ChefBot: the bot which finds recipes for you!

Elisa Degara (3176276), Tommaso Giacomello (3173616)

Motivation

As two university students far from home, we always struggle to find some new, tasty recipes with the few ingredients we have. So, instead of heating up a frozen pizza as usual, we decided to build a bot telling us finally what to eat, given some ingredients and, since we want to stay fit, also dietary metrics. So we can have the perfect helper at all times!

Data

Firstly, before diving into the technical part, we should find a dataset full of recipes that will represent the knowledge of our ChefBot.
For this, we rely on the Kaggle Food.com recipes and interaction dataset [2]. In the file `RAW_recipes.csv` we have more than 200.000 recipes, with the respective IDs, ingredients, steps, description and all dietary metrics: calories (cal), time (minutes), proteins (PDV), etc. Here, PDV stands for "Percentage Daily Value", that is the percentage of the suggested daily intake of each nutrient contained in the recipe.
The raw file has been processed, filtering more than 50,000 recipes. Subsequently, the data were tokenized using the GPT subword tokenizer, assigning an ID to each ingredient. The processed data were saved into the file `PP_recipes.csv`. In ordet to speed up the predictive step, ingredients' names were simplified and mapped to approximately 8,000 distinct entries, all stored in the `ingr_map.pkl`. An example of the simplification procedure was to substitute "ice lettuce", "red lettuce" both with the word "lettuce". Lastly, the first two files were merged for speeding inferring procedure in the file `Recipe_final.csv`, which along with `ingr_map.pkl` and a trained model will be the base of our final bot.

Model

The model was built on `PP_recipes.csv` in Python. Its aim is to find the closest recipe in the dataset based on a set of ingredients given by the user.
We decided to employ Doc2Vec[1], a machine learning model designed to represent entire documents as vectors. In this context, a "document" corresponds to a recipe, and its "words" are the list of ingredients. By training a Doc2Vec model on a collection of recipes, we can encode each recipe into a vector that captures the relationships and patterns among the ingredients. Recipes with similar ingredients are represented by similar vectors. The similarity between recipes can then be calculated by comparing their vectors using cosine similarity, a metric that computes the cosine of the angle between the vectors, which reflects how similar their directions are.
Once trained, the Doc2Vec model can be inferred to generate vectors for new sets of ingredients given by the user, by analyzing those ingredients in the context of the training data. After computing the vectors, we take the recipes in the dataset that are closest to the generated vector and suggest them to the user. Specifically, we consider recipes whose similarity is more than 90% times the similarity value of the best recipe.

Metrics

Along with the ingredients, the user can also indicate some characteristics of the recipe, such as calories, proteins (PDV), time, and many more. If the user does not want to insert the value for a specific metric, it will be set to a default number that corresponds to the median value of the metric in the dataset. Both the data and the new inserted values are standardized and absolute values are applied to prevent negative values and ensure accurate Euclidean distance calculations. The Euclidean distance is computed between the inserted metrics and the metrics of the subset of recipes chosen previously by the model. Finally, the five recipes from the latter subset with smaller Euclidean distance are shown to the user, including details such as ingredients, steps, a description, and the difficulty level.

Bot

Putting all this together, we created a Telegram Bot, with a nice interface that can perform the latter operations.
We created two versions of the bot:

- **ChefBot:** this version structures the chat in three steps.
  1. **Ingredient insertion:** the user inserts ingredients (separated by commas), which are then processed by the model and the closest recipes are computed;
  2. **Metrics insertion:** the metrics are inserted by the user, either by deciding to use the median value (by typing 'n' in the chat) or to employ a custom value. After being standardized with absolute values applied, the top five closest recipes are chosen in the previous subset using Euclidean distance;
  3. **Continuation:** if the user decides to ask for other recipes, we go back to point 1, otherwise the session ends.
- **InteractiveChefBot:** this version is analogue to **ChefBot**, but we have a nicer interface during the metrics insertion step. The user can click on two different buttons: **Use Default** or **Custom**. If the user clicks on the first one, the default median value will be selected. Contrarily, if the user clicks on the second one, he will be allowed to manually set a custom value.

Limitations

ChefBot has nice features, as well as some limitations. The main one is the dimension of the data: to train our model, we employed a dataset with 150,000 recipes, which is a reduced number compared to the vast cooking world. But, given the limited power of our machine, we opted to work with this type of data. As a result, the bot may sometimes suggest recipes that include only a subset of the ingredients specified by the user, rather than all of them.

References

[1] Q. V. Le and T. Mikolov, *Distributed representations of sentences and documents*, 2014. arXiv: 1405.4053 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1405.4053.

[2] S. Li, *Food.com recipes and interactions dataset*, Accessed: 2025-01-02, 2023. [Online]. Available: https://www.kaggle.com/datasets/shuyangli94/food-com-recipes-and-user-interactions/data.