Detecting Dietary Restrictions in Recipes

Human Language Technologies Project Report

Group 15

Irene Dovichi - i.dovichi@studenti.unipi.it
Marco Lavorini - m.lavorini2@studenti.unipi.it
Alice Nicoletta - a.nicoletta@studenti.unipi.it
Nicola Pitzalis - n.pitzalis@studenti.unipi.it

A.Y. 2023/2024

Contents

1 Introduction

This project aims to develop a classification system designed to identify and categorize recipes based on specific dietary labels. These labels include whether a recipe is vegetarian, dairy free, gluten free, low carb, low fat, and low sodium. We will explore machine learning techniques and natural language processing (NLP) methods to process and analyze recipes to determine their membership in these dietary categories.

1.1 Motivation

The ability to accurately identify and categorize recipes according to specific dietary labels, would not only enhance the user experience on culinary websites by accommodating diverse dietary needs, but also represent a significant step forward in personalized nutrition and inclusive technology. Moreover, this project highlights the intersection of technology and real problems by showing the application of language models in everyday life.

1.2 Data Source

The dataset used in this project is available on Kaggle at the following link: Food Recipe Dataset[1].

2 Literature Review

This literature review aims to provide an overview of the readings that we have found to be most relevant to our work.

In [2] the problem to be solved was to predict the types of cuisine from the ingredient lists. Although this article addresses a slightly different and simplified problem compared to ours, given that the ingredients of the dataset appear to be rather clean, it made us think about how to preprocess our data. It also made us reflect on the limitations of one hot encoding (ohe); in fact, it points out that even the same ingredient can be referred to in slightly different ways, perhaps due to the use of different adjectives, which can result in the ingredient appearing multiple times in the global list. This issue can significantly increase the dimensionality of the ohe and encouraged us to try alternative techniques.

We therefore investigated word embedding techniques. For everything concerning Word2Vec, we followed the approach shown in [3], where the aim is to overcome the problem that the user is unable to identify which dishes they can cook using the ingredients they have available. Another significant resource was [4], from which we got the idea of generating an embedding for a recipe by taking the average of its ingredients' embeddings.

In paper [5] they used a dataset that has some similarities to ours in terms of class imbalance, even if the labels concern the degree of difficulty of the recipe. This work was particularly insightful for our preliminary tests, where we worked with the BERT-generated embeddings by first taking the dense 768-dimensional representations for each token and then using a multi-layer perceptron for classification. The task presented was dealing with multi-class classification with an higly unbalanced dataset. Therefore, the paper was particularly significant for the evaluation part since we could see how micro-average is referred to as the primary metric in this case and the macro F1, precision and recall as additional evaluation metrics. Finally, this work could provide us with a starting point for future improvements: adding linguistic features to deep neural models. For example, the number of words that make up the preparation of the recipe, the number of ingredients, and the presence or

absence of discriminating words of some labels.

Regarding Named Entity Recognition, we found confirmation in [6] that using a fine-tuned model allows for much better results. Although the model used in the paper's experiments (FoodNER) is not exactly the one we used (FoodBaseBERT[7]), it appears that the two were fine-tuned on the same dataset: the FoodBase corpus[8].

3 Data

In this section, we analyze the dataset[1] used for our project. It contains 82.245 instances and 9 columns: `category`, `cooking_method`, `cuisine`, `image`, `ingredients`, `prep_time`, `recipe_name`, `serves`, `tags`. We immediately decided to eliminate most of the features both due to the large amount of missing values, and because they were superfluous for our task. We are therefore left with:

- `cooking_method`: the procedures necessary to prepare the dish
- `ingredients`: the necessary ingredients, including dosages and adjectives
- `tags`: a list of labels representing the recipe.

3.1 Tag Processing

The feature we had to work on the most was `tags`; in fact there are 765 different tags in the original dataset. Most of them are too generic, like '4th of July' and 'Kid Friendly', and some are ambiguous like 'Vegetable' that specifies if the recipe uses vegetables, but it is present also in dishes with meat and fish. That said, we first decided on the classes we wanted to use in our work: *Vegetarian*, *Dairy Free*, *Gluten Free*, *Low Carb*, *Low Fat*, and *Low Sodium*. Then, we had to manually identify the tags which, if present, ensure belonging to a certain class. For example, we identified *Vegetarian* recipes as those that presented at least one of the following tags: 'Vegan', 'Vegetarian', 'Vegetarian Meals'.

3.2 Definition of New Features

We then realized that considering the recipes that had the *Vegetarian* feature at 0 as Non-Vegetarian could present some issues. In fact, it may be that some recipes were written with the aim of having a user insert the ingredients available rather than a diet and therefore a recipe that is actually vegetarian does not appear as such but has the tags 'Cheese', 'Dairy Recipes' or 'Fruit'. In other cases, the speed and simplicity of the recipes is the focus of attention and therefore tags such as 'Easy' and 'Easy Side Dish Recipes' have been placed without further specifications.

We therefore decided to identify non-vegetarian recipes in a more direct way, defining the `Meat&Fish` feature. We identified all the tags relating to meat (98 tags) and fish (49 tags) and set the recipes that featured them at 1.

Regarding the *Vegetarian* feature, only around 11.000 recipes in the dataset are vegetarian. Since the desserts are considered vegetarian we decided to define the new `Vegetarian&Dessert` feature which allows us to go from 14.1% of vegetarian recipes to 32.9%. To do this we have similarly identified the tags referring to desserts.

To prove what was said at the beginning, we report in Table 1 the first 10 most frequent tags and their value count for recipes that have both `Meat&Fish` = 0 and `Vegetarian&Dessert` = 0.

Tag	Count
Gluten Free	10.185
Vegetable	9.589
Side Dish	5.336
Fruit	5.202
Cheese	4.280
Low Sodium	3.989
Easy	3.305
Low-Fat	3.187
Appetizer	2.789
Dairy Recipes	2.044

Table 1: Most frequent tags for `Meat&Fish` = 0 and `Vegetarian&Dessert` = 0 recipes.

3.3 Class Imbalance

Let's see what the distribution of our classes is in the dataset. From Figure 1 you can immediately see that adding desserts to the *Vegetarian* feature made it double. Moreover, the most frequent class is *Gluten Free*, while *Dairy Free* is the least frequent, with only 414 instances. The large imbalance between the classes will almost certainly make the correct prediction of the *Dairy Free*, *Low Carb*, *Low Fat* and *Low Sodium* labels problematic.

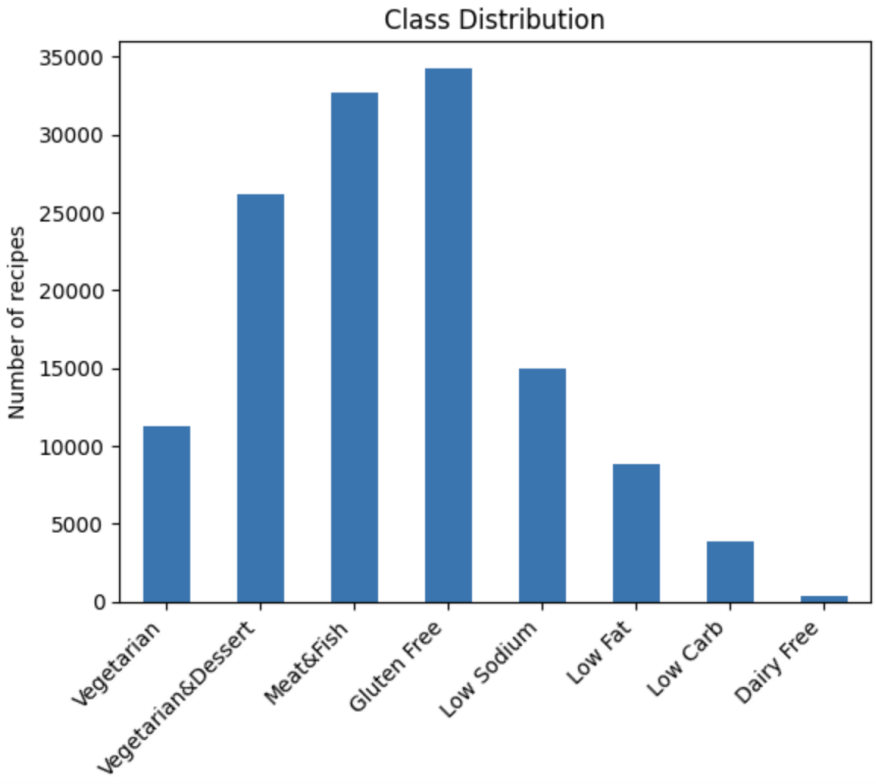

Figure 1: Class Distribution

3.4 Datasets Used

After deleting the null elements of `cooking_method` and `tags` and removing the equivocal recipes (`Vegetarian&Dessert` = 1 and `Meat&Fish` = 1), we are left with 79.617 recipes. For finetuning, we remove the recipes with more than 512 tokens (the maximum input length for BERT) and we take a balanced set: we keep all the `Vegetarian&Dessert` recipes, and we add the same number of recipes from the `Meat&Fish` class, getting a total of 50.902 instances. For the experiments with ohe and Word2Vec we use a balanced dataset of 10.000 recipes: half of them are `Vegetarian`, and the others are `Meat&Fish`.

4 System Overview

In this section, a brief description of the architectures used during the experiments will follow.

4.1 Random Forests, Support Vector Machines and Multi-Layer Perceptrons

We decided to start with standard ML techniques. First of all, we considered Random Forests to be an appropriate architecture for solving our task, given the (reasonable) assumption that recipes can be classified as vegetarian or non-vegetarian based on their ingredients. To be more specific, one might classify a recipe as non-vegetarian given that some specific ingredients are present, such as any type of meat. For this reason, Random Forest might seem a plausible option since they base their rules on ingredients. Both SVM and MLP are very powerful ML tools used for classification tasks. Hence, we chose to consider them as well, and we can find the chosen hyperparameters in Table 2.

Model	**Hyperparameters**
MLP	`alpha` = 0.01, `hidden_layer_sizes` = 100, `learning_rate_init` = 0.001
Random Forest	`criterion` = `gini`, `n_estimators` = 200
SVM	`C` = 1000, `gamma` = 0.001, `kernel` = `rbf`

Table 2: Hyperparameters for the models in the single-label classification.

Although the intuition that a recipe can be classified based on the presence or absence of certain ingredients is reasonable, the actual implementation faces several challenges. These include the retrieval of all relevant ingredients and their representation. To address the first problem, we can exploit a Named Entity Recognition (NER) model to extract the food entities, however we cannot expect to have perfect results for our task. Regarding the representation of the data, we might use one hot encoding to build up a matrix of encoded ingredients. As one can easily observe, two main issues arise here: there might be many ingredients that vary only slightly (e.g., *pepper* and *black pepper*) and should be treated as equivalent for our task. Additionally, the matrix size is directly related to the vocabulary size, which is not optimal.

4.2 Static Embeddings

Considering the expense in space and complexity to create the above-mentioned one-hot encoded vectors, we took the previous architectures and fed them with word embeddings learned by a Word2Vec model. In particular, the recipe embeddings (the actual input) were calculated by averaging their ingredients' embeddings, following the approach shown in [4]. The embedding size was fixed to 100, while the window of context was as large as the number of ingredients of the longest recipe, i.e. 43. Finally, the method chosen was CBOW. The resulting hyperparameters after a brief grid-search are

shown in Table 3.

For the multi-label classification task, the previous models were re-used and fit on each label for SVM and Random Forest, while for the MLP the output layer was replaced with a 6-units layer, each unit with sigmoid activation function and two hidden layers of size 64 and 16 respectively.

Model	**Hyperparameters**
MLP	`alpha` = 0.0001, `hidden_layer_sizes` = (64,32), `learning_rate_init` = 0.001
Random Forest	`criterion` = `entropy`, `n_estimators` = 200
SVM	`C` = 5000, `gamma` = 0.1, `kernel` = `rbf`

Table 3: Hyperparameters for the models in the single-label classification using static embeddings.

4.3 Recurrent Neural Networks and BERT-based architectures

The final RNN architecture for the single label task is represented in Table 4. For the multi-label task we use the same model, with the exception of the Dense layer at the end whose shape is modified to fit the number of classes to be predicted. The hyperparameters used for the model creation and for the training can be found in Table 5.

Layer	**Output Shape**	**#Param**
Embedding	(528, 512)	5.523.456
Bidirectional	(528, 2)	4.112
BatchNormalization	(528, 2)	8
AttentionLayer	(2)	530
Dense	(1)	3
Total params:		**5.528.109**
Trainable params:		**5.528.105**
Non-trainable params:		**4**

Table 4: RNN summary of the binary classification task.

Hyperparameter	**Value**
`embedding_dim`	512
`lstm_units`	1
`dropout`	0.3
`optimizer`	`adam`
`loss`	`binary_crossentropy`

Table 5: Hyperparameters used for RNN.

In addition to RNNs, we also experimented with the BERT model, in particular with the `'bert-base-cased'` model available on HuggingFace [9], since it has shown state-of-the-art performance on a wide range of NLP tasks. We loaded the pre-trained model with the `AutoModelForSequenceClassification` method of the class `transformer` and trained it with the hyperparameters shown in Table 6.

Hyperparameter	Value
`eval_steps`	250
`learning_rate`	1e-4
`per_device_train_batch_size`	16
`per_device_eval_batch_size`	16
`num_train_epochs`	5
`weight_decay`	0.05
`warmup_steps`	2
`metric_for_best_model`	`f1`

Table 6: Hyperparameters used for BERT fine-tuning.

5 Experiments

In this section, we consider the architectures presented before and report the experiments carried out.

5.1 Methodology

Starting with the standard ML architectures introduced in Section 4.1, as we anticipated, we had to deal with the exponentially increasing size of the one-hot encoding matrix. In fact, this size scales with the size of the vocabulary, that increases with the number of recipes. For this reason, we decided to use the balanced dataset of 10.000 instances.

For what concerns the NLP architectures of Section 4.3 we decided to exploit the semantic structure of text, referring to the `cooking_method` column. Since we are no longer constrained by the size of the ohe matrix, we use the balanced dataset of 50.902 instances.

As a bridge between the two approaches, we proposed standard ML models supported by static word embeddings with Word2Vec. Even if the size of the embedding space is reduced with respect to ohe, we worked with 10.000 recipes in order to compare the results with those of the architectures in Section 4.1.

5.2 Standard ML approaches

As mentioned before, the first step was to encode the data to reformulate our task as a standard Machine Learning problem. Specifically, this involved ensuring that the inputs could be fed into the input layer, all having the same size, i.e. with ohe. The problem, as we can see from the Figure 2, is that our lists of ingredients are not clean: they carry too much information. This causes even the same ingredient to appear many times with different adjectives or dosages, resulting in multiple encodings and a larger vocabulary. To address this issue, we decided to use NER models that have been trained on food recognition to extract a cleaner list of ingredients.

NER for food entities

We tried two different NER models; the first[7] was a BERT based model, while the second[10] was based on DistilBERT. The first model extracted the ingredients from the `cooking_method` column, while the second was directly given the `ingredients` column as suggested in the documentation. In fact, the finetuned DistilBERT is able to extract food entities from lists of ingredients that precisely

resemble our dataset's column. The overall performance of the DistilBERT were better since the obtained vocabulary was smaller and the ingredients were cleaner.

```
 ['2 baking potatoes, like russets, thinly sliced',
'2 tablespoons olive oil',
'Salt and pepper',
'2 bunches fresh spinach',
'1 cup bulghur',
'1 cup hot water',
'1 orange',
'1/2 cup walnuts',
'1 tablespoon olive oil']
```

Figure 2: List of the ingredients of a recipe without processing.

Processing NER's Results

Since the output of NER models appeared to be improvable, we processed more the data. Using some regular expressions, we remove words with less than 3 characters,numbers and special characters, and fixed the blank spaces. We referred to SpaCy[11] for the finetuned BERT, and to Stanza[12] for the DistilBERT to address tokenization, PoS Tagging and lemmatization. We removed the adjectives, verbs and proper names from each ingredient, and kept the lemmatized form of the words.

Classification on Extracted Ingredients

At this point the vocabulary of all the ingredients is ready and we can proceed with the one-hot encoding of the recipes. The encoded vectors have length equal to the size of the vocabulary, and each component refers to the corresponding ingredient. Each ingredient will have 0 in every position other than its own, and a 1 in its position in the vocabulary. We now define the global encoding of a recipe as the sum of the encodings of its ingredients. In this way the order of the ingredients does not modify the encoding of the recipe. The final ohe matrix will be of dimension $N \times V$, where N is the number of recipes and V is the size of the vocabulary. This matrix will be the input of the models described in Section 4 used to predict the label *Vegetarian*. We explored the hyperparameters' space using a regular grid search.

5.3 Embeddings with Word2Vec

Our second approach was to fit a Word2Vec model on the list of the ingredients for each recipe. The context window was set as the length of the longest list, so that each ingredient has as context every other ingredient in the recipe. The chosen architecture for the model was the Continuous Bag of Words (CBOW), given that the order of appearance for the ingredients is not relevant. The size of the embeddings was set to 100; larger sizes have also been tried, but they have not led to any significant improvement. Finally, the embedding for a recipe is taken as the average of its ingredients' embeddings.

The obtained embeddings were used as input to the standard ML models both for single and (for the first time) multi-label classification, as it was less computationally demanding than the previous approach. Only the MLP was able to perform the multi-label task, since the other models are designed for single label classification. To solve this problem, we used Scikit-learn's `MultiOutputClassifier` function, which takes as input a model and fits one classifier per target.

5.4 NLP-driven approaches

After some experimentation with the standard tools of ML, we mainly explored two strategies more familiar to the NLP world, specifically RNNs and encoder-only transformer. The choices are motivated by the goal of the task, i.e. sequence labeling, for which both are very suited.

Recurrent Neural Networks

We experimented with different RNN architectures, starting from a standard bidirectional LSTM, to which we then added an attention layer. While the latter worked slightly better, the training time took much longer (given the incredible amount of dot products the layer had to perform). We also explored different configurations for the network, using a regular grid search, increasing both the number of LSTM units and the size of the embedding space. Given the simple structure of our problem, very shallow networks worked perfectly.

BERT Finetuning

Lastly, we proceeded with the fine-tuning of BERT. The training process followed a standard approach, using the hyperparameters of Table 6 both for single and multi-label classification. The hyperparameters were selected based on a small grid search, where different combinations of values were tested and analyzed to identify the optimal settings. An extensive grid search was not performed due to the costly training time. Furthermore, a check-point approach was used to save both the best and last model weights, in order to have the configurations available. The F1 score was chosen as the metric for selecting the best model to ensure a balance between false positives and false negatives since in our task both types of errors are significant.

6 Results and Analysis

In this section we will report the results obtained for the experiments described before. As mentioned in Section 3 we work with two different datasets, one of 10.000 instances and the other of 50.902. The datasets were splitted before the experiments leaving a 30% out for testing. For the results we will only compare the models that worked with the same dataset, and the performances in the tables refer only to the (same) test set. For binary classification the results shown are the average of the two classes.

Tables 7 and 8 show the results obtained through the standard ML approaches for single and multi-label classification respectively. The models worked both with the ingredients encoded via one-hot encoding and Word2Vec. We can observe how the classifiers scored decently in the single-label classification, but did not achieve comparable results in the multi-label task. This outcome is likely due to the highly unbalanced dataset and the models not being specifically designed for multi-label classification. It also indicates the necessity of deploying more sophisticated NLP-specific models for both representation and classification to improve performance.

Tables 9 and 10 exhibit the results obtained with both attention-based RNN and BERT fine-tuned for our task. As we can see the models perform very well on the binary classification with BERT having the best results. For the multi-label task we observe that the performances worsen; this happens due to the imbalance in the classes, as noted in Section 3.

Model	Accuracy	F1	Precision	Recall
MLP	90	91	91	90
Random Forest	87	88	88	87
SVM	91	92	92	92
MLP - Word2Vec	77	77	78	77
Random Forest - Word2Vec	77	77	78	77
SVM - Word2Vec	81	82	82	82
Logistic Regression - Word2Vec	75	75	75	75

Table 7: Single-label task on the reduced dataset (10.000 instances).

		Micro			Macro		
Model	**Accuracy**	**F1**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**
MLP - Word2Vec	41	67	68	68	49	48	48
Random Forest - Word2Vec	48	77	67	72	66	45	51
SVM - Word2Vec	43	75	62	68	50	38	43
Logistic Regression - Word2Vec	44	74	66	70	50	40	44

Table 8: Multi-label task on the reduced dataset (10.000 instances).

Model	Accuracy	F1	Precision	Recall
RNN	96	96	95	97
BERT	97	97	98	98

Table 9: Single-label task on the balanced dataset (50.902 instances).

		Micro			Macro		
Model	**Accuracy**	**F1**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**
RNN	62	80	86	75	41	51	39
BERT	74	88	90	86	66	72	62

Table 10: Multi-label task on the balanced dataset (50.902 instances).

We can justify that the micro-average is much higher than the macro-average with the fact that our classifiers perform well on the larger classes: `Vegetarian&Dessert` and `Gluten Free`, but give poor results on the smaller ones. Since micro-average treats each instance equally, it tends to be dominated by the performance on the most frequent classes; on the other hand, the macro-average treats each class equally regardless of its frequency, and poor performance for minority classes causes it to be lower.

7 Conclusions

During this project we were able to experiment with textual data, and we saw that the use of NLP methods allowed us to solve our task more satisfactorily than standard Machine Learning models.

Fine-tuning BERT on our dataset allowed us to be highly accurate for classes with a large number of instances.

Unfortunately the dataset's quality was the limiting factor, as the labels were not provided by default but an effort had to be made to create them (by exploiting the `tags` column of the original dataset). In this regard, recall that in Table 1 you can see that many recipes were neither `Vegetarian&Dessert` nor `Meat&Fish`, and some had both labels assigned, which is definitely not desirable.

A clear advantage of NLP models is the reduced need for extensive data pre-processing. The standard models heavily rely on the encoding of the ingredients, which themselves rely on other ML techniques (e.g. NER for ingredient extraction). These encodings have inherent limitations, for instance, it is a problem to deal with out-of-vocabulary words. Moreover, using Word2Vec can result in a loss of information when creating recipe embeddings by averaging the ingredients. This approach can place two recipes very close by chance, even though they do not share any ingredient, or even be mapped far away due to some ingredients being distant.

In conclusion, while NLP models showed great results, we observed that some ML techniques were still a valid choice, with the adequate amount of data pre-processing and the right approach to feed the data to the models. Future developments should aim to improve the reliability of the dataset and explore more advanced methods for feature extraction, including not just the ingredients but also other lexical information. By addressing these challenges, we can improve the quality of our work to achieve a more accurate and reliable classification.

References

[1] Snehal Lokesh. *Food Recipe Dataset*. https://www.kaggle.com/datasets/snehallokesh31096/recipe/data. Version 1. Retrieved 18/03/2024. 2020.

[2] Richard Ye. *Classifying recipes using NLP and Logistic Regression*. https://medium.com/the-power-of-ai/classifying-recipes-using-nlp-and-logistic-regression-40934ef0ece3. Accessed 29/05/2024. 2023.

[3] Ajit Rajput. *Exploring Food Recipes using Machine Intelligence*. https://medium.com/@ajit.rajput/exploring-food-recipes-using-machine-intelligence-87cb1a983cfd. Accessed 30/05/2024. 2020.

[4] Jaan Li. *food2vec - Augmented cooking with machine intelligence*. https://jaan.io/food2vec-augmented-cooking-machine-intelligence/. Accessed 30/05/2024.

[5] E. Mohammadi, N. Naji, L. Marceau, M. Queudot, E. Charton, L. Kosseim, and M.-J. Meurs. "Cooking Up a Neural-based Model for Recipe Classification". In: *Proceedings of the Twelfth Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, May 2020, pp. 5000–5009. URL: https://aclanthology.org/2020.lrec-1.615.

[6] A. Wróblewska, A. Kaliska, M. Pawłowski, D. Wiśniewski, W. Sosnowski, and A. Ławrynowicz. "TASTEset - Recipe Dataset and Food Entities Recognition Benchmark". In: *arXiv preprint arXiv:2204.07775* (2022).

[7] Dizex. *FoodBaseBERT-NER*. https://huggingface.co/Dizex/FoodBaseBERT-NER. Retrieved 18/03/2024. 2023.

[8] G. Popovski, B. K. Seljak, and T. Eftimov. "FoodBase corpus: a new resource of annotated food entities". In: *Database* 2019 (2019). URL: https://doi.org/10.1093/database/baz121.

[9] J. Devlin, M. Chang, K. Lee, and K. Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[10] Charlene Chambliss. *FoodBERT: Food Extraction with DistilBERT*. https://github.com/chambliss/foodbert. Retrieved 18/03/2024. 2020.

[11] M. Honnibal and I. Montani. "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing". To appear. 2017.

[12] P. Qi, Y. Zhang, Y. Zhang, J. Bolton, and C. D. Manning. "Stanza: A Python Natural Language Processing Toolkit for Many Human Languages". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. 2020. URL: https://nlp.stanford.edu/pubs/qi2020stanza.pdf.