Machine Learning Applications

Final Project

100489210 | Xin Ying Leong
100496657 | Adeline Poncet
100496636 | Blanca Sánchez

DATASET CREATION

The dataset was prepared by scraping from the website www.allrecipes.com using the Python library Selenium. The starting point of the web scraping is in the website's cuisine page where we can collect the links to all types of cuisines ordered by their initial alphabet. These cuisine links are then used, one by one, to collect the links to each recipe for the cuisine. The recipe links are then saved into a data frame which is then saved as a .csv file. The collection of these links were rather quick, so the next step is to collect the details and reviews of each recipe via the recipe links.

In this phase, we analyzed both the visible recipe information on the website and the underlying html source code to determine which data fields to extract. We defined two sets of fields: one for scraping recipe metadata and another for extracting individual review information. For each stored recipe link, the script accesses the page and dynamically loads all available reviews, or as many as allowed by our defined scroll limit (300 scrolls). The process then follows two main steps: first, it extracts the recipe metadata and saves it to a CSV file; second, it retrieves the review data and stores it in a separate CSV file. Each entry includes appropriate identifiers to ensure that reviews are correctly linked to their corresponding recipe.

The scraping of each recipe web page took a significant amount of hours, as most recipes have at least 200 reviews. Therefore, with around 800 recipes retrieved, and about 110,000 reviews collected in total, we stopped scraping.

DISCLAIMER: We have utilized generative AI as a tool to help us in writing the necessary code to scrape data. We have used those code snippets to adjust to our requirements of our dataset.

Note: the notebook used to collect data is implemented in 'AllRecipes_Scrape.ipynb'.

TASK 1

To begin with, we loaded a DataFrame containing the collected reviews. Each review includes, among other fields, the username of the reviewer, a star rating from 1 to 5, and, in most cases, a written text.

Text Preprocessing

We first removed samples that did not contain any textual content and only had a star rating, as these were not useful for our NLP analysis. After this filtering step, we retained over 110,000 reviews.

Next, we applied lemmatization to the review texts. Initially, we tested both SpaCy and NLTK pipelines. After comparing the results, we chose to keep NLTK, as it produced, in our opinion, higher-quality lemmas.

Fine-tuning the pipeline required multiple attempts through the NLP process. In the final version, we included the following steps:

- Removal of HTML tags using BeautifulSoup, as many reviews contained embedded HTML.
- Removal of URLs via regular expressions.
- Expansion of contractions to ensure consistency in tokenization.
- Tokenization using NLTK's `word_tokenize`, followed by POS tagging.

In the beginning, we retained major parts of speech (nouns, verbs, adjectives, and adverbs). However, in a later stage, specifically for topic modeling, we excluded adverbs, as we found that they interfere with the clarity and coherence of the extracted topics.

Lemmatization was then performed using NLTK's `WordNetLemmatizer`. We remove standard English stopwords provided by the NLTK library, along with a custom list of additional stopwords identified during iterative testing. These custom stopwords frequently appear but add no value to the analysis of the reviews, such as "without", "enough", "would", "could", and so on. In addition, we decided to remove all digits, as recurring numbers in the text did not contribute meaningful information.

We encapsulated the full preprocessing pipeline into a reusable function, which we applied to the DataFrame using the Swifter module for efficiency, by parallelizing over the CPU cores.

Throughout the notebook, we saved intermediate results to CSV files to facilitate recovery and reuse in future runs. This allowed us to resume work from any main section of the notebook in a new runtime, without needing to re-execute code from earlier sections, thus saving us a significant amount of time.

Note: Spacy pipeline can be seen on Notebook 'MLA Project_NLP.ipynb'.

Text Vectorization

With the cleaned lemmas prepared, we proceeded to the vectorization phase. We first generated the corpus and performed bigram detection, iterating multiple times to fine-tune the `min_count` and `threshold` parameters (final values: 25 and 40, respectively). This process yielded a meaningful set of bigrams, which we incorporated into the Gensim Dictionary. However, their overall frequency remained too low to be seen in later stages such as t-SNE visualizations and topic modeling. As an example, we can cite: 'cottage_cheese', 'sour_cream', 'dijon_mustard', 'deep_fry' or 'bread_machine'. Overall, the bigrams cover ingredients, brands, techniques and tools, along with some that are not directly related to cooking but that are retained due to their frequency.

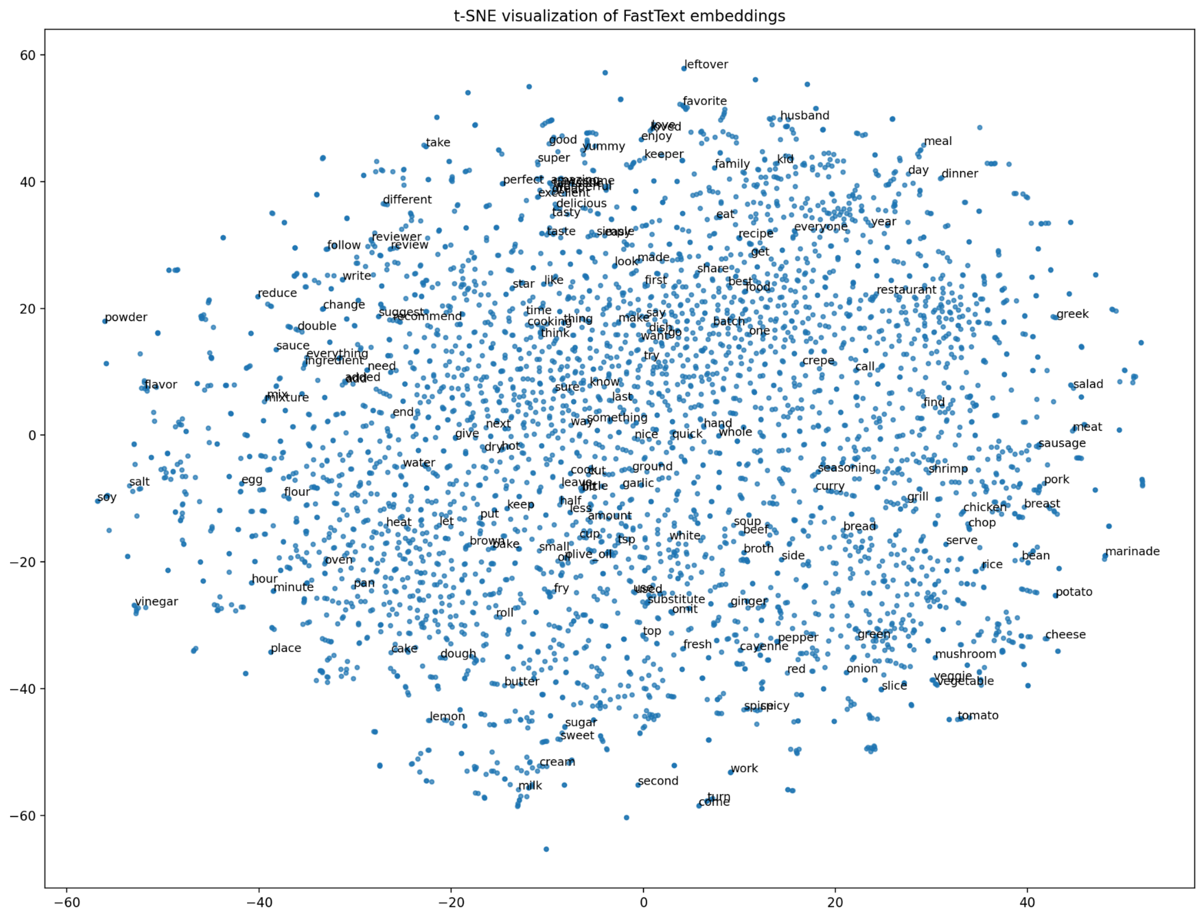
After constructing the Gensim Dictionary, we computed both BoW and TF-IDF representations of the corpus. The BoW representation was stored in a `MmCorpus` object to enable efficient access during later stages, particularly for running the LDA topic modeling algorithm.

Regarding the models, we explored several options, all using word embeddings with a vector size of 300 features:

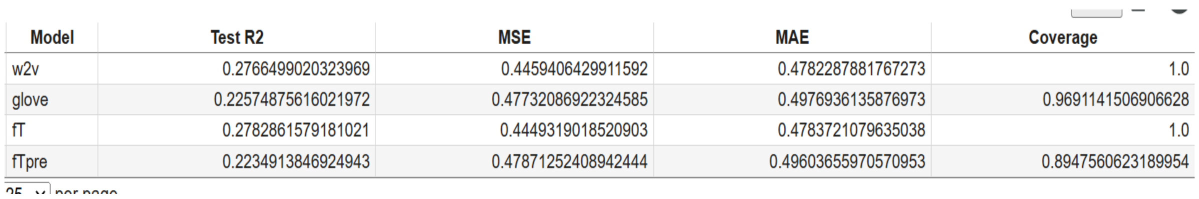
- Word2Vec (`w2v`)
- Pre-trained GLoVe (`glove`)
- FastText (`fT`)
- Pre-trained FastText (`fTpre`)

To compute the embedding for each review, we averaged the word vectors of its cleaned lemmas. Specifically, for each review, we extracted the embedding of every token present in the vocabulary of the selected model and we computed the mean of all available word vectors to obtain a single fixed-size review representation.

Lastly, to compare the resulting embeddings, we generated t-SNE visualizations for both Word2Vec and FastText embeddings, focusing on the most relevant words. While the overall structure appeared similar, we found that FastText produced slightly more coherent and meaningful word neighborhoods.



To support our observations with a more objective evaluation, we conducted a ridge regression analysis to predict the rating associated with a review using the embeddings from the 4 models mentioned above:



Model	Test R2	MSE	MAE	Coverage
w2v	0.2766499020323969	0.4459406429911592	0.4782287881767273	1.0
glove	0.22574875616021972	0.47732086922324585	0.4976936135876973	0.9691141506906628
fT	0.2782861579181021	0.4449319018520903	0.4783721079635038	1.0
fTpre	0.2234913846924943	0.47871252408942444	0.49603655970570953	0.8947560623189954

Taking into account the obtained results, it seems that the FastText models have gotten better R-squared and MSE scores, so we opted to proceed with the text embeddings of FastText over alternatives like Word2Vec or GloVe. Furthermore, FastText can generate embeddings for out-of-vocabulary (OOV) words by leveraging subword information, which is a feature particularly valuable in domains with specialized or uncommon vocabulary. Regarding the decision between using a pre-trained model versus a domain-specific one, we believe that training the model on our recipe corpus allows the embeddings to better capture the semantic nuances and context-specific terminology used in cooking-related texts, making the representations more meaningful for our tasks. This can be justified by the coverage that the FastText models have over our corpus.

Finally, we saved the reviews, along with their cleaned lemmas and corresponding embeddings, into a CSV file, for later use.

Topic Modeling

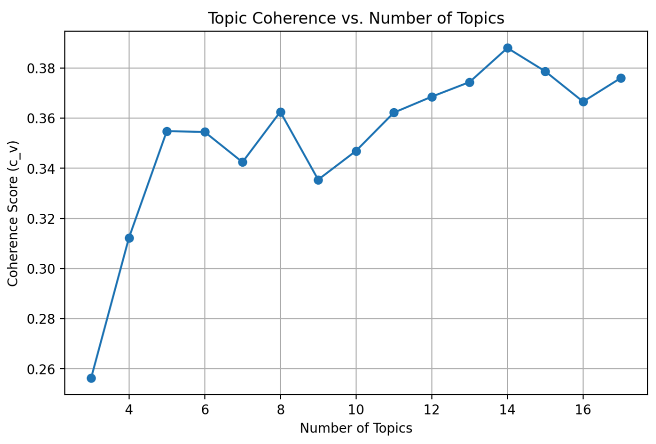
The idea behind the Topic Modeling approach we ended up using is that individual reviews are not the best type of document for this analysis. This is because the reviews, when looked at together, do not really form a consistent or coherent set, something we noticed from our initial results applying the LDA algorithm. Therefore, we decided to treat each recipe as a single document by aggregating the lemmas from all its reviews.

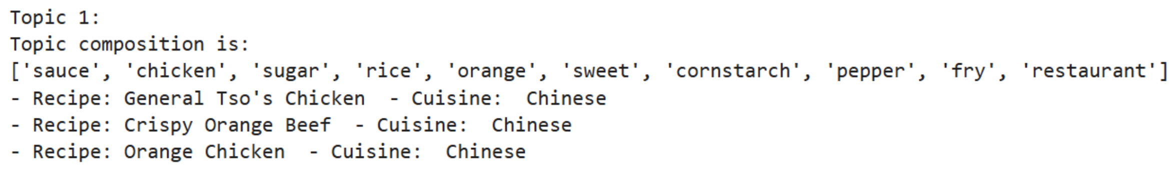
We created a new corpus based on the recipe, constructed the corresponding Gensim Dictionary, and computed the BoW representation. With this setup, we began building the LDA model.

From our initial results, we confirmed, as previously mentioned, that adverbs introduced noise into the topics. Consequently, we updated the lemmatization function to exclude them and ran the whole NLP process again. We then ran LDA with five topics and reviewed the outputs. Early topic distributions included overly generic words that lacked discriminative value, so we iteratively refined our preprocessing by introducing a custom list of stopwords. Words like 'dish', 'cook', 'taste', and 'flavor' were removed in subsequent runs. This iterative filtering process led to more meaningful and coherent topics. To guide us along this process, we made use of the top 10 terms per topic plot and the pyLDAvis graph.

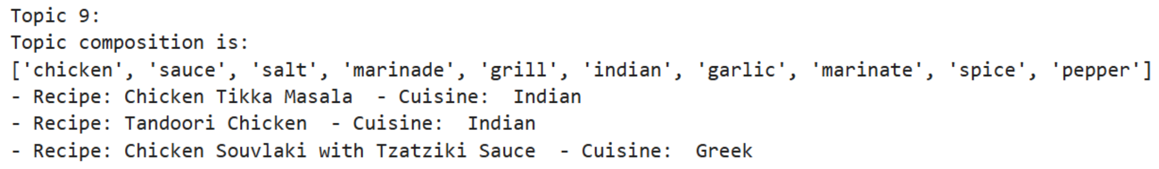
Concerning the model selection, we computed topic coherence scores and experimented with the number of topics suggested by the coherence model.

The resulting topics appeared interpretable, so we extracted the most representative documents for each one. Crossing this data with the cuisine labels obtained during scraping revealed logical groupings: some topics aligned with specific cuisines (e.g., Greek, French, Cuban, Chinese), others with food types (e.g., Soup, Salad, Dessert, Bread/Dough), and some with cooking techniques (e.g., Marinade).

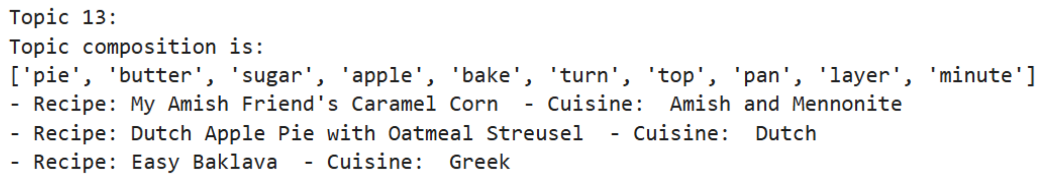




```
Topic 1:
Topic composition is:
['sauce', 'chicken', 'sugar', 'rice', 'orange', 'sweet', 'cornstarch', 'pepper', 'fry', 'restaurant']
- Recipe: General Tso's Chicken  - Cuisine:  Chinese
- Recipe: Crispy Orange Beef  - Cuisine:  Chinese
- Recipe: Orange Chicken  - Cuisine:  Chinese
```



```
Topic 9:
Topic composition is:
['chicken', 'sauce', 'salt', 'marinade', 'grill', 'indian', 'garlic', 'marinate', 'spice', 'pepper']
- Recipe: Chicken Tikka Masala  - Cuisine:  Indian
- Recipe: Tandoori Chicken  - Cuisine:  Indian
- Recipe: Chicken Souvlaki with Tzatziki Sauce  - Cuisine:  Greek
```



```
Topic 13:
Topic composition is:
['pie', 'butter', 'sugar', 'apple', 'bake', 'turn', 'top', 'pan', 'layer', 'minute']
- Recipe: My Amish Friend's Caramel Corn  - Cuisine:  Amish and Mennonite
- Recipe: Dutch Apple Pie with Oatmeal Streusel  - Cuisine:  Dutch
- Recipe: Easy Baklava  - Cuisine:  Greek
```

Overall, the topic modeling process was labor-intensive but ultimately rewarding, as the final results revealed interpretable patterns.

Note: different experimentations can be found on Notebook 'MLA Project_Topic Modeling.ipynb'.

TASK 2

For this task, we have decided to implement a recommender system that recommends users with dishes (recipe titles) that may be of interest to them, by leveraging the information of their interactions on the website, i.e., reviews and ratings given to recipes. We have decided to implement this instead of the other possible ML tasks as this is something new for us, and we wanted to take on the challenge.

Given that the available user data is limited to usernames and sparse feedback, as many users have rated or reviewed only a small number of recipes, we decided that a content-based recommender system was the most suitable approach. Furthermore, to leverage the word embedding component of the project, we chose to use embeddings to compute recipe similarities, rather than relying on recipe metadata.

Consequently, the first step we took was to compute the word embeddings (WE) for the recipes. We explored two approaches:

- Training a new FastText model on the recipe corpus created during topic modeling, and then generating a vector representation for each recipe.

- Calculating the recipe WE as a weighted average of the reviews embeddings from the first part of the project, and using TF-IDF scores as weights, with the idea to emphasize more informative terms.

Both approaches produced similar results; in this report, we present findings based on the first method. The second approach and the comparison between both can be found on Notebook 'MLA Project_RS.ipynb'.

Following that, we analyzed the review dataset and confirmed that many users had only a few contributions, while a smaller group had a significantly higher number of reviews, as expected from theory class. We identified that the largest number of reviews came from the username 'Anonymous', which likely does not represent a single individual. Therefore, we excluded these reviews from the dataset.

Next, we constructed a user DataFrame by assigning a unique ID to each user. To finalize the data preparation phase, we applied a filtering step to retain only users and recipes with at least five reviews each, ensuring a more reliable dataset for subsequent analysis

In accordance with the project requirements, we implemented two approaches to a content-based recommendation system: one based on recipe-to-recipe similarity using content features (in this case, word embeddings), and another using collaborative filtering principles constrained to content-based data, that is, looking to predict ratings for user-recipe pairs (*u, i*), by leveraging the ratings of recipes similar to *i* that have been rated by the same users who rated *i*, and also by user *u*.

To implement the content-based recommender system, we relied on the similarity between recipe word embeddings, as previously described. Specifically, we used cosine similarity to measure the closeness between recipes. This metric captures how aligned the content of two recipe embeddings are, based on the language used in their reviews. Since cosine similarity considers only the angle between vectors and not their magnitude, it is especially suitable in this context, as recipes vary in the number of reviews. Then, this approach ensures that semantic similarity is evaluated independently of review length or word count, focusing solely in the direction of the content in the embedding space.

To evaluate the results, we generated recommendations for a given user by retrieving the recipes they rated with 4 or 5 stars, and then identifying the three most similar recipes based on the similarity scores. For example:

```
Recommended recipes similar to Brazilian Chicken with Coconut Milk:
  - Indian Chicken Curry (Similarity: 0.992)
  - Jamaican Style Curry Chicken (Similarity: 0.992)
  - Brazilian Fish Stew (Similarity: 0.992)
```

Moreover, in order to compare the results with the Collaborative Filtering approach, we implemented a function that calculates the rating of a pair (*u, i*) according to the formula:

$$\hat{r}_{ui} = \frac{\sum_{j \in N_K(i,u)} \text{sim}(i,j) \cdot r_{uj}}{\sum_{j \in N_K(i,u)} \text{sim}(i,j)}$$

On the other hand, we implemented the content-based collaborative filtering recommendation system using the ***Surprise*** library. We took advantage of its built-in tools for evaluation and data manipulation along with its easy implementation of algorithms for recommendation systems.

First of all, the dataset used consists of user-item-rating triplets. The following format is required by Surprise: a Dataset object consisting of (user, item, rating) tuples. We utilized the Reader class to define the rating scale and parsed our dataset accordingly.

As we have to train and test the models, the previous data is shuffled to ensure randomness and is manually splitted. 75% of the data is used for training and the remaining for testing.

For model selection, two versions for evaluation have been used:

1. Manual Evaluation Loop

The code implementation trains a KNNBasic algorithm on the train set, using content-based collaborative filtering with `user_based = False` and cosine similarity.

In each iteration, we tried a specified number of neighbors, k, and then evaluated the model with the RMSE metric for the predictions.

2. Grid Search with Cross-Validation

The class `GridSearchCV` automates hyperparameter tuning using cross-validation, which makes things simpler. Here, different techniques were used with different hyperparameters, while always specifying content-based with cosine similarity.

Some of the techniques used were:

- **SVD (Singular Value Decomposition)** algorithm from Surprise, which is a matrix factorization technique widely used in collaborative filtering. SVD decomposes the user-item interaction matrix into latent features, helping the model predict ratings for unseen items based on learned patterns.

- **KNN** algorithm from Surprise, which is derived from a basic nearest neighbors approach and where the actual number of neighbors that are aggregated to compute an estimation are fine tuned. This technique can be performed in its basic form (KNNBasic), mean correction (KNNWithMeans) and mean and standard deviation correction (KNNWithZScore).

For model evaluation, the models were tested and then with the help of the class *accuracy* from Surprise, the RMSE and MAE scores of every model were calculated. As expected, the best technique to use in this case was the SVD. This is due to the fact that latent factors could capture deeper, hidden relationships between users and recipes. It also handles large datasets more effectively and offers better scalability compared to KNN, which is perfect for sparse datasets like what we have. Having said that, SVDs are better in this case but we also acknowledge that KNN models have advantages like its interpretability (we can directly observe which users or items that are being used to make predictions) and good performance with small datasets with no sparsity.

As previously mentioned, the RMSE and MAE scores of the models were calculated. The scores of the SVD model are shown below. These values suggest that the prediction of ratings on a scale of 1 to 5 is fairly good, for a recommender system.

```
RMSE: 0.7477
MAE:  0.5455
```

Comparing the models

When comparing the models, it is evident that the model created with the **Surprise** library (collaborative filtering systems) performs better than the Neighborhood based version (content-based) as its values for both RMSE and MAE are better.

```
Neighborhood based version:
RMSE: 0.8058
MAE: 0.5770

Recommended version Surprise:
RMSE: 0.7477238318164303
MAE: 0.5455394018305509
```

Other than the above, we can evaluate the models depending on the type of users whose ratings are being predicted. For example, user 20 with reviews that were not always associated with a 5 star rating makes its predictions more difficult. Based on this, the predictions differ significantly from one model to the other. However, for users such as 21 with reviews that are always associated with a 5 star rating, the predictions are more similar.

```
Predicted rating of 20 user to 100 recipe
Neighborhood based version: 3.1927330783942014
SURPRISE using SVD: 3.9055178273213076



Predicted rating of 21 user to 100 recipe
Neighborhood based version: 4.196029284339703
SURPRISE using SVD: 4.472900786740111
```

DASHBOARD

The dashboard is created and runs with a Jupyter notebook using the Dash library. The main contents of the dashboard are:

1. LDA topic analysis with the PyLDAvis library.
2. Reduced embedding space visualization.
3. Interactive recommender system.

These contents are displayed in the dashboard using a third party library Dash Bootstrap Components. The reason behind using this library instead of the others like the Dash Design Kit or the Dash Mantine Components is due to our prior experience with Bootstrap in a previous course. Moreover, each of these main contents are displayed in a tab within the dashboard.

LDA Topic Analysis

Using our previously trained Gensim LDA topic model and the PyLDAvis module, we generated an interactive HTML visualization of the discovered topics, which has been integrated into the dashboard. Additionally, we implemented a feature that allows users to retrieve the 10 most relevant recipes corresponding to the identified LDA topics. The function used for the retrieval of the relevant recipes is the one created in the Topic Modeling section, but adapted for the dashboard.

Reduced Embedding Space Visualization

An interactive 3D plot of the word vectors from the previously trained FastText model is integrated into the dashboard on the second tab. As the original embedding dimension is of 300, we have performed PCA on them to reduce the dimensionality to just 3 components. We have used the PCA algorithm from the Sklearn module for this task.

The overall computational complexity for this tab is rather high. Therefore, to reduce computational complexity of plotting all the word vectors (around 4000 words), we randomly sample 300 of them for the plot. Even so, the plot still takes some time to be displayed. On the right panel, there are 3 types of interactions which would also require some patience from its users:

- Find 50 most similar words given an input word.
- Visualize all ngrams of the LDA topic model.
- Perform semantic operations, with the results displayed in the plot.

Interactive Recommender System

In this tab, users get a glimpse of our recommender system that recommends a set of recipes that they may like depending on their selection of a set of the most popular recipes. By choosing multiple of those popular recipes in the dashboard, a set of recommendations similar to their preferences would be displayed.

DISCLAIMER: The creation of this dashboard had included the use of generative AI for snippets of code (especially for the callback functions) which are then adapted to our own code, and also as a tool for debugging.

REFERENCES

The project was based on the following notebooks seen in Theory and Lab classes, in particular:

- Spacy notebook, by: Jerónimo Arenas-García
- Text Vectorization I notebook, by: Jerónimo Arenas-García, Lorena Calvo-Bartolomé and Jesús Cid-Suero.
- Text Vectorization II notebook, by: Lorena Calvo-Bartolomé
- Topic Modeling notebook, by: Jerónimo Arenas-García and Jesús Cid-Suero.
- Recommender Systems, by: Vanessa Gómez Verdejo.

Moreover, we used documentations from:

- scikit Learn.
- Gensim LDA.
- Surprise package.
- Dash Tutorial.
- Dash Bootstrap Components.

Finally, as noted in the report, we developed the web scraping code using the Selenium library in collaboration with ChatGPT, since we had no prior experience with web scraping. The use of ChatGPT was also present for the creation of the Dashboard.

GOOGLE DRIVE REPOSITORY

Any datasets or notebooks mentioned as well as saved models, dictionaries and corpus can be found in the following Google Drive Folder:

https://drive.google.com/drive/folders/1B7VleXdr6uv0l7yxgVi_s7m4D3TytdcT?usp=drive_link

Moreover, most files, including Final Notebook and Dashboard Notebook are on a GitHub repository:

https://github.com/berlin0489210/ML_Project

However, files over 100 MB could not be unloaded into GitHub, these are their names:

- recipe_reviews_embeddings.csv
- model_fastText_grouped.wordvectors.vectors_ngrams