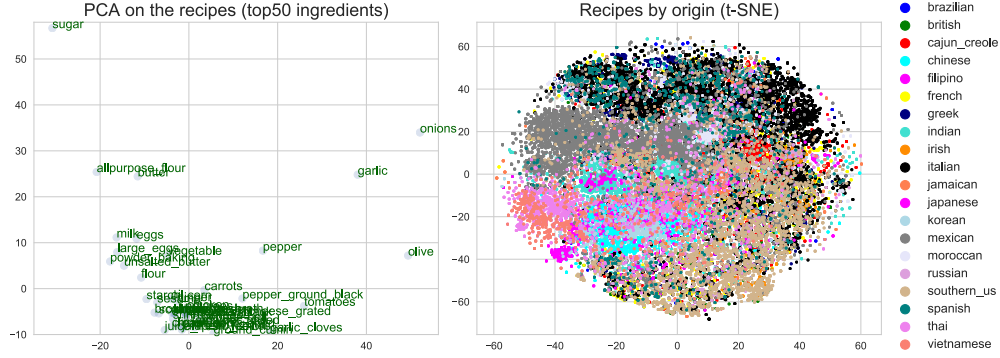
Figure 2: Data visualisation using dimensionality reduction.

We used dimensionality reduction methods to visualise clusters of ingredients based on their occurrences in recipes. For this task, we used Principal Component Analysis on the recipes (Figure 2, left), implemented by using the `PCA` class in scikit-learn library, which carries out Singular Value Decomposition behind the scenes. For better interpretability, we display only the 50 most popular ingredients. The results make intuitive sense. For example, common baking ingredients – *milk, eggs, baking powder, unsalted butter, flour* – appear in a distinct cluster. Moreover, the extremely common ingredients mentioned earlier were plotted far away from the main cluster.

To analyse the similarity between recipes of the same cuisine we tried different techniques to visualise very high-dimensional data in a 2-dimensional representation. First, we implemented PCA to reduce the number of ingredients, but this technique was not very successful. In the most interpretable visualisation we were able to observe that Asian and European were grouped together, but distinct cuisines were not separated. We then experimented with t-distributed Stochastic Neighbor Embedding (t-SNE), a visualisation technique that implements a heavy-tailed distribution in a space of reduced dimensionality, which has been demonstrated to outperform other non-parametric methods in multiple domains with high-dimensional data (Van Der Maaten and Hinton, 2008).

As expected, t-SNE was a more suitable technique for representing our data and it can be seen in Figure 2 (right) that it creates distinct clusters of most cuisines. This representation also captured subtler characteristics of cuisines – for instance, the Indian cuisine is close to the rest of the Asian cuisines (Thai, Vietnamese, Japanese, Korean Filipino, Chinese), but it forms a separate cluster, since there are some ingredients that are very frequent in Indian food but not so common in Western Asian recipes. Similarly, Mediterranean cuisines are grouped in adjacent clusters with Moroccan recipes being clearly separated. We find the biggest overlap in Spanish and Italian recipes, which makes sense, since both make abundant use of olive oil, tomatoes, seafood and certain spices. Most of the recipes come from the Italian (20%), Mexican (16%) and Southern American (11%) cuisines. The recommendations are likely to be biased towards the ingredients used in those cuisines.

4 Methodology

4.1 Item-based collaborative filtering

Item-based CF is implemented in two steps. First, the similarity between all ingredients is calculated: for two ingredients i and j, we calculate the similarity between the two vectors containing their 'ratings'. Then, for each ingredient, the model selects the k most similar ingredients, where the number of neighbours k is a parameter defined by the user. In the second step, the fit $P(u, i)$ between ingredients i and recipe u is predicted ((Sarwar et al., 2001)).

$$P(u, i) = \frac{\sum_{j \in N_i} s_{ij} \cdot r_{uj}}{\sum_{j \in N_i} |s_{ij}|} \tag{1}$$

4

where N_i is the set containing the k neighbours most similar to i, and s_{ij} is the similarity between ingredient i and j. The fit of an ingredient to a recipe is thus obtained by summing the similarities of the target ingredient's closest neighbours that are present in the target recipe and then scaling by the sum of all similarities in the target ingredient's neighbourhood to obtain values between 0 and 1. The fit is 0 if none of the ingredient's closest neighbours appear in a recipe, and 1 if all of its neighbours appear. The more close neighbours of an ingredient appear in a recipe, the higher the predicted fit of the ingredient to that recipe. Closer neighbours will contribute to the fit more, as their higher similarity increases the numerator more. Finally, recommendations for a recipe are made by selecting the N ingredients with the highest predicted fit, with N the desired number of recommendations.

4.2 Similarity measures

One part of our experiment is exploring the effect of different similarity measures on the performance of item-based CF. The first measure was suggested by Sarwar et al. (2001) at the introduction of item-based CF. The **cosine similarity (CS)** between two ingredients i and j is:

$$CS_{ij} = \frac{r_{:i} \cdot r_{:j}}{||r_{:i}||_2 \cdot ||r_{:j}||_2} = \frac{|U(i) \cap U(j)|}{|U(i)|^{1/2}|U(j)|^{1/2}} \tag{2}$$

where $r_{:i}$ is the column vector containing all entries for ingredient i, taken from the user-item matrix R, '$\cdot$' indicates the dot product, and $U(i)$ is the set of recipes containing ingredient i. Because all data is binary, the expression simplifies to the number of recipes containing both ingredient i and j divided by the square root of the number of recipes containing i times the number of recipes containing j (Aiolli, 2013). The cosine similarity is symmetric and efficient to calculate in sparse spaces. The second characteristic is positive for our application, but the first might be problematic (Aiolli, 2013). For example, if a recipe contains a very uncommon ingredient, then common ingredients that are similar will probably go well with the recipe. The opposite, however, does not necessarily hold.

For this reason, Aiolli (2013) proposes the **asymmetric cosine similarity (ACS)**:

$$ACS_{ij} = \frac{|U(i) \cap U(j)|}{|U(i)|^{\alpha}|U(j)|^{1-\alpha}} \tag{3}$$

where α is the asymmetry parameter, which puts more weight on common ingredients as it increases. It is expected that tuning α will provide an advantage in our dataset, because the least common ingredient occurs 250 times and the most common ingredient over 8,384 times (after data preparation). The tuning is done by carrying out a parameter sweep over values from 0 to 0.5, as suggested by the author.

The third similarity measure we investigate is **Jaccard similarity (JS)**:

$$JS_{ij} = \frac{|U(i) \cap U(j)|}{|U(i) \cup U(j)|} \tag{4}$$

The main downside of this measure is that it loses information when the magnitude of ratings is meaningful, because it only counts what has and has not been rated to determine similarity. In our case, this is not a problem, as similarity between ingredients is loosely defined as appearing in the same recipes 'often', which is precisely what Jaccard similarity measures.

Lastly, Bellogín et al. (2011) showed that measures from information theory are also suitable for collaborative filtering. To add variety to the set of similarity measures that is experimented with, we add the **pointwise mutual information (PMI)** to the selection:

$$PMI_{ij} = log\frac{p(i, j)}{p(i)p(j)} \tag{5}$$

where $p(i, j)$ is the probability of co-occurrence of ingredient i and j in a recipe, and $p(i)$ is the probability of ingredient i being present. The PMI measures how often ingredients are used together relative to their frequency of usage in general. It is therefore a very intuitive measure for our purpose. One downside is that it is symmetric, which could reduce its performance as explained earlier.

4.3 Principal Component Analysis

Candillier et al. (1997) describes how dimensionality reduction techniques such as Principal Component Analysis (PCA) can be applied to neighbourhood-based CF to obtain results of better quality with

less computational effort. The reason behind this is that using PCA to reduce the dimensionality of the data speeds up the computation of the similarity matrix and makes these computations more robust, as it enables the transformation of a highly sparse representation to a dense, low-dimensional model of latent factors. Given the high dimensionality of our ratings data, especially the number of recipes, we expect that both these benefits apply in our case. Therefore, the second part of our experiment is to apply the methodology outlined in the previous sections both on the dataset immediately after preprocessing, and after applying PCA to the rows (recipes) and compare the results.

Once again we use the `PCA` class in scikit-learn library to obtain a transformed representation of the recipe space consisting of 267 principal components, as many as there are ingredients. This cannot be more, because the maximum number of principal components is the rank of a matrix, in our case the number of ingredients.

4.4   Number of neighbours

The number of neighbours that is used for predicting the fit of ingredients to recipes can have a great influence on the performance of CF. If k is too low, the predictions will have a high variance and be biased towards only a few instances. If k is too high, the predicted fits will all be very similar, so the ranking will be susceptible to small changes. Therefore, we optimize k separately for each similarity measure and original/PCA-reduced data by performing a parameter sweep over the set $\{10, 20, 50, 100, 150, 200\}$ using 10% of the dataset. This should suffice, as we do not need to 'train' our model in a classical sense to obtain accurate parameters, but only need a ranking of the performance under different values of k.

4.5   Evaluation

The quality of the recommendations under different parameters, similarity measures, and original/PCA-reduced data is evaluated by means of a leave-one-out cross validation scheme. We select one recipe at a time, which we take out of the dataset. Then, we remove one ingredient at random from the recipe in question and make recommendations for the partial recipe based on the remaining dataset. Of interest is how good the recommendation algorithm is at recovering the missing ingredient. Following De Clercq et al. (2016), this is assessed using three metrics:

1. Percentage of recipes for which the missing ingredient is among the top-10 recommended ingredients, also called recall@10 (the logic being that in a practical application of our work, ten ingredients would be a reasonable number of recommendations to make to a user looking for suggestions).

2. Mean rank of the missing ingredients in the list of recommended ingredients.

3. Median rank of the missing ingredients in the list of recommended ingredients.

The first metric should be as high as possible, the other two as low as possible. After experimenting with different parameter settings on 10% of the data, we select the most promising configurations and evaluate their performance using the other 90% of the dataset as a test set. In addition to the quantitative metrics described above, we will also sample a few recipes and judge the corresponding recommendations qualitatively, simulating real-world use of our system.

5   Results

Experimenting with the parameter α for the asymmetric cosine similarity showed that the best performance was obtained when setting $\alpha = 0.05$. Figure 3 plots the results of tuning k for all similarity measures and the original/PCA-reduced data. We decided to show the median rank here as it is more robust against outliers, but the mean rank showed similar patterns.

It can be seen that the approaches using the PCA-reduced data clearly and consistently outperform those using the non-reduced data. Hence, we continue the analysis using only the PCA-reduced data. It can also be seen that, after applying dimensionality reduction, the asymmetric cosine outperforms the cosine as we expected. Another observation is that the algorithm becomes less sensitive to the number of neighbours after PCA. This makes sense, because with sparse binary data, the similarities between ingredients are overconfident, but in a dense representation the ratings are smooth and

additional neighbours have a subtler impact. Also, the experiments with the non-reduced data took 141 minutes to run, whereas those using the PCA-reduced data took only 12 minutes. PCA is thus a substantial improvement in speed, quality, and robustness. We select $k = 50$ for all similarity measures, since 50 makes a slightly better impression than 100 in the plots and it is more sensible in proportion to the total number of ingredients.
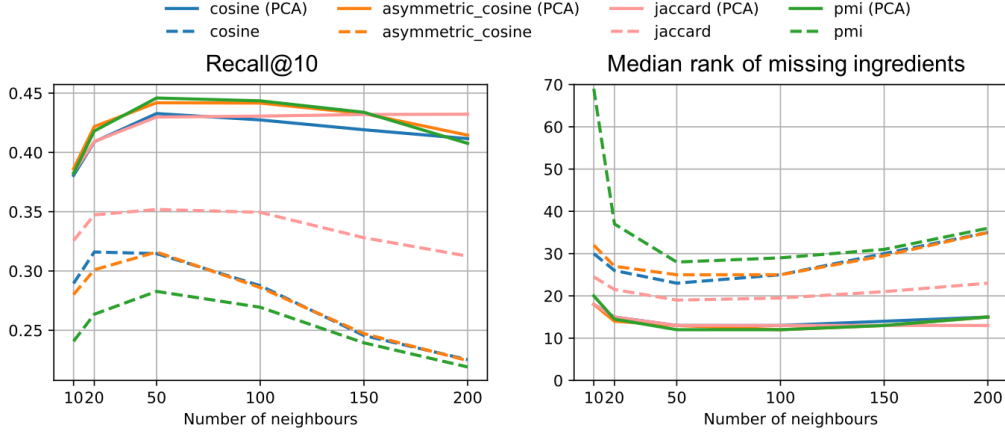

Figure 3: Effect of number of neighbours and PCA on recall@10 and median rank performance.

Table 1 shows the results of all similarity measures with the PCA-reduced test data using $k = 50$. The metrics are all close together. Based on the recall@10 and median rank, PMI and asymmetric cosine appear to perform best, and the cosine and Jaccard similarities perform slightly worse. Interestingly enough, the mean rank of PMI is the worst of the four, so it may have had a few more cases where it was clearly off, whereas the simple cosine does best on this. Depending on the specific application, any of the similarity measures could be useful. In our case, we would pick PMI, putting most weight on the robust metrics recall@10 and median rank.

Similarity measure	Recall@10	Mean rank	Median rank
Cosine	39.26%	40.91	16
Asymmetric cosine	40.16%	41.69	15
Jaccard	39.26%	41.22	16
PMI	40.25%	42.73	15

Table 1: Evaluation of similarity measures with $k = 50$ and PCA-reduced test data.

The work most similar to ours is De Clercq et al. (2016), who obtained a recall@10 of 57.5% with two-step recursive least squares (RLS), and 43.6% with non-negative matrix factorisation (NNMF) on a dataset with more ingredients, but also substantially more recipes. The higher score of RLS can be explained by the fact that it incorporated additional information about the ingredients that we did not have available, such as flavour and texture. NNMF did not do this, and still performs slightly better than our best model. We attribute this to a difference in cleanliness of the datasets. For example, even after preprocessing, our dataset still contained some duplicate ingredients, such as 'tortilla' and 'corn tortilla'. We expect that solving such issues would improve the results further and close the gap to De Clercq et al.'s NNMF approach.

Qualitative evaluation of our results strengthens this belief. Table 2 shows two random recipes from our dataset and the corresponding recommendations produced by our best model. The 'corn versus corn tortilla' issue shows for example with the cilantro, chicken and garlic ingredients. The algorithm could be improved by filtering such ingredients from the list of recommendations. Apart from this, however, the table paints a positive picture, because the recommendations make intuitive sense. It is interesting to see that although we did not provide the model with the recipe's cuisine, it seems to implicitly detect this and recommend related ingredients (avocado, lime, sour cream, corn and salsa are all very typical Mexican ingredients; chillies, chicken, potatoes and cumin are typical for an Indian curry).

Recipe 16903 (Mexican)	Current ingredients	cheddar_cheese, jalapeno_chilies, lettuce, lime, pork, purple_onion, peppers , olive, cilantro_chopped_fresh, pepper_ground_black tortillas_corn
	Recommended combinations	*avocado, cilantro, tomatoes, juice_fresh_lime, kosher_salt, cream_sour, cilantro_fresh, corn, salsa, ground_cumin*
Recipe 13162 (Indian)	Current ingredients	butter, cayenne_pepper, cream, garlic_paste, ground_cumin, masala, milk, oil, onions, shallots, pepper_black, skinless_boneless_chicken, powder_chili, yogurt
	Recommended combinations	*paprika, chilies_green, coriander_ground, chicken, flour, potatoes, cilantro_leaves, pepper, garlic, cumin_seed*

Table 2: Samples of recommendations using PMI similarity and PCA-reduced data with $k = 50$.

6 Conclusions

This project investigated the use of item-based collaborative filtering to recommend ingredients to partial recipes using sparse, high-dimensional data. We implemented four different similarity measures and used our algorithm directly on the preprocessed data as well as after an additional dimensionality reduction step. We found that our algorithms produce very intuitive recommendations. Reducing the number of recipes through PCA improves the quality of the recommendations substantially, while cutting the runtime by a factor 10 and making the results more robust against changes in the number of neighbours and the similarity measure. The best similarity measure depends on the requirements of a specific application. Our best method, using similarity measure PMI with $k = 50$ and PCA, achieves a recall@10 of 40%. This is slightly worse than the non-negative matrix factorisation approach taken by De Clercq et al. (2016), but we believe that this gap can be closed, and potentially turned around to our advantage, by improving the quality of the dataset.

References

Aiolli, F. (2013). Efficient top-n recommendation for very large scale binary rated datasets. In *Proceedings of the 7th ACM conference on Recommender systems - RecSys '13*, pages 273–280.

Bellogín, A., Wang, J., and Castells, P. (2011). Text retrieval methods for item ranking in collaborative filtering. In Clough, P., Foley, C., Gurrin, C., Jones, G. J. F., Kraaij, W., Lee, H., and Mudoch, V., editors, *Advances in Information Retrieval*, pages 301–306, Berlin, Heidelberg. Springer Berlin Heidelberg.

Candillier, L., Jack, K., Fessant, F., and Meyer, F. (1997). *Recommender systems*, volume 40.

De Clercq, M., Stock, M., De Baets, B., and Waegeman, W. (2016). Data-driven recipe completion using machine learning methods. *Trends in Food Science and Technology*, 49:1–13.

Freyne, J. and Berkovsky, S. (2010). Intelligent Food Planning: Personalized Recipe Recommendation. In *IUI '10 Proceedings of the 15th international conference on Intelligent user interfaces*, pages 321–324.

Sarwar, B., Karypis, G., Konstan, J., and Reidl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the tenth international conference on World Wide Web - WWW '01*, pages 285–295.

Su, X. and Khoshgoftaar, T. M. (2009). A Survey of Collaborative Filtering Techniques. *Advances in Artificial Intelligence*, 2009(Section 3):1–19.

Van Der Maaten, L. J. P. and Hinton, G. E. (2008). Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605.

Contribution of team members

- **Agnieszka Slowik**:
  - Report: Exploratory data analysis.
  - Code: Exploratory data analysis, evaluation metrics, exploration of alternative methods that did not make the cut.
- **Meeke Roet**:
  - Report: Methodology, results (text).
  - Code: Similarity measures, collaborative filtering algorithm, support with data preparation.
- **Paula Fermin Cueto**:
  - Report: Introduction, data preparation, results (plots and tables).
  - Code: Data preparation, experiments + processing, support with similarity measures.

Literature review, code validation, editing and proofreading were carried out by everyone.