Dipartimento di Ingegneria e Scienza dellInformazione

– KnowDive Group –

KDI Lab 2018-19

Document Data:	Reference Persons:
16/12/2018	Enrico Agrusti
	Mattia Buffa
	Denis Gallo

Index:

1 Scenarios and Personas description

Slurpy! is a new and innovative service that allows people to interact proactively with new kitchen technologies like smart fridges. It was never this easy to filter on which food you have at home and which ingredients you need to prepare some recipes. Discover new recipes from the catalog or use it to remind yourself about the right procedure while cooking. Dont let your food expire and get notifications on what needs to get used before expiring, this is done by informing your fridge about the expiring date when adding something new into it. Another great feature of Slurpy! is the possibility of selecting a weekly food plan to follow, specifying the cheat days and the limits in calories and variety.

Paola is a 34 years old secretary and in her free time she loves to make cook challenges with her best friends. This challenge is about inviting the others to your place, cooking something special and receiving a rating from everyone about the food. Repeat until everyone cooked at his place once. Slurpy! is not only a great tool for discovering new recipes, it also helps you while youre at the supermarket. Paola has the challenge dinner this weekend and shes currently going to buy everything that is needed. She still has no idea what to prepare and shes not even sure about what she has currently in the fridge. A simple look at Slurpy! might be enough for a great dinner. She has a feeling that vanilla could be a key ingredient, so she filters for recipes containing vanilla and voil: Mango-Cango Chicken. With a single click on the recipe, Paola can also identify what she already has at home and what she needs to buy since her smart fridge stores information on what she bought in the past and what has been used. Now nothing can stop her winning for the second time the cook challenge.

Giovanni is a 23 years old offsite student who loves being creative in the kitchen. However, in the evening he becomes a little bit lazy after several hours at the university. His roommates Marco and, especially, Katia often buy a lot of food which afterwards needs to get eaten before expiring. After a couple of messages on Whatsapp, Giovanni decided to cook dinner for everyone by using what is left in the flat. By getting the information from the fridge, he can search for recipes based on some specific ingredient that is expiring, trying to maximize the amount of ingredients that are already at home. Zucchini dinner is the best hit. Fortunately, the supermarket is on Marcos way, so Giovanni can inform him to buy some Parmesan Cheese since its the only ingredient missing for the recipe.

Cinzia, 47 years old, owner and cook of a restaurant, wants to increase the customers by adding a deal for lunch with the typical Italian fixed launch menu. Remembering that the highest wasted costs of a restaurant is thrown food because of expiry date, she wants to offer a menu with those ingredients she needs to use as soon as possible. Therefore, every day she searches for recipes based on expiry dates of what she has in the fridges. Obviously, it is not possible to get every day the perfect recipe including every ingredient that is expiring. This means that Slurpy! needs to take a decision. Cinzia might want to free the most space in the fridge by preferring recipes containing the most ingredients that are expiring or she wants to save as much money as possible, preferring the most expensive ingredients. After this choice, Slurpy! returns a list of recipes that Cinzia can use for her lunch menu.

Giulio is 33 years old and works as a fitness trainer. His job is also his lifestyle; thus, he follows a strict diet with an organized daily meal program. In Slurpy! he can add such a plan in order to see everyday what he needs to eat and if he has everything in his kitchen. On Sundays, instead, he can eat whatever he wants, as long as he stays in between his caloric limits. With a simple request, Giulio can search for recipes based on calories and based on something he particularly wants to eat or based on what he already has in the fridge. He loves his new smart kitchen and already advises it to his customers too.

Paolina is a bright 86-year-old lady that gets often invited for dinner to her sister Olga's house. Both Olga and Paolina are very big fans of Slurpy! and sometimes it happens that Olga asks to Paolina to cook, because she has not come home yet. Even if Olga's fridge does not belong to Paolina, through an authentication and authorization system it is possible for Paolina to connect her Slurpy! to Olga's fridge, and exploit the full potential of our app with a foreign fridge.

2 Queries description

Persona	Generalized Query	Answer
Paola	Give me all the recipes containing vanilla	A list of recipes with vanilla
Paola	Give me all the recipes containing vanilla, ordered by price (include already owned ingredients)	A list of recipes with vanilla, ordered by an approx. calculation of the total cost of the recipe minus the cost of the already owned ingredients in the fridge
Paola	Give me all the recipes containing vanilla, ordered by least missing ingredients	A list of recipes with vanilla, ordered by the % of missing ingredients of each recipe
Paola	Give me all the recipes containing vanilla, ordered by price	A list of recipes with vanilla, ordered by the approx. calculation of the total cost
Paola, Giovanni, Cinzia, Giulio	Give me a specific recipe like Pineapple Rice Bake	A list of ingredients, a step by step guide, calories information, a list of ingredients that are missing, a list of ingredients that are already in the fridge (with a special symbol if expiring)
Giovanni	Give me all the recipes containing a specific ingredient, ordered by maximizing already owned ingredients	A list of recipes with the specified ingredient, ordered by the number of already owned ingredients in the fridge
Giovanni	Give me ingredients in my fridge that are expiring	A list of ingredients in the fridge that are expiring
Cinzia	Give me all the recipes containing expiring ingredients in the fridge, ordered by number of ingredients	A list of recipes that contain expiring ingredients already owned, ordered by the number of expiring ingredients in the fridge that are included in the recipe.
Cinzia	Give me all the recipes containing expiring ingredients in the fridge, ordered by the cost of expiring ingredients	A list of recipes that contain expiring ingredients already owned, ordered by the total cost of expiring ingredients that are used
Giulio	Give me all the recipes, filtered by a min and a max of total calories	A list of recipes that have total calories between the set minimum and maximum
Giulio	Give me all the recipes containing a specific ingredient, filtered by a min and a max of total calories	A list of recipes that contain a specific ingredient and that have total calories between the set minimum and maximum
Giulio	Give me my saved food plan	A list of possibilities for recipes that follow (strictly or with a margin, with respect to what Giulio had selected) the weekly plan of Giulio.

3 Dataset extraction, generation and description

For the extraction of information, 2 websites and 1 existing dataset were used. The hardest part of this task was to find existing datasets with enough data and information needed for the described scenario. A lot of them were small despite the fact of having a lot of information. The bigger ones usually had less features but, at least, a lot of rows. By searching websites, instead, there were even more constraints. A website needs to be as static as possible since web scraping through dynamic websites is a lot more sophisticated, especially figuring out the URLs of each recipe. In addition, it is easier if for each recipe link there is a simple XPath query to get a specific information from it. During the research of cooking websites, some incredibly weird html structures were discarded because XPathing or analysing URLs were quite difficult.

The existing dataset is taken from Kaggle[1], a data repository for data scientists which contains a lot of different datasets. This one, in particular, is called Epicurious[2] since it is scraped from the well-known cooking website[3]. There are 20000 recipes containing various information like a title, some calories/protein/fat information, some guidelines on how to cook the recipe and a list of ingredients. This dataset was simply and directly downloaded as json file. Then it was analyzed and well-formatted with a simple json formatter for readability.

The other 2 datasets are taken directly from websites through a procedure of web scraping. A python script[1][2] scrapes through a website by collecting the links of every recipe, then it analyzes asynchronously every page through XPath queries to collect various information about each recipe: title, directions, ingredients and so on. The two scraped websites are Whatscooking[4] and TheKitchn[5]. For each of the two websites the python scripts are very similar, with some adjustments to collect every link of recipe and some adjustments about the XPath queries. In both cases a json dataset is created which is stored locally. These two datasets have 1226 and 999 recipes respectively.

A python script[3][4][5] is used to import a json dataset into Rapidminer. This is the most suitable procedure[6] found out, since Rapidminer is not the user-friendliest application in importing non-csv/xml files. By translating a json file into a pandas dataframe, Rapidminer can easily read the resulting dataset.

Some cleaning and refinement were already done during the extraction procedure, but this is explained later in the next chapter. The resulting datasets contain the data shown in the table below:

Epicurious	Thekitchn	Whatscooking
• title	• title	• title
• directions	• directions	• directions
• description	• description	• description
• a list of ingredients	• a list of ingredients	• a list of ingredients
• calories	• calories	• calories
• proteins	• proteins	• proteins
• sodium	• sodium	• sodium
	• fat	• fat
	• other nutritional info and labels	
• user rating from 1 to 5		
	• # of servings	• # of servings

The lists of ingredients are structured in a different way across the 3 datasets. In Epicurious they are a not

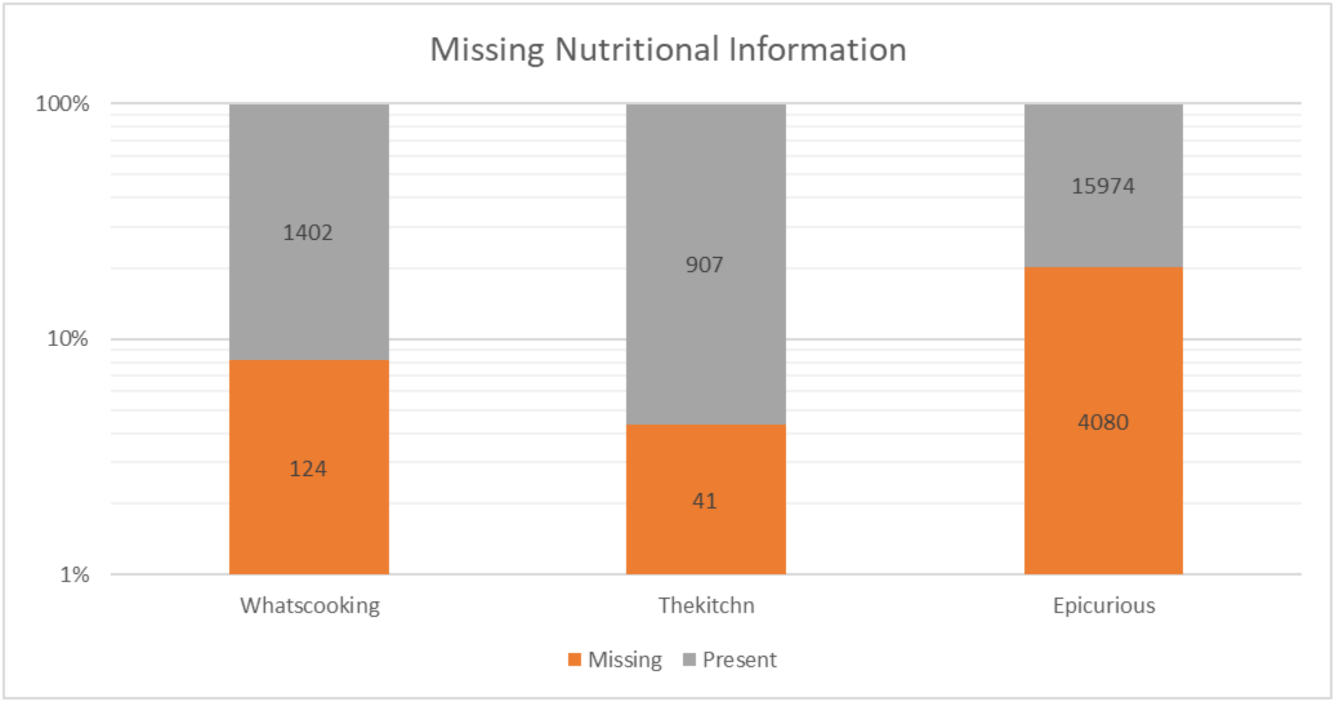
[1]sources/whatscooking/whatscooking.recipes.scraper.py
[2]sources/thekitchn/thekitchn.scraper.py
[3]sources/thekitchn/thekitchn.importer.py
[4]sources/whatscooking/whatscooking.recipes.importer.py
[5]sources/epicurious/formatted_recipes.importer.py
[6]output/whatscooking.storing.rmp

well-formatted list with ingredients and quantities in the same string. In Thekitchn each value of the array is well-splitted into a number, a quantity unit and an ingredient. Whatscooking is similar to Thekitchn with the number and the quantity unit in the same value, while the ingredient is detached. Transforming these ingredient lists into a well-formatted table with the right quantities, quantity unit and type of ingredient is shown in the next chapter since this kind of data refinement was done after the extraction of datasets.

4 Datasets cleaning, merging and analysis description

The collected datasets were scraped or downloaded directly from the respective source. This means that a lot of junk like useless information, html code, unicode/ascii tags, incomplete data were introduced.

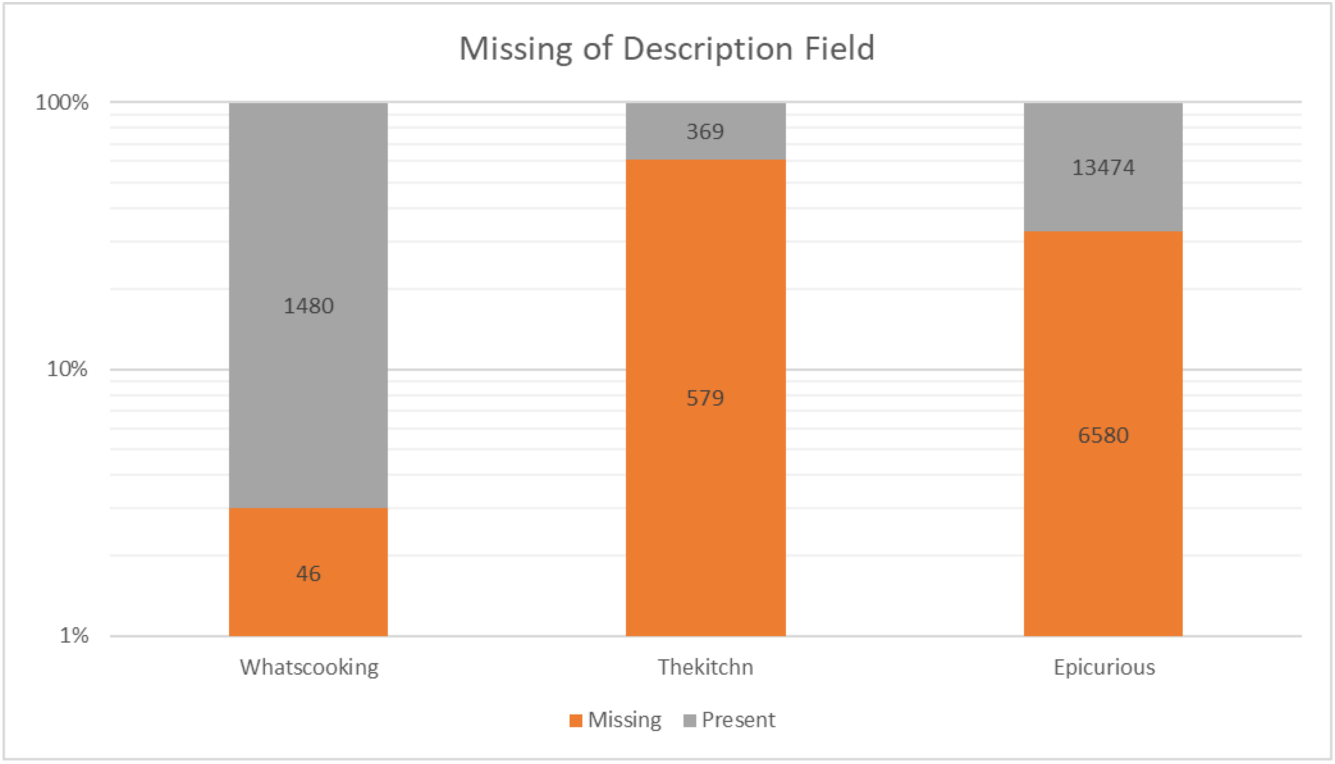
In the Whatscooking dataset every "\u00a0" and "\n" in the description were replaced with white spaces or empty spaces and then merged into a single string representing the entire description for the given recipe. The servings and calories values were transformed or at least treated as integers, every other nutritional value, instead, was saved as float. In whatscooking also an estimated cost is given for each recipe as column. This cost was normalized into a value from 0 to 100. There is a high chance that this field will be removed entirely during the delivery of the project since it is pretty pointless. The directions of a recipe is slightly messed up by unicode special characters, which needed to be removed. Therefore, every "\u00a0" was replaced by a white space and every "\n" was used to split a direction from the following one, generating a complete array of strings, each containing a direction for the given recipe. It is explained later how ingredients were processed since it is a main part of the cleaning and should be compared with the other datasets as well.



The Thekitchn dataset, similarly to Whatscooking, is a obtained by scraping a website. Therefore, columns like calories needed to be saved as integers and other nutritional values as floats. The description was scraped as a list of strings, so every string got merged into a single one with "<br>"s instead of "\n"s since Rapidminer doesn't like Unicode characters. In the ingredients list, some "null" entries were detected. So, they were removed and if no ingredient remained, the entire recipe got deleted. Going even further, some recipes had only a title, without any other information: these were removed as well.

The Epicurious dataset is the one that was downloaded from Kaggle. The entries were already cleaned, besides of unicode characters in the description and some recipes containing only a title and nothing else. So, basically, the cleaning and data adjustments were similar to the previous ones.

The ingredients are stored in every dataset as an array of "quantity in number", "unit" and "ingredient". Unfortunately in a couple of datasets this information is not well-formatted: quantity and unit could be in the same value or, even worse, everything is in the same value. In addition, keeping the ingredients in an array inside the table of recipes is inconvenient for different reasons. The idea is to refine the ingredients in a more logic way and to save these in a separated table, with a link to the original recipe. It can be easily seen as a one-to-many table with an id refering to a recipe and 3 values refering to the quantity of an ingredient (with its unit). With some regular expressions all these ingredients information were properly formatted to fit into the desired structure "quantity", "unit", "ingredient". It could happen that ingredients suggests to use "some of this ingredient OR some of that". For simplicity only the first choice was taken for every of these situations. An example could make these example clearer: if a recipe contains id, description, some nutritional information, a number of servings and an array of ingredients in the form of "200 g of pasta", in the new generated ingredients table[7][8][9] there are several rows for every ingredient of this recipe, one of these is id and "200", "g," "pasta".



Cleaning happened for the most part during the importing from the local json file to Rapidminer, so basically in the importer script in python. An id was created for each entry as well. To recognize from which source each recipe come, the id is "ep"+seq_no for the Epicurious dataset, "wc"+seq_no for Whatscooking and "tk"+seq_no for

[7]output/whatscooking.ingredients.xlsx
[8]output/thekitchn.ingredients.xlsx
[9]output/epicurious.ingredients.xlsx

Thekitchn.

Moving to the analysis of data, it is important to talk about the generation of a label containing the ”type of cuisine” of a recipe. Since this relatively type of information is missing in every dataset, a machine learning algorithm is used to predict this information. In this chapter the data analysis will be described, while the process and the training set will be shown in the data integration chapter, which is more appropriate for this kind of manipulation.

After the described cleaning, in the Whatscooking dataset[10] 1526 recipes are stored. Only 3% of these are without a description, 308 rows dont have an estimated cost (even though this information isnt really relevant in this project), around 100 recipes dont have complete information about nutritional specs (like calories, proteins, ...) and only two rows are missing directions on how to prepare the recipe.

In the Thekitchn dataset[11] there are a lot of missing values. In the 948 recipes, 61% do not have a description, no one has an estimated cost and 41 rows have missing servings or nutritional information. Only one recipe has no directions.

Epicurious is the dataset[12] with 20054 recipes. 33% of them have missing description, servings and estimated cost are not present at all, about 4000 rows have missing values in the nutritional information. Only eleven recipes dont have directions.

[10]sources/whatscooking.recipes.xlsx
[11]sources/thekitchn.recipes.xlsx
[12]sources/epicurious.recipes.xlsx

5 Related ontology proposal

In this chapter three ontologies are proposed with some basic evaluation based on the delivered data described until now. Before going into details, the data structure of the output datasets is shown.
Recipe contains the following properties: name, description, servings, cost, calories, fat, saturated, carbohydrates, fiber, sugar, protein, sodium, directions, ingredients, cuisine type. Ingredient contains only: quantity, unit, name.

The first ontology[13] has been taken from schema.org[6]. As usual, by just using a standardized ontology there are a lot of fields that are not covered. Therefore, it's a little bit too complicated to represent the described data structure. Especially a lot of information and structure for NutritionalInformation are proposed such as calories, fat, sugar, protein and so on. For the preparation time, four different structures are contained in the ontology: preparation time, cooking time, perform time and total time. Using this ontology the coverage of the data structure is 17/29 properties, which is 58,6%. The 17th property is the cuisine type which is explained in the next chapter since it was added later on.

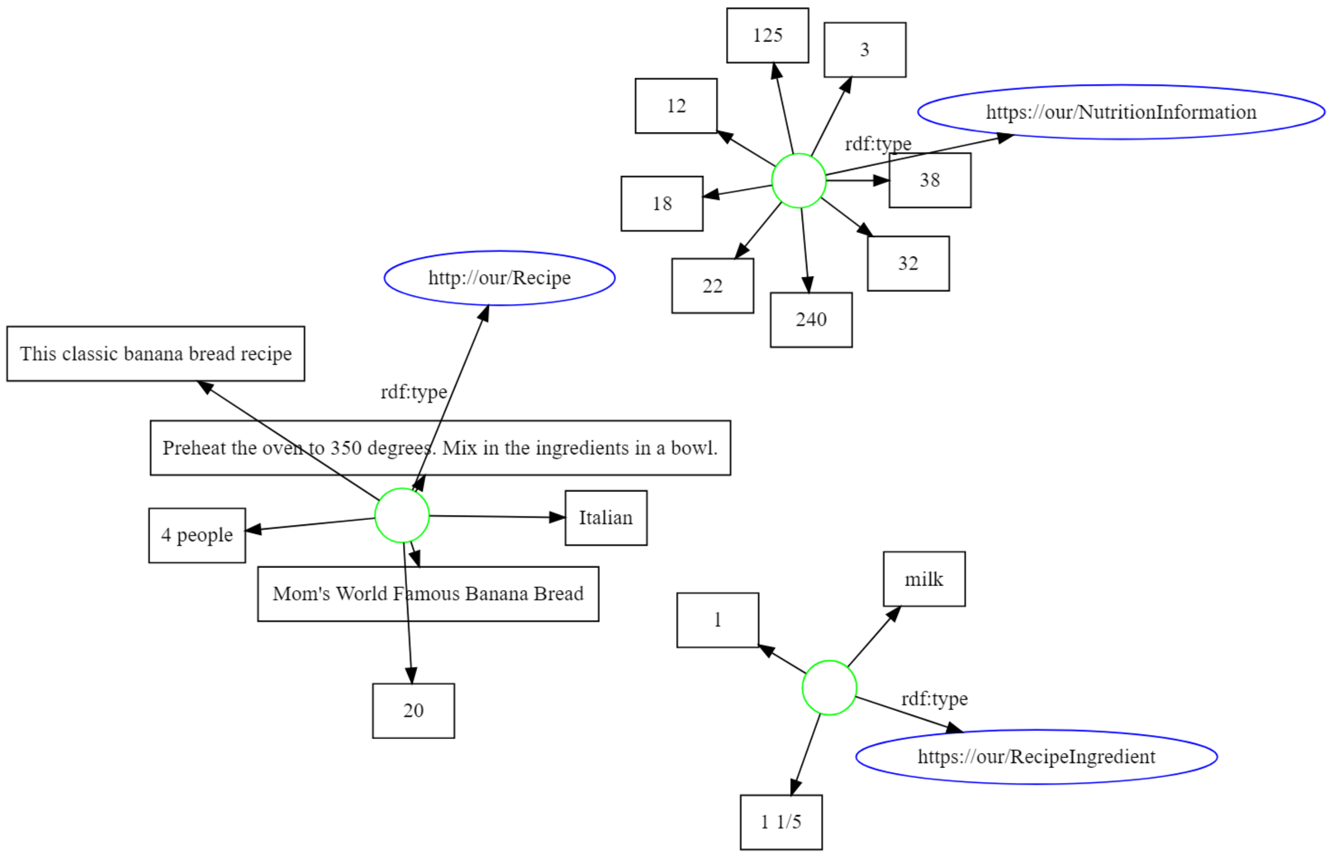

125
3
12
https://our/NutritionInformation
rdf:type
38
18
32
22
240
http://our/Recipe
This classic banana bread recipe
rdf:type
Preheat the oven to 350 degrees. Mix in the ingredients in a bowl.
4 people
Italian
Mom's World Famous Banana Bread
milk
1
rdf:type
20
https://our/RecipeIngredient
1 1/5

Figure 1: A coarse-grain graph example of the second ontology (field labels are missing)

The second proposed ontology[14] is modelled directly based on the data structure described above. The NutritionalInformation part has been simplified and the recipeIngredient field has been extended, splitting it into ingredientQuantity, ingredientUnit and ingredientName. It was particularly important since the ingredients data were not compatible with the first ontology. The coverage is for obvious reasons total for all 17 properties (including

[13]ontology/schema.rdf.txt
[14]ontology/our.rdf.txt

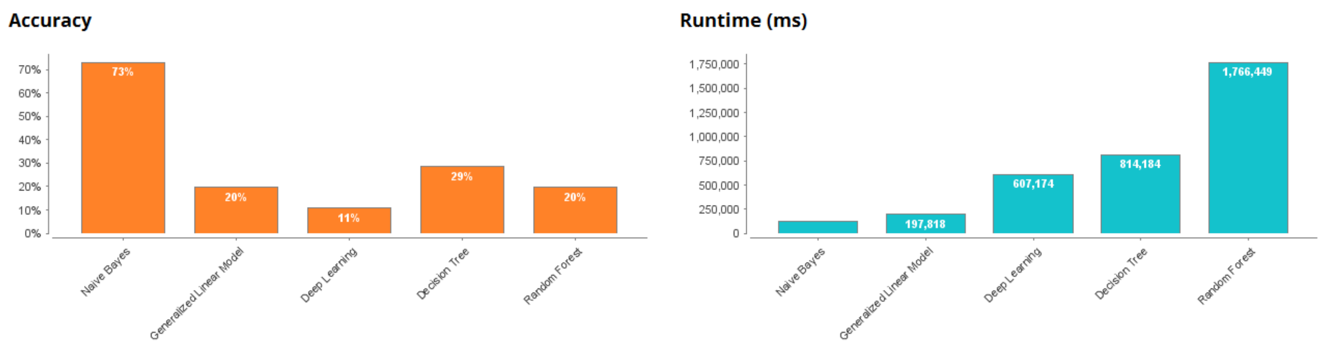
cuisine type, explained in the next chapter).

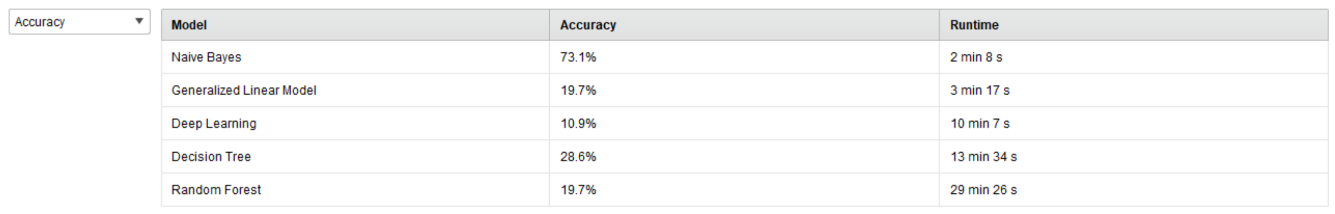
The third ontology[15] is an hybrid ontology between the previous two. It's more like an idea ontology, the one that every group member of the project had in mind while searching and extracting data. Starting from the second ontology, it keeps the ingredient structure with the three properties and adds some fields from schema.org that are quite important but weren't present in the input sources.
The added information are: cookTime, cookingMethod, recipeCategory, isSuitableForDiet, author
Using this ontology the coverage of the data structure is 77,3% (with cuisine type).

Talking about ranking or evaluation of these three ontologies it is simple to conclude that the second is the best to represent the output data structure, followed by the third and the first. The scope of this project, however, is to encourage to fulfill the missing data with further data integration. Therefore, it is suggested to start directly from the third ontology to move towards a real application for smart fridges/kitchens. It needs to be reminded that initially even "cuisine type" was an added information but it's integration was implemented and, therefore, treated as covered field.

6 Related datasets proposal and integration example

The integration with smart fridges is neither required in this project nor feasible since it is a fresh technology and still a niche. Nevertheless, adapting the data structure from an eventual smart fridge to the one of the ingredients table, could already be enough to store the quantity, the remaining quantity and the expire date of ingredients. The technology to store items in a smart fridge is left to the relative producer. As for the recipes part, if new ingredients need to get inserted to the system, it should be enough to adapt the data structure of those to the already existing data provided in this project. A very important information that is not present at all in the source data is the cuisine type (French, Japanese, Thai, ...). A useful way to get this information is provided now.





Model	Accuracy	Runtime
Naive Bayes	73.1%	2 min 8 s
Generalized Linear Model	19.7%	3 min 17 s
Deep Learning	10.9%	10 min 7 s
Decision Tree	28.6%	13 min 34 s
Random Forest	19.7%	29 min 26 s

It is possible to predict the cuisine type given the ingredients. To do that, a training set is required, containing for a set of ingredients a type of cuisine. Such a training set with 39774 examples was found on Kaggle. In Rapidminer,

[15]ontology/hybrid.rdf.txt

auto-modelling was used[16] with different kind of models, like nave bayes, generalized linear model, deep learning, decision trees and random forests. After training these models on the training set itself with a classic splitting into train and test set, the nave bayes got a hugely higher accuracy than decision trees (around 75% against 20%) as shown in the figure above. The ingredient table is the one that needed to be used to predict the cuisine type of recipes. It got adapted to the training dataset and then the prediction algorithm got started. It is not possible to have an accuracy of the results, but by checking manually a couple of predictions, it seems to work quite well. This procedure will be explained more precisely now.

The training set[17] is a json file containing recipes with an id, a cuisine type and some ingredients. This data is transformed into a table that is imported in Rapidminer. The table has every ingredient contained in the whole dataset as columns. Every row, instead, is a recipe with an id, a label and has a Boolean value for each ingredient. Since a table of 39774x6716 elements is very large, the python importer script[18] divided this task into four different batches. After the previously described auto-modeling, the ingredient table of this project needed to be adapted in order to be used for predictions. Thus, the same columns of the training set table are adopted, and each ingredient of each recipe is compared to every column using the Levenshteins distance (string similarity). If there is an ingredient with a resulting distance less than a threshold (in this case five), then the column value is set to true; if there are more, instead, the one with the lowest similarity is chosen. This way, the small inconsistencies between ingredient names are not a problem among the different data sources. Now, prediction is possible by using the generated pipeline of the auto-modeling in Rapidminer. The only issue arisen is that the pipeline automatically splitted the training set in order to produce a test set, but it was not the goal of the application. So, with some tweaking, the just described adapted ingredient table is inserted into the pipeline. Repeating this last step of the procedure with all three datasets[19][20][21] lead to cuisine type predictions for every data source.

7 Final considerations and open issues

This project is aimed to provide a dataset usable for the implementation of such an application described in the first chapter. Smart fridges and smart kitchens are still a fresh technology with high costs and a lot of research in front. A suitable way to use the provided data is up to the producers of such smart devices. The scenarios included features such as expiring dates, quantities of owned ingredients, food plans which are important fields and values that need to be modelled and integrated. A way to integrate information is provided in the chapter above but every type of integration task has obviously its own procedure.

The core part was about preparing data, which means that data extraction/generation and cleaning/merging were primary tasks while ontology proposals and integration examples were just a coarse-grained overview. Rapidminer is a great tool for manipulating information in various ways, but it also has huge limitations when the initial dataset is not in a csv format. The same goes for web scraping which is possible in Rapidminer but is a lot more intuitive with a simple ad-hoc python script.

[16]output/whatscooking.automodeling.rmp
[17]sources/training/train.json
[18]sources/training/machine_learning.importer.py
[19]output/whatscooking.predicted.xlsx
[20]output/epicurious.predicted.xlsx
[21]output/thekitchn.predicted.xlsx

References

[1] https://www.kaggle.com/.

[2] https://www.kaggle.com/hugodarwood/epirecipes/home.

[3] https://www.epicurious.com/.

[4] https://whatscooking.fns.usda.gov/.

[5] https://www.thekitchn.com/.

[6] https://www.schema.org/.